MECHATRONIC SYSTEM PARAMETER IDENFITCATION VIA GENETIC ALGORITHMS

by

Mathew I. Adamson
Bachelor of Engineering in Aerospace Engineering
Ryerson University, June 2004

A THESIS
PRESENTED TO RYERSON UNIVERSITY
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

IN THE PROGRAM OF

MECHANICAL EGINEERING

TORONTO, ONTARIO, CANADA, 2007

Mechatronic System Parameter Identification via Genetic Algorithms

Masters of Applied Science
In the Program of
Mechanical Engineering
2007

Mathew I. Adamson

School of Graduate Studies
Ryerson University

Abstract

This thesis develops a novel way to identify both the joint friction parameters and a built in torque sensor gain and offset. The identification method is based on a genetic algorithm (GA). A model based friction compensation method and a real coded GA are selected from a variety of methods available. A model of a single degree of freedom mechatronic joint with a link is presented. Numerical simulations are run to determine the optimum configuration of the GA with respect to the population size and maximum number of generations necessary to identify the parameters to within 5% of their actual value. The GA identification technique is then used on an experimental mechatronic joint with a harmonic drive and built-in torque sensor. The friction parameters as well as the sensor gain and offset are identified in the experimental system and the position tracking error is reduced. Based on the experimental results, the method is found to be an effective way of identifying system parameters in a mechatronic joint.

Acknowledgements

The author would like to offer his sincere thanks to his advisor, Dr. G. Liu who offered technical and moral support throughout the entire process. A debt of gratitude is also owed to Sajan Abdul whose expertise in controls helped guide the author through the development of the simulations and experiments. Thanks are also offered to the Modular and Reconfigurable Robot (MRR) group at Ryerson, which includes Daqing Wang and Richard Mohamed for helping to design and build the MRR module.

The experimental portion of this research would not have been possible without the financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC) and Engineering Services Inc. (ESI). The technical support staff from the Ryerson Aerospace Engineering department, Jerry Karpynczyk, Hamid Ghaemi and Primoz Creznik, provided technical assistance, as well as timely advice for manufacturing and fabrication of the prototype module.

The author is also grateful to Brian Leonard and Hillary Burridge who provided support and assistance to the author.

Contents

List of Figures

List of Tables

Nomenclature

Roman

b - Viscous friction

e - Position error

$\dot{e}$ - Velocity error

f_c - Coloumb friction coefficient

f_s - Static friction coefficient

f_v - Stribeck friction coefficient

g_{max} - Maximum number of generations

k - Adaptive control gain

q - Radial position

$\dot{q}$ - Rotational velocity

$\ddot{q}$ - Rotational acceleration

q_d - Desired position

$\dot{q}_d$ - Desired velocity

$\ddot{q}_d$ - Desired acceleration

r - Mixed tracking error

u - Random number between [0,1]

v - Velocity

y - Mutation range

z - Internal friction state

A - Trajectory amplitude

C_i - Children solutions

F - Friction

I - Inertia

P_i - Parent solutions

P_c - crossover probability

P_m - mutation probability

P_{min} and P_{max} - the minimum and maximum values in two parent solutions

V_{min} - lower bound for dynamic adaptation crossover

V_{max} - upper bound for dynamic adaptation crossover

Greek

α - BLX-α crossover coefficient

β - Sensor offset

β_q - Probability distribution

δ_s - Stribeck power coefficient

ε - Robust control constant

φ - Average bristle stiffness parameter

γ - Gear ratio

ρ - Parametric uncertainty bound

σ - Sensor gain

σ_0, σ_1 - LeGru friction parameters

τ - Torque

τ_c - Commanded torque

τ_j - Coupling torque

τ_s - Sensed torque.

Chapter 1

Introduction

1.1 Background

Mechatronic joints have applications that range from the Canadarm for use in space to hard drive arms that are used to read data from magnetic disks to Computer Numerical Control (CNC) machines that produce manufactured goods. Robotic joints can be utilized in many ways, which creates almost endless opportunities for improvements and research. In all robotic joints there is a common phenomenon – friction. Friction is caused when two surfaces come in contact with one another and microscopic asperities on the surfaces interlock. In order for the surfaces to move with respect to one another, some force must be exerted. Once the surfaces begin to move, the asperities constantly come in contact and break. The end result of this is twofold, there is a loss of kinetic energy which is dissipated as heat and the two surfaces wear down. The phenomenon can be very difficult to deal with, depending on the application, and there are numerous research papers devoted to dealing with this complex behaviour.

There are fundamental components required in order to control a joint. The first is a model of the joint. This often includes mathematical relationships between the actuation and the joint, as well as links that may be involved with the joint. In a system with multiple joints, it becomes complicated to model mathematically because the joints become coupled in their motion. The ability to solve a high degree of freedom system in real time is dependant on the available computing power and is limited. The second component is the use of sensors. Sensors that are typically used in robotics are a position encoder and an accelerometer which measure position and acceleration, respectively. While sensors provide information about the system, they have physical limitations. Common problems are caused by electromagnetic interference (EMI) which limits the sensitivity of the senor as well as sensor accuracy which is generally due to how precisely the sensor itself can be manufactured. Within the last two decades, torque sensors have been developed to measure the joint torque. Joint torque sensors have opened up new fields of

research. The torque sensor can be used to directly compensate for friction, but perhaps a more interesting application is to use it to measure the upstream link dynamics and avoid using complex dynamic equations.

The modular and reconfigurable robot (MRR) project under development at Ryerson University intends to utilize the joint torque sensor to further system controls research. By using the joint torque sensor to compensate coupling dynamics and friction, one of the purposes of the project is to show these techniques are viable and practical by doing a variety of experiments. Due to the particular equipment used to conduct this research, friction compensation is an integral part in this overall project. A major component of friction compensation is parameter identification. In addition to identifying friction parameters by using robust search techniques, other information about the system can be obtained, such as sensor parameters. This thesis focuses on identifying such parameters.

1.2 Thesis Objectives and Contributions

1.2.1 Problem Statement

To do precise motion control of a mechanical joint an appropriate control law must be designed. The dynamics of a joint where friction is present are non-linear. To achieve precise control, it is necessary to use a non-linear control law. Such control laws generally include a model of the system integrated into them. The system model relies on various parameters of the system. These parameters are not always directly measurable. The performance of the control law will rely on the accuracy of the parameter identification. Parameter identification can rely on a series of experiments that are a) difficult to accomplish in certain situations b) time consuming and therefore costly. The goal of this work is to develop a tool which is applicable to systems in order to identify system parameters. The parameters that are specifically identified in this thesis include friction coefficients as well as the joint torque sensor gain and offset.

1.2.2 Parameter Identification

The purpose of this thesis is to develop a parameter identification method by combining two different fields of research: a) friction modeling and compensation and, b) genetic algorithms (GAs), aiming to find a novel solution to the problem of friction compensation and joint control. Using information from an un-calibrated joint torque sensor, it is hypothesized that a GA will be able to identify the system parameters, including both the friction parameters and the joint torque

sensor parameters. GAs immediately come to mind when one considers noisy feedback signals, discontinuous friction functions and because of their robust nature, do not depend on a perfectly accurate model for the plant. In this way, it is possible to calibrate the joint torque sensor and the performance can be improved. As the temperature changes or the gears wear the GA will be available to update the system parameters accordingly over time. The end result is an effective and practical identification technique that will be available for the MRR as well as similarly designed joints.

1.2.3 Genetic Algorithms

As a field of research, GAs were introduced in the 1970s and have grown alongside the advancement of digital computers. GAs are global optimization techniques that are particularly well-suited for highly nonlinear functions. GAs can handle noisy, discontinuous functions because there is no requirement for a derivative in the fitness function. Moreover, GAs accumulate information about the system during the search process, which makes them more desirable than other random search algorithms.

A major contribution of this thesis is to develop a suitable fitness function that can be used to identify the parameters. The fitness function has been tested in both simulations and in experiments to evaluate its effectiveness.

1.3 Thesis Layout

This thesis is organized in the following fashion: Chapter 2 includes an overview of four different friction models. There are several friction compensation techniques that are discussed. Chapter 3 introduces GAs and the advantages of using GAs as well as discussions about the various kinds of GAs that are available to solve various problems in robotics and parameter estimation. Chapter 4 details the system model that is the main focus of the identification procedure. Once the system has been modelled, the identification technique is introduced. A fitness function is developed for this problem. Based on this fitness function, the procedures to identify friction and torque model parameters using GAs are formulated. Chapter 5 studies the effectiveness of the proposed procedure by conducting simulations of a single degree of freedom joint with a joint torque sensor. The simulations not only demonstrate that the feasibility of the GAs, but also provide a roadmap for the experiments. Chapter 6 takes the procedure to an experimental setup to show that it can be applied in a real mechatronic joint. Several

experiments are conducted to demonstrate the effectiveness of the proposed algorithm as well as investigating the limitations of the identification procedure. Finally, Chapter 7 details the conclusion, overall result of this research and possible future research in this topic.

Chapter 2

Friction Modeling and Compensation Techniques

2.1 Friction Compensation

Friction arises when two surfaces come in contact as a consequence of irregularities of the two surfaces. At a microscopic level, the asperities on each surface catch on one another must be broken in order for the surfaces to move relative to one another. As this process takes place, a thin layer of lubricant is accumulated and the two surfaces can move more readily [1]. The result of this contact is a resistance to any force exerted on either of the bodies in contact. The mechanics of the interaction depend on many things, including the presence of foreign lubricants, temperature and the materials. There are a variety of models which accurately capture this behaviour in many ways, from a simple Coulomb model to complex static and dynamic models.

2.1.1 Friction Models

2.1.1.1 Coulomb and Viscous Model

A simple Coulomb and viscous model can be expressed as:

$$F(\dot{q}) = \left[f_c \operatorname{sgn}(\dot{q}) + b\dot{q} \right] \tag{2-1}$$

where F is the friction force, $\dot{q}$ is the relative velocity of the contact surfaces, b is the viscous friction coefficient, and f_c is the Coulomb friction. The sign function is defined as:

$$\operatorname{sgn}(\dot{q}) = \begin{cases} 1 & for \quad \dot{q} > 0 \\ 0 & for \quad \dot{q} = 0 \\ -1 & for \quad \dot{q} < 0 \end{cases} \tag{2-2}$$

2.1.1.2 Static and Stribeck Model

The model in equation (2-1) does not accurately reflect what takes place at low speeds in real systems. It is known that when two objects are in contact, it takes an initial force to push

them apart. This force is often referred to as the breakaway force and the phenomenon is described as static friction or "stiction". What follows is a nonlinear region of motion between the break away force and the viscous friction. This region is referred to as the Stribeck region. The following model describes this behaviour:

$$F(\dot{q}) = \left[f_c + (f_s - f_c) e^{-\left|\frac{\dot{q}}{f_\tau}\right|^{\delta_s}} \right] \operatorname{sgn}(\dot{q}) + b\dot{q} \qquad (2\text{-}3)$$

where f_τ and f_s are the Stribeck coefficient and the static friction coefficient, respectively. The exponential term, δ_s, can be equal to 2, but depending on the material, not necessarily [1]. The curves in Figure 2-1 show the two friction models graphically. It can be seen that the static friction and Stribeck effect occur at low velocities, but at higher velocities the models exhibit the same behaviour.

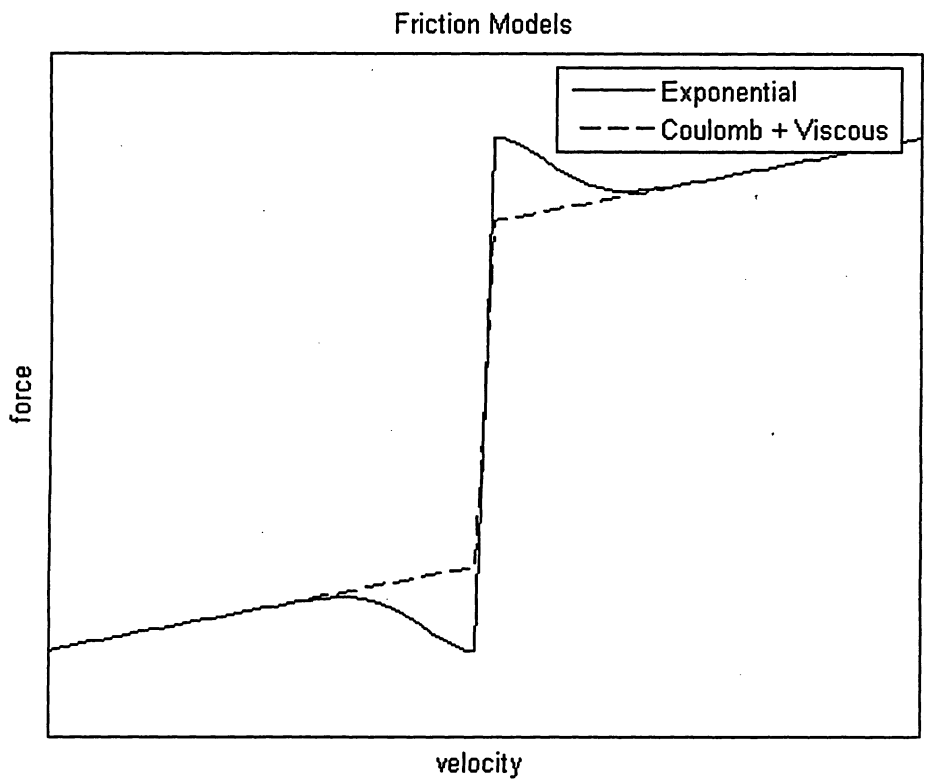


Figure 2-1: Friction models

2.1.1.3 Dahl Friction Model

Dahl [2] introduced the idea of analogizing friction with two layers of bristles rubbing against each other. If this is the case, the average bristle deflection at the contact points of the moving surfaces can be represented by introducing an internal state, z. The friction can then be defined as:

$$\dot{z} = v - \frac{|v|}{F_c}\varphi z \tag{2-4}$$

$$F = \varphi z \tag{2-5}$$

where φ is the average bristle stiffness parameter. This model does not include the static friction or the Stribeck effect.

2.1.1.4 LuGre Friction Model

The Dahl model does not account for a small amount of motion that occurs before the two contacting surfaces break away from one another. This motion is called "pre-sliding" and can be found in some applications where this very small deflection is important [3], [4]. The LuGre model [5] expands on the Dahl model by including the static friction and Stribeck friction. The two parameters, σ_0 and σ_1 are assumed to be constant. The function $g(v)$ and $f(v)$ model the Stribeck effect and the viscous friction, respectively.

$$\dot{z} = v - \frac{|v|}{g(v)}\sigma_0 z \tag{2-6}$$

$$F = \sigma_0 z + \sigma_1 \dot{z} + f(v) \tag{2-7}$$

This method is difficult to apply because the internal state, z, is not a measurable quantity. Moreover, observing this internal state requires high precision sensors in order to measure the "pre-sliding" phenomenon.

2.1.2 Friction Compensation Strategies

As friction in physical systems is a complex non-linear phenomenon, several different techniques have been developed to compensate for it. Depending on the application, the system may require different levels of precision. The following sections will highlight some advantages and disadvantages of the different compensation techniques.

2.1.2.1 Fixed Friction Compensation with a Known Model

In friction compensation with a known model a general control scheme is used with a fixed friction term. The friction model parameters are known or identified offline. This method can be effective if the parameters do not vary over time due to temperature or wear. Papadopoulos and Chasparis show in [6] that using a fixed friction compensation for the model in equation (2-1) cannot provide accurate tracking and positioning results compared to the friction model in equation (2-3).

2.1.2.2 Friction Compensation with a Partially Known Model

Friction compensation with a partially known model techniques rely on using a developed control algorithm, and then adding terms to compensate for the friction. For this technique, the exact friction model is not necessarily known, nor does the technique rely on the knowledge of knowing the model exactly.

In 1997, after introducing the LuGre model, Canudas de Witt and Lischinsky [7] implement the new model into a real system. They first describe how to determine the friction parameters of their model. This method is carried out offline. Once nominal friction values have been determined, the control system adapts to compensate for time dependant phenomenon such as temperature effects, and physical wear. The proposed method is compared to a proportional, derivative and integral (PID) controller without friction compensation and is found to be more effective. Another contribution of this paper is a method for determining the friction parameters via a series of experiments. The experiments were designed to isolate each parameter and identify them independently. For example, to identify viscous velocity, the joint is run at different constant speeds, and using the relationship between the applied torque and the velocity, a straight line can be found. The slope of this line determines the viscous damping coefficient.

Southward et al. [8] develop a proportional and derivative (PD) controller that includes an additional nonlinear torque term to compensate for "stick-slip" at low velocities. The "stick-slip" phenomenon occurs due to the static friction and has the effect of the surfaces "sticking" together with zero velocity. Once the command torque overcomes the static friction the surfaces "slip". The joint becomes unstuck and overshoots the desired trajectory because the static friction is higher than the Coulomb friction. The controller reverses the torque to compensate for the error. If not properly compensated, the two surfaces slow down and can become stuck together once again. The controller Southward et al. developed is asymptotically stable so long

as the static friction is bounded within a known region. The nonlinear compensation torque is only active in the "stick-slip" region and is essentially a PD controller elsewhere.

This technique has been upgraded by Kang [9] to include terms for hysterises. However, the added hysterises parameters must be carefully chosen to avoid any problems due to the time delay. Kang also found that a wide margin for the selection of such parameters will increase the error as well as increasing the stability margin.

2.1.2.3 Adaptive Control Schemes

It has been noted before that friction behaviour can change due to temperature, humidity or machine wear, so a natural solution to such a problem is an adaptive control scheme. However, using such a scheme presents two major problems a) some friction parameters are only modelled in a nonlinear way and b) there is a part of the system dynamics which relies on parameters that are not measurable. There is no global solution to estimate both the system parameters and an internal state. Depending on the application and the tools at hand there are several different ways to solve this problem. In one such example, Canudas de Witt and Lischinsky [7] considered a dynamic friction model in their compensation method. In order to compensate for changing temperature, two assumptions must be made. First, it is assumed that the temperature change affects the static region. Second, it is assumed that the dynamic region is reliant on a unchanging lubricant characteristics. The viscous coefficient can then be dealt with by a linear controller. Implementation of such a controller relies on a high precision resolver that is necessary for the observer to estimate the internal state, z. Using this compensation technique results in a trade off between the necessity of estimating the immeasurable part of the friction dynamics, and updating the friction parameters on-line. The nonlinear parameter related to the Stribeck effect is lost.

2.1.2.4 Adaptive and Robust Control

Song et al. [10] propose a combination of adaptive and robust controls. Their paper proposes an adaptive control method to estimate friction terms and a robust controller to compensate for the un-modeled friction terms. The authors argue that linearization of the model worked poorly because of inaccurate friction modeling. They also note that PD friction control does not have good steady state error performance while a PID controller produces a limit cycle instability. Their control algorithm eliminates both of these problems.

The adaptive controller consists of terms that are fed forward based on the known dynamic model of the system and the servo mechanism and part of a "stick-slip" friction model. This control method is able to compensate for the Coulomb friction and viscous friction forces based on the fact that they are linear or parametric linear. However, the exponential (Stribeck friction) and position-dependent forces which are nonlinear cannot be feed-forward cancelled.

In order to compensate for the nonlinear parameters and nonlinear behaviour, a robust compensator is introduced. The residue that is not cancelled is shown to be bounded. The adaptive compensator is embedded into the robust compensator to "learn" the proper upper bounding function.

The overall compensator is shown to by asymptotically globally stable via Lyapunov's theorem, which is used to proof stability in non-linear control theory. The control scheme is tested in numerical simulations, but not experimentally. In the simulations the stiction behaviour is only observed at zero-velocity crossings. It is concluded that the proposed scheme compensates for "stick-slip" friction.

2.1.2.5 Adaptive and Robust Control with Feedback Linearization

The control theory suggested by Liu [11] is one based on a decomposition control design. There is a separate compensator for the individual friction parameters that, when combined, provide an overall control scheme. The two control schemes are adaptive control and robust control. The adaptive control is used to compensate linear parametric friction for tracking control. The robust control is used to compensate for the un-modeled friction. In order for the robust controller to achieve a good tracking error, it would be necessary to use very high gain values which may not be physically available. Alone, the adaptive control by itself may become unstable due to an incompleteness of the friction model. Therefore, the overall determination is to integrate both techniques into a single control design.

Combining the robust and adaptive control is necessary to compensate for the parametric uncertainty of the friction. As the surface lubrication or surface temperature changes, the friction parameters are apt to change as well. The adaptive compensator is designed with the assumption that the friction parameters are constant but not exactly known. The robust controller is designed to account for the parametric uncertainty and un-modeled phenomenon. The theory is shown to be Lyapunov stable.

Liu et al. [12] demonstrated the proposed control scheme experimentally in 2004. Using a direct drive, one degree of freedom robot, they were able to show that the tracking error is reduced and the controller performs robustly, even in the presence of changing friction values. The experiments indicated that the friction parameters do not need to be known exactly; however, the closer the estimated values are to the real values, the better the performance of the controller will be.

2.2 Compensation and Control Using Torque Sensors

One method of measuring joint torque is to attach the sensor to the gear head and thus any upstream torque generated will be able to be measured. The configuration of this method is shown in Figure 2-2.

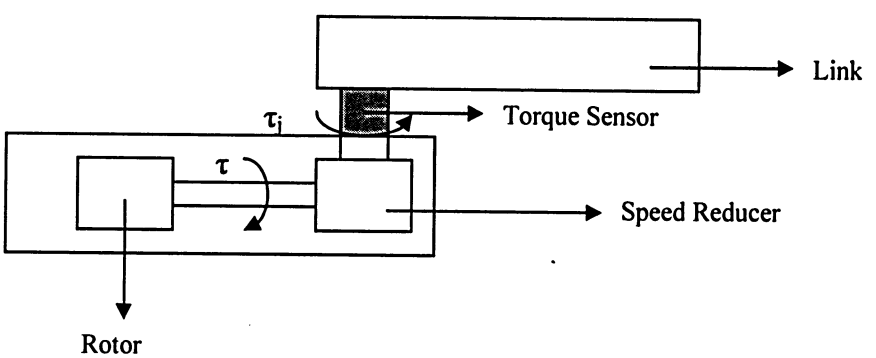


Figure 2-2: Joint diagram

One way to implement this sensor is to attach it to a harmonic drive, also referred to as an asymmetric drive. The drive consists of three main components. There is a wave generator which is elliptical shaped and fits onto the motor shaft. The second component is the flexspline which is a flexible piece of material that fits over the wave generator. Finally, there is a circular spline which fits over the flexspline and attaches to the link. The assembly can be seen in Figure 2-3. The principle behind the drive is that the teeth on the flexpline have a slightly different pitch and have two fewer teeth than the circular spline. When the elliptical wave generator rotates, the flexpline will deform and the teeth of the flexpline will engage those of the circular spline along the major axis of the wave generator. In this way, the speed between the wave generator and the circular spline can be reduced by a factor of up to 320:1, depending on the number of teeth in the gear.

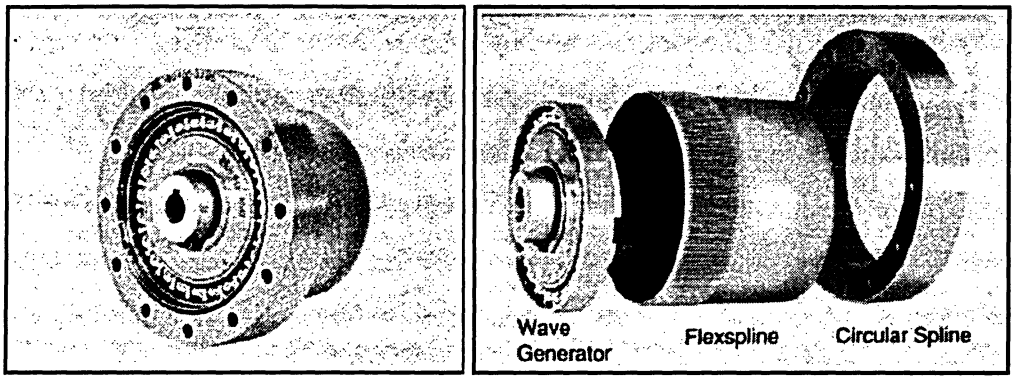


Figure 2-3: Harmonic drive [15]

Because of the flexibility of the flexpline, it is possible to attach a series of strain gauges which form a Wheatstone bridge. The bridge produces a voltage when there is an applied torque on the flexpline. The strain gauges are glued onto the bottom of the flexpline in the configuration seen in Figure 2-4.

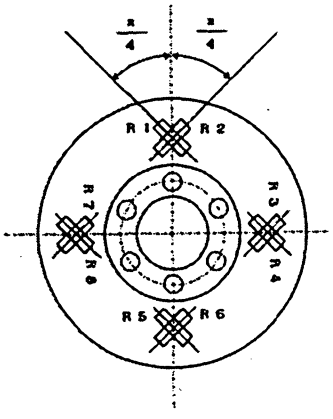


Figure 2-4: Strain gauge configuration on a harmonic drive flexpsline [16].

Hashimoto [14] first suggested using a joint torque sensor for position control in 1987. Prior to this time, the dynamics of the system were calculated using computed torque. This was done by using the inverse dynamics of the system to compute how much torque was necessary to move to the desired position (or velocity etc.). Clearly, for a multi-link robotic arm, this would be computationally expensive and becomes impossible to do in real time if there is a sufficient number of links. Hashimoto replaced this method by using the joint torque sensor. The sensor is used to measure the inertia and the gravitational effects of the upstream links. By feeding back

the torque sensor information, it is possible to use a PD controller to achieve satisfactory position tracking.

Many of the teeth are always in contact making a large amount of friction in this type of gear head. While Hashimoto implemented his control law successfully, it did not specifically address the presence of this increased friction. Moreover, a PD controller has poor steady state tracking error performance, as discussed in [1]. However, this procedure is very useful in compensating the nonlinear dynamics of multi-link systems and can be expanded on in future work.

Using the joint torque sensor to compensate directly for friction has been explored in the literature. Pfeffer et al. [17] used the joint torque sensor in a PUMA robot in 1989 to compensate for the friction effects. The effective friction was reduced by 97%. In 1998, Hashimoto and Kiyosawa [18] used the joint torque sensor built into a harmonic drive as described above to compensate for friction. By using joint torque feedback Morel et al. [19] achieved high precision control of a robotic manipulator without using a friction model. These three separate methods demonstrate the effectiveness of using the joint torque sensor to directly compensate for joint friction.

In 1990, Imura et al. [20] proposed a control method which uses joint torque sensor placed at each joint in the multi-link system. First a dynamic model based on inverse dynamics was developed. The dynamics are highly nonlinear and require a lot of computing power to solve as well as high sampling rates. To overcome this problem the dynamic equations are solved using a recursive Newton-Euler method using the information from the joint torque sensors that are place at each joint. This method does not require the information of the mass of the links, only the rotors, which are known. Thus, the robustness of the control law is improved against parameter variations caused by the loads attached to the end of the arm.

In 1991, a different control method using joint sensors was proposed by Kosuge et al. [21]. The authors suggest that a robust control system based on inverse dynamics is not convenient for design. The coupling terms of the links and modeling error of the actuator are not negligible. This is due to the fact that the dynamic expression does not have an explicit expression. The control scheme is considered robust because it is able to deal with the uncertainty of the actuators. The proposed method needs less knowledge about uncertainties of the actuator system

because of the joint torque sensors. This method is also able to reduce the computational load and reduce the feedback gain to achieve the tracking precision.

The wave generator in the harmonic drive causes an issue that is problematic with built-in joint torque sensors. The rotation of the wave generator causes strain in the flexspline, which is picked up by the strain gauges that are placed there to measure the torque. The strain gauges are placed in such geometry that this ripple effect is supposed to be cancelled out, but in practice this is impossible to do exactly [21], [22]. In 1998, Taghirad and Belanger [23] suggested using a Kalman filter to filter the noise from the signal. In 1999, Godler et al. [22] suggested that by adjusting the amplified gains on each of the strain gauges, the ripple can be cancelled out. Figure 2-5 shows how Godler et al. proposed to change the geometry of the strain gauges in order to help cancel out the ripples in the torque signal. The Kalman filter is difficult to implement in practice and requires an acceleration signal which is usually found by differentiating the velocity and is not accurate. Moreover, the calculations required to implement the Kalman filter need to be carried out in real time and add a computational burden to the controller. The Golder et al. proposed method is only useful if implemented before the sensor is constructed. If the sensor is already built, it cannot be modified in this way.

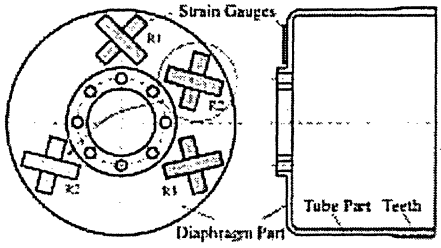


Figure 2-5: Modified geometry of strain gauges on flexspline [22]

There have been attempts by Tuttle and Seering [24] as well as Kennedy and Desai [25] to model harmonic drives. Tuttle and Seering found that, due to the amount of analysis required for the experimental data, it was unlikely that any comparably sufficient representation could not be made from parameters available in catalogues or simple experimental observations. Kennedy and Desai developed an experimental algorithm to determine the harmonic drive parameters for a Mitsubishi PA-10 robot kit that is commercially available. While the modeling and identification of harmonic drive parameters are beyond the scope of this work, it may be useful in future work or when analysing the experimental results.

Chapter 3

Genetic Algorithms

This chapter introduces the concepts of genetic algorithms. Following the introduction, discussion into techniques that are relevant to this work will be discussed. Also, applications of GA pertaining to robotic joints and controls will be discussed to show how they have been employed in this field of research.

3.1 Introduction into Genetic Algorithms

Throughout this work references will be made to a number of characteristics of genetic algorithms. Genetic algorithms are global optimization techniques that are particularly well suited for highly nonlinear functions. GAs are based on the Darwin's theory of natural selection. As new solutions are found, they are evaluated in accordance with their fitness. The fit solutions are crossed with one another to produce new generations. As the number of generations increase, the overall fitness of the population will also increase. An added advantage of occasional mutation helps to prevent the solution to converge to a local optimum [13]. The three main operators are selection, cross-over and mutation. The general process is shown in Figure 3-1. First, an initial population is randomly selected. Once the population is created it is necessary to evaluate the fitness of each individual. The individuals are chosen to mate based on their fitness. This process is called selection. After the mating has occurred, a random number of children are selected to be mutated. The mutation adds a random amount of new genetic material into the population. Finally, the objective function is re-evaluated and the process is repeated over a set number of generations.

This chapter will include a brief discussion of the two major types of GA coding. In addition, some examples of how GAs have been successfully applied in robotics and system identification will be discussed.

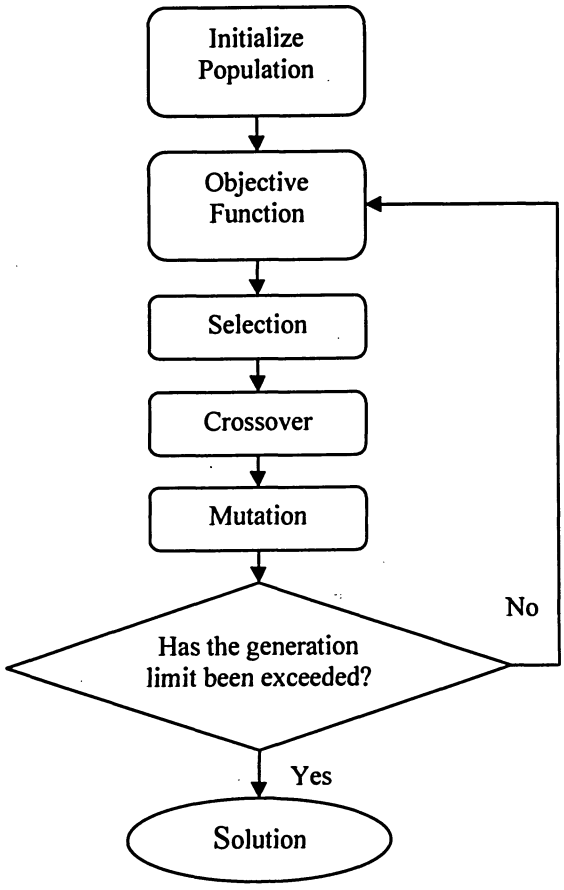


Figure 3-1: Genetic algorithm

3.2 GA Coding

There are two distinct types of GA coding methods, binary coded and real coded. While both coding techniques mechanically operate in the same fashion, i.e., they evolve through generations seeking global solutions, the ways in which they go about it are distinctly different. The two techniques will be discussed in the following sections to point out some of the key differences, but for a comprehensive overview, the reader is referred to [13], [26], [27], and [28].

3.2.1 Binary Coded GA

Binary coded GAs work on the idea that the information in the code is stored in a binary string, i.e., a string of 1's and 0's that represent integer numbers. The integer numbers represent solutions to the problem at hand. In order to use integers to model in real number problems, it is necessary to map the integers into a continuous domain to evaluate the fitness. This procedure

requires several arithmetic operations, which alone are not significant in computational time, but when a population of strings with multiple parameters over hundreds of generations is considered, it can become time consuming.

Another phenomenon that arises when using binary coding is called the Hamming cliff. The Hamming cliff effect occurs when a binary number is converted to a decimal number. For instance, consider two binary strings:

$$[0\ \ 1\ \ 1\ \ 1\ \ 1]=15$$
$$[1\ \ 0\ \ 0\ \ 0\ \ 0]=16$$

While the two strings have five different digits when represented by binary, there is only a difference of one in the real number domain. Herrera et al. [28] note that this phenomenon can be problematic when searching a continuous search space.

3.2.2 Real Coded GA

A real coded GA is used because it allows for the search of large domains without sacrificing precision as would be necessary in a binary coded algorithm [28]. Furthermore, real coded GA are not susceptible to the Hamming cliff effect [27]. Whereas the binary coded GA relies on a crossover technique that switches bits between two parent strings, the real coded GA generates random numbers around the parents to generate the children.

Vasconcelos et al. [27] discuss a technique called dynamic adaptation crossover and mutation. This technique varies the probability of crossover and mutation while the GA is executing. Generally, crossover is desirable when the solutions are spread throughout the search space and mutation is desirable when the solutions begin to converge to a single point which could be a local solution. In order to determine if the solution is converging on a single point, the concept of genetic diversity (*gdm*) is introduced. This term is the ratio between maximum value of the fitness function and the average value. Figure 3-2 shows how the probability of crossover (P_c) and mutation (P_m) vary as the *gdm* is increased. V_{min} and V_{max} are user defined variables.

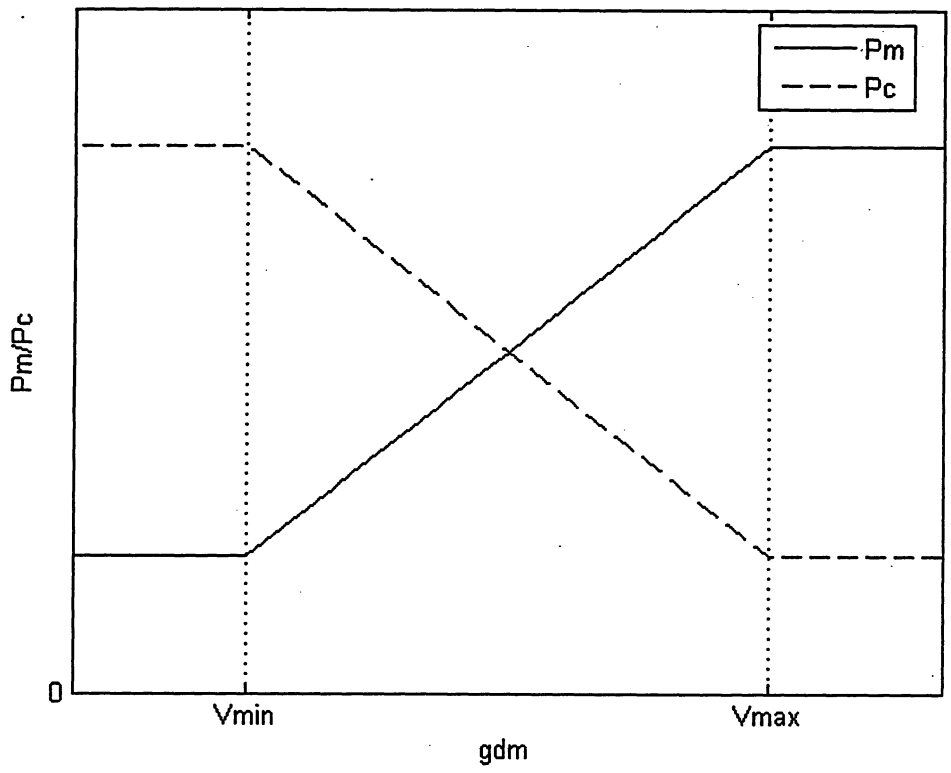


Figure 3-2: Dynamic crossover and mutation

3.2.2.1 Real Coded Crossover

Several crossover techniques are presented in the literature, [28, 29, 30 31]. Simulated binary crossover (SBX) is a crossover technique developed by Deb and Beyer [26]. This technique was developed from the blend crossover (BLX-α) which can be expressed as:

$$[P_{\min} - \alpha \cdot I, P_{\max} + \alpha \cdot I] \tag{3-1}$$

where P_{min} and P_{max} are the minimum and maximum values in two parent solutions, α is an adjusting parameter for the range, I is the difference between the two parent solutions. The best known value for α is 0.5 [28]. The SBX improves on this technique by making it more likely that the children will be selected close to the parents as opposed to far away. However, there is still a small probability that they will be selected far away, which increases the randomness in the search algorithm to avoid a local maximum. The SBX crossover can be expressed as:

$$C_1 = 0.5[(1+\beta_q)P_1 + (1-\beta_q)P_2]$$
$$C_2 = 0.5[(1-\beta_q)P_1 + (1+\beta_q)P_2] \tag{3-2}$$

where C_1, C_2, P_1, and P_2 are the children and parent solutions, βq is obtained from a probability distribution function with a random number u between 0 and 1, and βq can be expressed as:

$$\beta_q = \begin{cases} (2u)^{1/(\eta+1)} & if\ u \le 0.5 \\ \left(\frac{1}{2(1-u)}\right)^{1/(\eta+1)} & if\ u > 0.5 \end{cases} \tag{3-3}$$

where η is the distribution index and a nonnegative real number. A large value of η gives a higher probability for creating children solutions near parent solutions.

3.2.2.2 Real Coded Mutation

Another consideration in a real coded GA is the mutation. In a binary coded GA the mutation is done by simply selecting a random bit and switching it from 1 to 0 or vice versa. This is a totally random function and could turn 14 into 15 or 256 into 128. In real coded GA it is possible to better control the mutation. Michalewicz and Schlierkamp [32] proposed a non-uniform mutation rate in 1992. A uniform mutation rate can be expressed as the random child selected, C_i' is any uniform number in the domain of the parameter$[a_i, b_i]$. The proposed non-uniform mutation (NUM) takes the form of:

$$C_i' = \begin{cases} C_i + \Delta(t, b_i - C_i)\ if\ \varsigma = 0 \\ C_i - \Delta(t, C_i - a_i)\ if\ \varsigma = 1 \end{cases} \tag{3-4}$$

$$\Delta(t, y) = y\left(1 - u^{\left(1-\frac{\psi}{g_{max}}\right)^{\omega}}\right) \tag{3-5}$$

where g_{max} is the maximum number of generations and ω is a number selected by the user to determine the dependency on the number of iterations. The term ς is randomly chosen as 1 or 0, and ψ is the current generation number. The function gives a value in the range of [0,y] such that the probability of returning a number close to zero increases as the algorithm advances. The size of the gene generation interval shall be lower with the passing of generations. This property causes this operator to make a uniform search in the initial space when t is small, and very locally at a later stage, favouring local tuning.

3.3 GA Applications in Robotics

Genetic algorithms are a useful tool in many research fields including robotics. There are a variety of practical applications including control, parameter identification or trajectory tracking. The following subsections will illustrate some of the interesting ways that GA have been applied in the field of robotics.

3.3.1 Genetic Algorithms and Fuzzy Logic

In 1998, Lih-Chang and Ywh-Jeng [33] suggested using genetic algorithms combined with fuzzy logic and adaptive control to compensate for friction. The authors use the LeGru friction model. In this application there is significant frictional memory and "pre-sliding", which is the justification for the author to use this particular friction model. "Pre-sliding" refers to the amount of torque that is applied before the surfaces break away.

The control scheme combines adaptive compensation with fuzzy control. The fuzzy logic is combined with genetic algorithms in such a way that the binary string from the genetic algorithm yields the membership function for the fuzzy logic output set. In this way, the GA is used to learn the fuzzy subsets. The use of fuzzy controllers is beyond the scope of this work, but the combination of various artificial intelligent methods may be considered for future work. A complete stability theory was not introduced and the authors suggest it as future work. Also, no experimental work was done to justify the simulation.

3.3.2 GAs for Trajectory Planning

GAs are used in robot trajectory planning. The process of the path generation can be broken into two parts. In the first part, the dynamic properties of the robot are not taken into consideration; this is due to the fact that the dynamic behaviour of the arm is complex and is being influenced by factors such as gravity, joint elasticity, friction or reaction forces due to adjacent links. The programmer must specify by a series of points to be reached by the end-effector. In the second portion of the path planning, the inverse kinematics are calculated to created a smooth path for each link through the desired points. The number of paths that are possible is very large and the number of points is not specified.

A GA approach to this problem was suggested by Davidor [34]. The robot considered by Davidor consisted of three links restricted to the vertical plane with all dynamic effects considered. The inputs are the arm configuration vectors which define the end-effector's

positions. The output is the actual position of the robot via the position sensors. Each arm-configuration vector can be considered as a chromosomic structure and an n-step trajectory can be represented by a string. Because there is no set number of steps for each trajectory, new GA operators must be introduced which can vary the length of the strings. This technique is compared to a hill-climbing algorithm and the GA is found to be more consistent.

3.3.3 Parameter Identification

In 1998, a genetic algorithm is proposed by Liaw and Huang [35] to determine friction coefficients for a robot with contact friction. They use a control method with feedback linearization to do position control. The friction is modeled with only Coulomb and viscous friction, equation (2-1). The control scheme considers Stribeck friction as a disturbance. If the parameters for Coulomb and viscous friction are known exactly, the tracking error will be driven to zero (assuming position and velocity are exactly measured as well).

The overall control scheme is shown in Figure 3-3. It can be seen that the error in the position and velocity are used directly in the fitness function. This implies that the GA is run online.

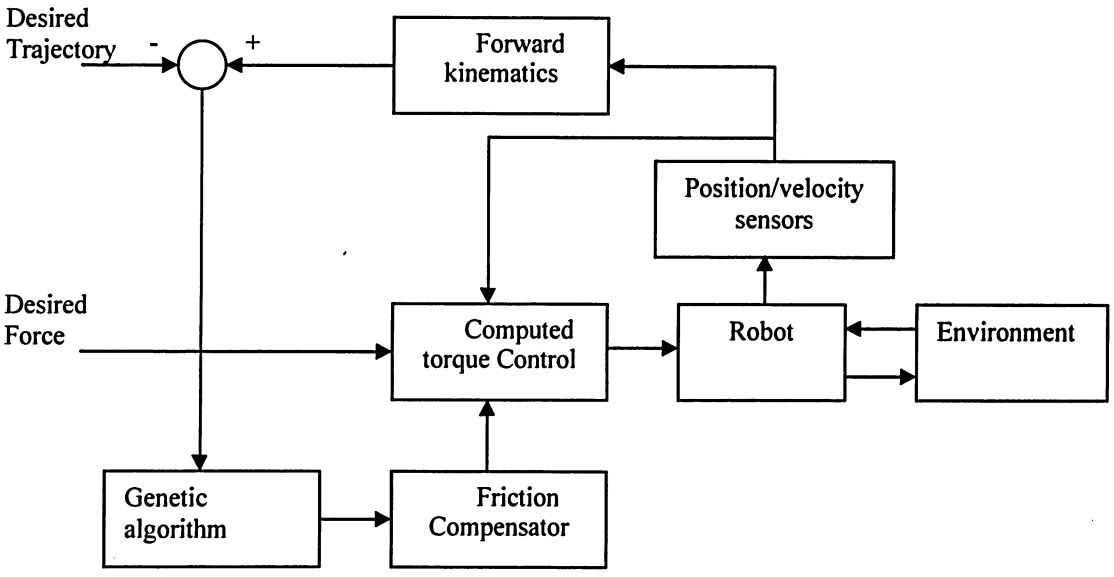


Figure 3-3: Closed loop control diagram [35]

Liaw and Huang chose to use an initial population of 100, a crossover probability of 0.98, and a mutation probability of 0.2% over 50 generations. After running simulations, it is

concluded that the friction parameters can be found accurately within a few seconds. Simulations with artificial measurement noise are also attempted successfully. The authors even try to simulate a three parameter friction model with the Stribeck friction involved and find that it is successful. However, they note that if a large deviation from the friction model occurs in a real system, the tracking errors will be very large and the system may even become unstable. There is no experimental result to demonstrate whether or not the simulations are accurate.

Chapter 4

System Modeling and Parameter Identification Method

In order to effectively implement the GA identification technique, it is important to understand the system that is being investigated. The key components of the model are the system equations, the friction phenomenon and the joint torque sensor. Once the system is modelled, it will be possible to proceed to the identification method.

4.1 Selecting a Friction Model

Chapter 2 outlined many different friction models and friction compensation techniques. It is necessary to decide which model best suits the current application. Two friction models, the Coulomb + viscous model and the Dahl model do not include terms for static and Stribeck friction. It is noted that in low speed trajectory tracking, compensating for these phenomenon is necessary to achieve small errors. For the MRR project, it is desirable to have high precision tracking at low speeds so both the Dahl model and the Coulomb + viscous models are rejected. The two remaining models both provide the characteristics desirable, but from a controls point of view, one is more desirable than the others. As noted in Section 2.1, the LuGre model relies on internal state estimation and requires a high accuracy velocity sensor. Such a sensor is not available for the MRR project; therefore, the four parameter static + Stribeck model from equation (2-3) is selected.

4.2 Modeling the Plant

Consider a dynamic model for a joint with a torque sensor and a speed reducer between the rotor and the link with the following assumptions:

1) The rotor is symmetric with respect to the axis of rotation.
2) The joint flexibility is negligible.

3) The torque transmission does not fail at the speed reducer, and the inertia between the torque sensor and the speed reducer is negligible.

The following notations will be used;

I_m: the moment of inertia of the rotor about the axis of rotation;

I_l: the moment of inertia for the link about the axis of rotation;

γ: the reduction ratio of the speed reducer ($\gamma \geq 1$);

q: the joint angle;

$f(q,\dot{q})$: the joint friction, which is assumed to be a function of the joint position and velocity;

τ_s: the coupling torque at the torque sensor location; and

τ: the output torque of the rotor;

G(q): the forces due to gravity.

The joint friction is modeled as equation (2-3) with the addition of a term that accounts for unmodelled position dependent friction,

$$f(q,\dot{q}) = \left(f_c + f_s \exp\left(-f_\tau \dot{q}^2\right)\right)\mathrm{sgn}(\dot{q}) + b\dot{q} + f_q(q,\dot{q}) \qquad (4\text{-}1)$$

The dynamic equation for joint with the link can be written as:

$$\tau = G(q) + (I_m\gamma + I_l)\ddot{q} + f(q,\dot{q}) \qquad (4\text{-}2)$$

If the joint torque sensor is positioned as shown in Figure 2-2, then it can be assumed that the sensor signal will be able to replace the terms for the gravitational effects of the link, as well as the link inertia, so long as the friction of the joint is compensated. So it is possible to replace those two terms with a coupled joint torque at that point.

$$\tau_j = I_l\ddot{q} + G(q) \qquad (4\text{-}3)$$

An un-calibrated joint sensor is known to have two critical parameters that must be identified [36], the sensor offset and the sensor gain. The sensor offset is caused by a change in temperature of the joint. The Wheatstone bridges that make up the sensor should be symmetric, but it is not possible for the sensors to have exactly the same characteristics, so a voltage bias appears. This voltage bias acts as a steady offset for the sensor [37]. The voltage created in the Wheatstone bridges is generally around 0-20 mV, while analog to digital converters use signals that require 0-5 V, so it is necessary to amplify the signal. It can be assumed that this gain value

is not precisely known and hence the sensor is un-calibrated. The torque sensor is modeled as $\tau_j = \beta\tau_s + \sigma$, where β is the sensor gain and σ is the sensor offset.

The model can now be written as:

$$(I_m\gamma)\ddot{q} + \left(f_c + f_s\exp\left(-f_\tau\dot{q}^2\right)\right)\mathrm{sgn}(\dot{q}) + b\dot{q} + f_q(q,\dot{q}) + \frac{(\beta\tau_s + \sigma)}{\gamma} = \tau \qquad (4\text{-}4)$$

4.3 Control Scheme

The overall control scheme is shown in Figure 4-1.

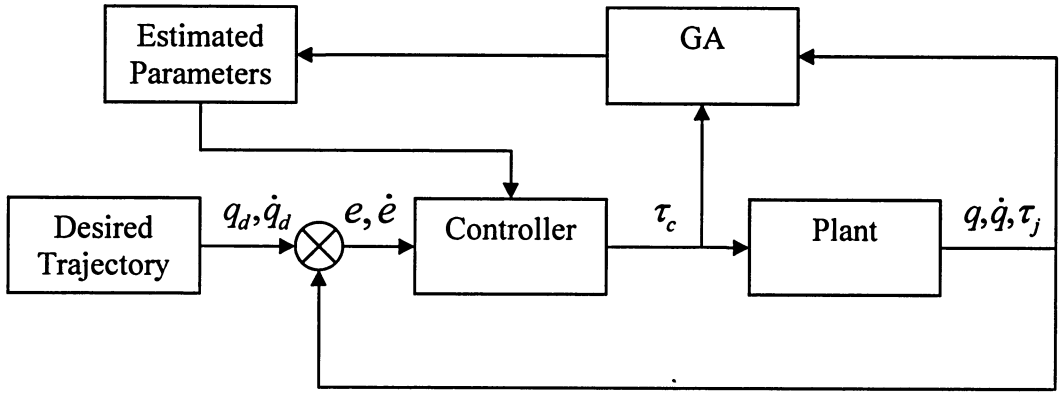


Figure 4-1: Overall control scheme

In order to understand what exactly has to be identified, it is useful to state the control scheme that will be used. The control scheme implemented here is a modified version of the linearized decomposition based control by Liu [11]. In that scenario, a link was not considered. In the following treatment, the effect of the link will be compensated by feeding back the torque sensor signal. For completeness, the control scheme will be presented.

In order to implement the decomposition based control scheme, it is first necessary to linearize the nonlinear model:

$$I_m\gamma\ddot{q} + \hat{b}\dot{q} + \left(\hat{f}_c + \hat{f}_s\exp\left(-\hat{f}_\tau\dot{q}^2\right)\right)\mathrm{sgn}(\dot{q}) - Y(\dot{q})\tilde{F} + f_q(q.\dot{q}) + \tau_j = \tau \qquad (4\text{-}5)$$

where $\hat{b}, \hat{f}_c, \hat{f}_s, \hat{f}_\tau$ denote the nominal values of their respective parameters. Also:

$$Y(q) = \left[\dot{q} \quad \mathrm{sgn}(\dot{q}) \quad \exp\left(-\hat{f}_\tau\dot{q}^2\right)\mathrm{sgn}(\dot{q}) \quad -\hat{f}_s\dot{q}^2\exp\left(-\hat{f}_\tau\dot{q}^2\right)\mathrm{sgn}(\dot{q})\right] \qquad (4\text{-}6)$$

$$\tilde{F} = \begin{bmatrix} \hat{b}-b & \hat{f}_c - f_c & \hat{f}_s - f_s & \hat{f}_\tau - f_\tau \end{bmatrix}^T \tag{4-7}$$

There are two different control techniques integrated: an adaptive control law to compensate for the constant uncertainty of the friction parameters, and a robust control compensator to deal with the unmodelled friction and the variable friction parameters.

The objective of the control scheme is to provide precise tracking. The position and velocity tracking errors are defined by:

$$e = q - q_d, \; \dot{e} = \dot{q} - \dot{q}_d \tag{4-8}$$

The mixed tracking error is defined by:

$$r = \dot{e} - \lambda e \tag{4-9}$$

where λ is a positive constant.

For tracking control, the nominal torque value is defined as:

$$\tau_0 = I_m \gamma a + \hat{b}\dot{q} + \left(\hat{f}_c + \hat{f}_s \exp\left(-\hat{f}_\tau \dot{q}^2\right)\right)\mathrm{sgn}\left(\dot{q}\right) + \tau_s \tag{4-10}$$

where a is defined by:

$$a = \ddot{q}_d - 2\lambda\dot{e} - \lambda^2 e \tag{4-11}$$

Here, the nominal torque term has been modified to include the corrected torque sensor signal in order to compensate for the link dynamics. Therefore, the overall control law is given by:

$$\tau = \tau_0 + Y\left(\dot{q}\right)u_p + u_u \tag{4-12}$$

The adaptive control technique is employed to deal with the unknown but constant parametric uncertainty:

$$u_p = -k\int_0^t Y\left(\dot{q}\right)^T d\tau \tag{4-13}$$

where k is a tunable control gain.

The non-parametric uncertainty, $f_q(q,\dot{q})$, is bounded by:

$$\left|f_q\left(q,\dot{q}\right)\right| < \rho \tag{4-14}$$

where ρ is a known constant bound for any position q and velocity $\dot{q}$. The robust control used to compensate for the non–parametric uncertainty is defined by:

$$u_u = \begin{cases} -\rho \dfrac{r}{|r|}, & if \quad |r| \geq \varepsilon \\ -\rho \dfrac{r}{\varepsilon}, & if \quad |r| < \varepsilon \end{cases} \tag{4-15}$$

where ε is a positive control constant that can be tuned for achieving better tracking results [11].

4.4 Identification Technique

Before proceeding into this task in depth, it is important to understand how the parameters affect the motion of the joint. This task will be accomplished by conducting a sensitivity study on the model to test the effect of each parameter. It is critical to understand how the variance of the parameters affects the model in order to choose a suitable excitation for the joint.

4.5 Sensitivity Study on Model Parameters

To demonstrate where each parameter affects the function, it is necessary to plot the function with a small change in each parameter. Figure 4-2, Figure 4-3, Figure 4-4, and Figure 4-5 show a variance of each friction parameter within 10% of its original value. The sensor parameter sensitivity is shown in Figure 4-6 and Figure 4-7 where the parameters are also varied within 10% of its original value.

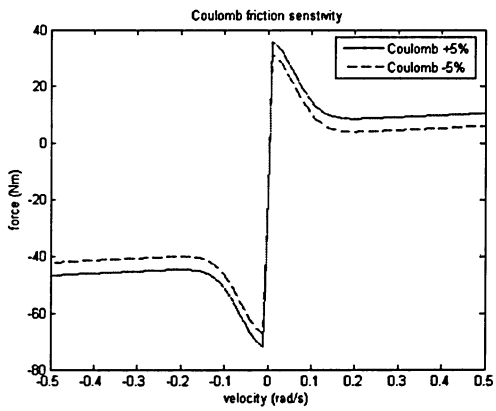


Figure 4-2: Coulomb friction sensitivity

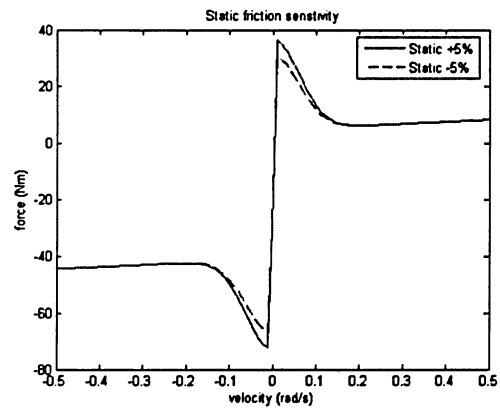


Figure 4-3: Static friction sensitivity

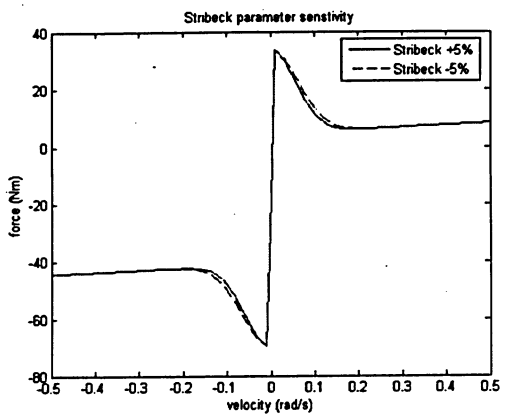


Figure 4-4: Stribeck friction coefficient sensitivity

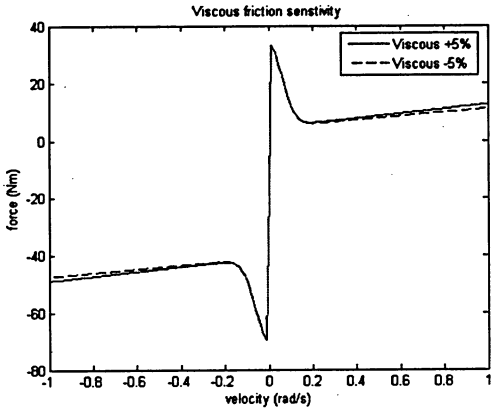


Figure 4-5: Viscous friction sensitivity

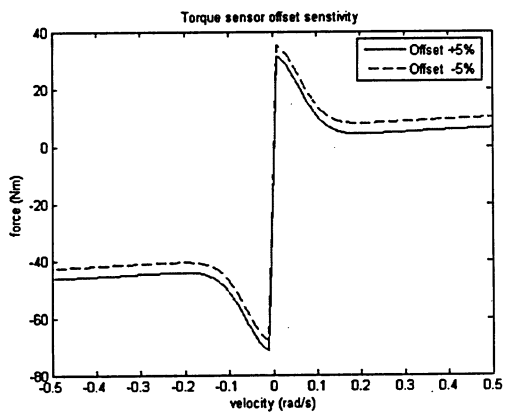


Figure 4-6: Sensor offset sensitivity

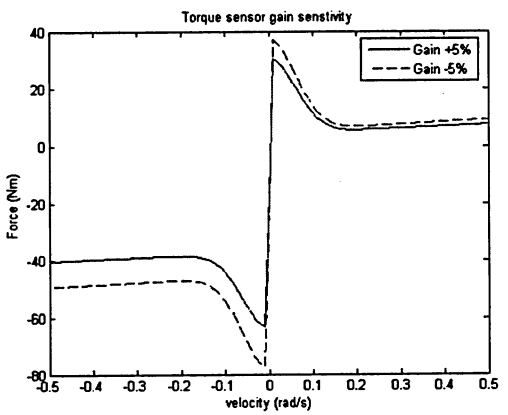


Figure 4-7: Sensor gain sensitivity

Several observations can be made from the sensitivity study. First, several of the system parameters are coupled within different regions of the velocity space. It can be seen that at all speeds the Coulomb friction, sensor gain and sensor offset values affect the curve. The Stribeck and static effect can only be seen at very low velocities whereas the viscous effect only appears at higher velocities. This implies that in order to capture the behaviour of all of the parameters, it is necessary to get sensor readings from all of the different velocity values. Moreover, it is important when considering the excitation of the system that a certain region is not emphasized more than another region, or it might dominate the result.

4.6 GA Fitness Function

The fitness function of the GA must be constructed such that it effectively identifies the variables. The sensitivity study showed where each parameter affects the motion of the joint.

Based on this knowledge a fitness function is constructed. The identification technique relies on feedback from the joint. The feedback signals are from both the encoder, which measures the position, and from joint torque sensor, which measures the torque of the joint at the sensor location. By differentiating the position signal, it is possible to obtain the velocity, but it is important to keep in mind that this term will be noisy. Because of the noise in the velocity signal, it is not practical to differentiate the signal again. The information would not reflect the actual state of the robot.

In order to understand the derivation of the fitness function, a simple curve will be used as an example. This will illustrate the procedure and thought process used to obtain the fitness function. Consider the two simple sine curves as shown in Figure 4-8.

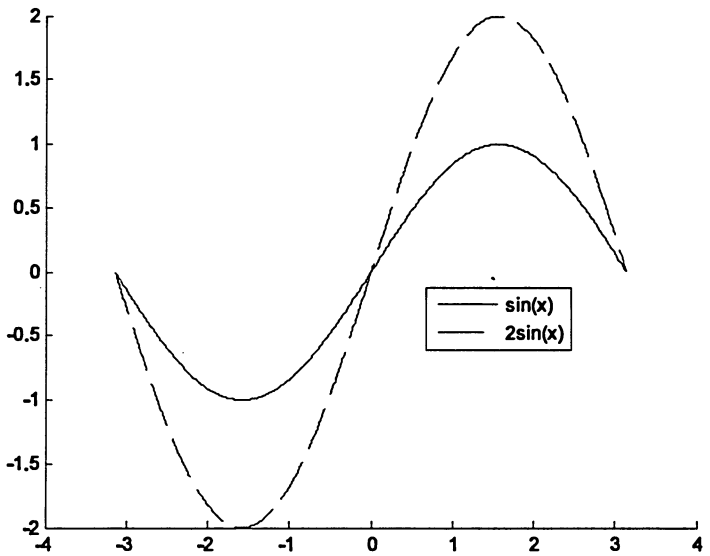


Figure 4-8: Sine curves

The sine curve over this domain is an odd function. This means that when integrated, the negative area exactly cancels the positive area and the overall area is zero.

$$\int_{-\pi}^{\pi} \sin(x)dx = 0, \int_{-\pi}^{\pi} 2\sin(x)dx = 0 \qquad (4\text{-}16)$$

Clearly, the two curves are very different, but the integral function does not highlight the differences in the curve shape. It only highlights the total area that each contains. Consider different definite integral limits, such as [-π, 1]. It can be seen in Figure 4-9 that the area under each curve over this domain is not the same in this case.

$$\int_{-\pi}^{1} \sin(x)dx = -0.7738 \;,\; \int_{-\pi}^{1} 2\sin(x)dx = -0.1548 \qquad (4\text{-}17)$$

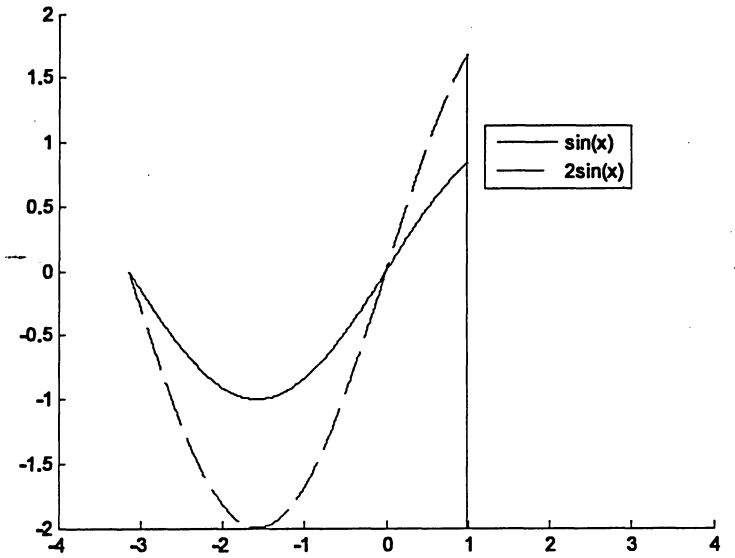


Figure 4-9: Sine curve evaluated at -π and 1

The value of the integral clearly changes as the limits of the integral change. A graph of the integral shows where there is the greatest difference between the two curves. Figure 4-10 shows the results of the definite integral along x.

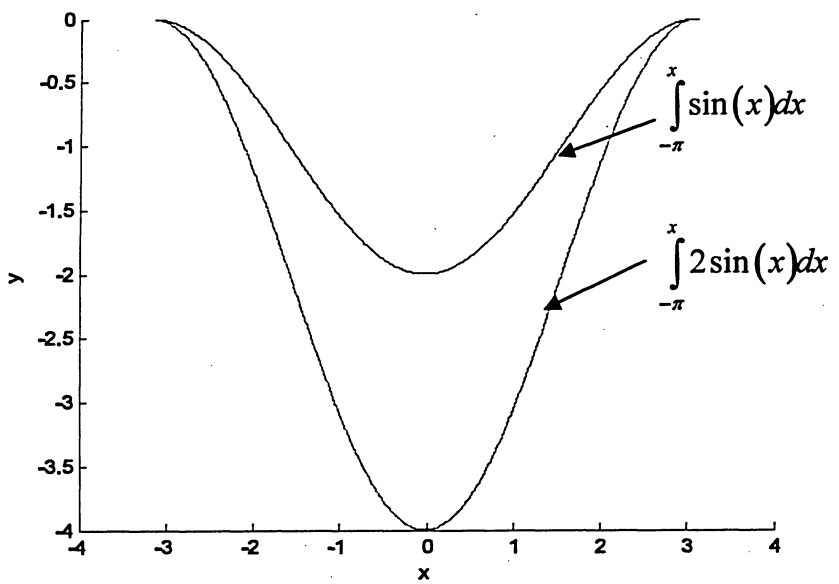


Figure 4-10: Definite integration of two sine curves

By comparing the two curves at various points, it is possible to get a quantitative value for how different they are. For example, consider the difference between the two curves at 10 places

along the integral curves. Let the function, P, be a penalty assigned to quantify the difference between the two curves. Figure 4-11 shows the different values of x where the definite integral is chosen.

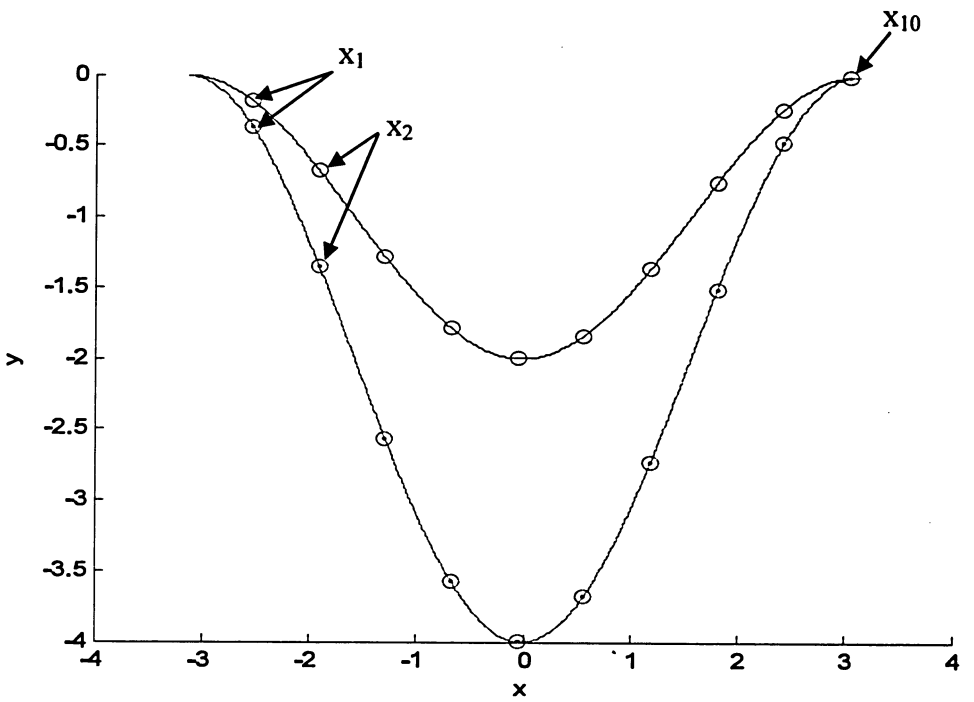


Figure 4-11: Definite integration of sine curves at various points

This is can be expressed as:

$$P=\left(\int_{-\pi}^{-\pi+\left(\frac{2\pi}{10}\right)}\sin(x)dx-\int_{-\pi}^{-\pi+\left(\frac{2\pi}{10}\right)}2\sin(x)dx\right)+\left(\int_{-\pi+\frac{\pi}{10}}^{-\pi+\left(\frac{4\pi}{10}\right)}\sin(x)dx-\int_{-\pi+\frac{\pi}{10}}^{-\pi+\left(\frac{4\pi}{10}\right)}2\sin(x)dx\right)+\ldots$$
$$+\left(\int_{-\pi+\frac{9\pi}{10}}^{\pi}\sin(x)dx-\int_{-\pi+\frac{9\pi}{10}}^{\pi}2\sin(x)dx\right)\approx 0.1909+0.6909+\ldots+0\approx 10.1 \qquad (4\text{-}18)$$

In general this expression can be expressed for any number, n, of divisions.

$$P=\sum_{i=0}^{n-1}\left[\int_{x_1+i\frac{x_2}{n}}^{x_1+(i+1)\frac{(x_2-x_1)}{n}}\sin(x)dx-\int_{x_1+i\frac{x_2}{n}}^{x_1+(i+1)\frac{(x_2-x_1)}{n}}2\sin(x)dx\right] \qquad (4\text{-}19)$$

For this expression it should be noted that as $n \to \infty$ then the penalty, $P \to \infty$ as well.

This procedure quantifies the different shapes of the original functions by comparing how the areas are different along the path. Consider the function that is used to describe the joint, neglecting the unmodeled friction, $f_q(q,\dot{q})$.

$$I_m\gamma\ddot{q} + \left(f_c + f_s\exp\left(-f_\tau\dot{q}^2\right)\right)\mathrm{sgn}\left(\dot{q}\right) + b\dot{q} + \frac{\left(\beta\tau_s + \sigma\right)}{\gamma} = \tau \qquad (4\text{-}20)$$

As stated previously, the acceleration term is something that cannot be obtained through the available sensors. While the velocity is obtained by differentiating the position signal, differentiating this signal again to obtain acceleration is not practical. The resulting signal is too noisy to be useful. To alleviate this problem, the entire function can be written as an integral function:

$$\int_{t_1}^{t_2}\left(I_m\gamma\ddot{q}\right)dt + \int_{t_1}^{t_2}\left(f_c + f_s\exp\left(-f_\tau\dot{q}^2\right)\right)dt + \int_{t_1}^{t_2}\left(b\dot{q}\right)dt + \int_{t_1}^{t_2}\left(\frac{\left(\beta\tau_s + \sigma\right)}{\gamma}\right)dt = \int_{t_1}^{t_2}\tau dt \qquad (4\text{-}21)$$

This entire equation can be rearranged to explicitly state the sensed torque value:

$$\int_{t_1}^{t_2}\tau_s dt = \beta^{-1}\left[\gamma\left\{\int_{t_1}^{t_2}\tau dt - I_m\gamma\left[\dot{q}\right]_{t_1}^{t_2} - f_c\left[t\right]_{t_1}^{t_2} - f_s\int_{t_1}^{t_2}\exp\left(-f_\tau\dot{q}^2\right)dt - b\left[q\right]_{t_1}^{t_2}\right\} - \sigma\left[t\right]_{t_1}^{t_2}\right] \qquad (4\text{-}22)$$

The equation defines a curve that can be obtained through the torque sensor. What is desirable is to estimate this curve by estimating all the parameters which are unknown.

The unknown parameters can be estimated and the resulting estimated torque can be expressed as:

$$\int_{t_1}^{t_2}\tau_e dt = \tilde{\beta}^{-1}\left[\gamma\left\{\int_{t_1}^{t_2}\tau dt - I_m\gamma\left[\dot{q}\right]_{t_1}^{t_2} - \tilde{f}_c\left[t\right]_{t_1}^{t_2} - \tilde{f}_s\int_{t_1}^{t_2}\exp\left(-\tilde{f}_\tau\dot{q}^2\right)dt - \tilde{b}\left[q\right]_{t_1}^{t_2}\right\} - \tilde{\sigma}\left[t\right]_{t_1}^{t_2}\right] \qquad (4\text{-}23)$$

If equation (4-23) is a cyclical function, then the torque signal can become an odd function. As described above, this can be detrimental when trying to compare the two curves, the sensed torque and the estimated torque. While the integral of the two curves may have the same result, the shapes may be very different. One may suggest that, in order to avoid this, do not use a cyclical velocity trajectory to obtain the sensor information for the identification process. However, it is the zero crossing data which is most critical to identify the aforementioned parameters. Therefore, the example that was illustrated for the simple sine curves will be used

for the more complex estimated and sensed torque curves. The expression can be simplified by assuming t_1 is equal to 0. The penalty can be defined as:

$$P = \sum_{i=0}^{n-1} \left[\int_{t_1+i\frac{t_2}{n}}^{(i+1)\frac{t_2}{n}} \tau_s dt - \int_{t_1+i\frac{t_2}{n}}^{(i+1)\frac{t_2}{n}} \tau_e dt \right] \tag{4-24}$$

The next task is to select a suitable value of n for the penalty. The torque sensor will have some kind of sampling time associated with it. It is not truly a continuous function, but a series of discrete values from the sensor. So if the sampling time is 1 ms then n would be a maximum of the total time divided by the sampling times of the torque sensor; for instance for 6 seconds of excitation, there would be 6000 pieces of information available. In the lower range, it is important to keep in mind that a larger value of n gives more information about the shape of the curve. The best way to determine a suitable value is to conduct some simulations and find what value of n provides enough information to obtain good results for the solution.

4.7 Implementation of GA for Parameter Identification

In Chapter 3, the necessary components of a GA are the fitness function, the crossover method and the mutation method. For this identification method, the crossover and mutation method used are from Equation (3-2) and Equation (3-4), respectively. The fitness function is inversely proportional to the square of the penalty function defined in Equation (4-24), i.e., the lower the penalty, the higher the fitness.

To implement the GA, a set of data must be collected which includes information of the joint position, joint velocity and the joint torque sensor. In the simulation, this will be accomplished by solving the necessary differential equations. In the experiment, this information will come directly from the sensors. Once the data is collected, the GA can be run and the optimal solution is found. Having found the set of parameters which best suit the system model, the simulation or experiment is run again to see whether or not the parameters estimated by the GA reduce the tracking error.

Chapter 5

Simulations

In order to evaluate the effectiveness of the proposed method, a series of simulations have been devised. The simulations serve several necessary functions; first the GA requires extensive tuning to find a good number of generations, population, crossover method or mutation rate that provides an accurate solution to the problem. A simulation is better suited to this task than an experiment because it is less time consuming, less expensive, and easier to control various parameters that may interfere with the method, such as sensor accuracy. Another benefit of the simulation is that the parameters that the GA is identifying can be set by the user and are thus exactly known. Once the method has been optimized, it can then be performed on a real system, which is the ultimate goal.

5.1 Simulation Layout

The simulation is written in the scripting language of MATLAB and uses Simulink. There is an initialization program in MATLAB that initializes the variables used in the simulation. Parameters such as the rotor inertia, the link size and mass are defined. Once the initial parameters are defined, Simulink is used to simulate the plant dynamics from equation (4-2). The values for the position and velocity of the joint are obtained through the integration of the plant equation. The simulated sensor information is written to a file while the simulation is running. Upon the completion of the simulation, the GA is run, which is also programmed as a MATLAB script file.

5.1.1.1 Simulation Trajectory

The following trajectories shown in Figure 5-1 and Figure 5-2 were chosen for two reasons. First, the position and velocity both begin at zero, so there is no jump, but rather a smooth transition from the initial position. Secondly, the velocity trajectory contains many zero crossing points. These crossing points are important because this is where the friction discontinuity

occurs and, thus, is the most difficult to compensate. The largest tracking error due to the friction in both position and velocity should occur in this region. It is desirable to have zero velocity crossings from both the positive and negative directions. However, it is also important not to collect too much data from a single region, whether it be high speed or low speed, because this will give too much weight in the fitness function to that one parameter which is dominant in that region. The position and velocity trajectories are shown in Figure 5-1 and Figure 5-2 respectively and are given by the following:

$$q_d = \sin(t) - 0.5\sin(2t),\ \dot{q}_d = \cos(t) - \cos(2t) \tag{5-1}$$

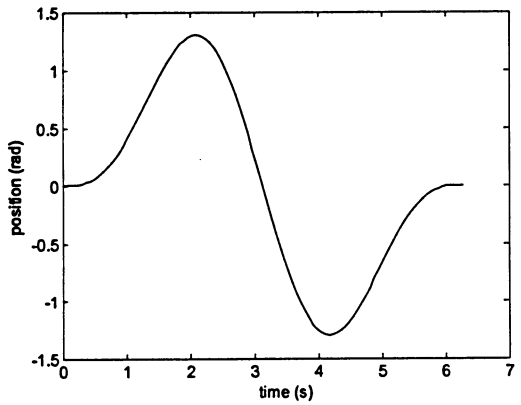


Figure 5-1: Position trajectory

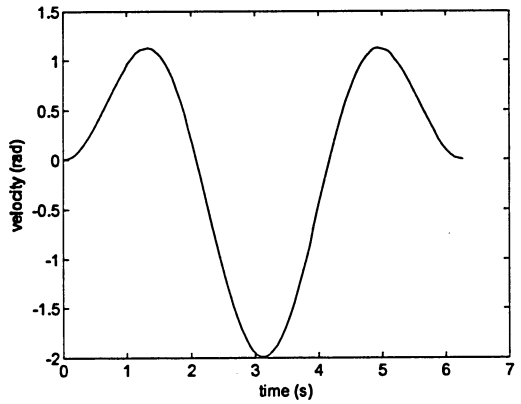


Figure 5-2: Velocity trajectory

5.2 Simulink Program

The simulations were carried out in Simulink. The Simulink blocks are shown in Figure 5-3. The Simulink program first reads the estimated parameters from an ASCII text file. These estimated parameters are used in the control law. The control law produces the required torque in order to achieve the desired trajectory. The control law has been introduced in Section 4.3. Equation (4-4) is integrated in order to solve for the velocity and position of the joint. A fixed step numerical solver is used to solve the plant equation with a time step of 1 ms. Once the dynamics have been calculated, a term for the torque sensor must be generated. In the physical system the torque signal would come directly from a sensor, but in this case it is necessary to estimate it numerically as seen in equation (4-3). In order to make the simulation more realistic, some sensor noise is added to the signal. The sensor noise is generated based on the harmonic drive dynamics derived by Taghirad and Belanger [23]. The motor velocity

determines the frequency of the oscillation and the sensor characteristics given by the manufacturer of a torque sensor indicate the magnitude of the ripple is 0.4% of the full scale value of the rated torque for the harmonic drive.

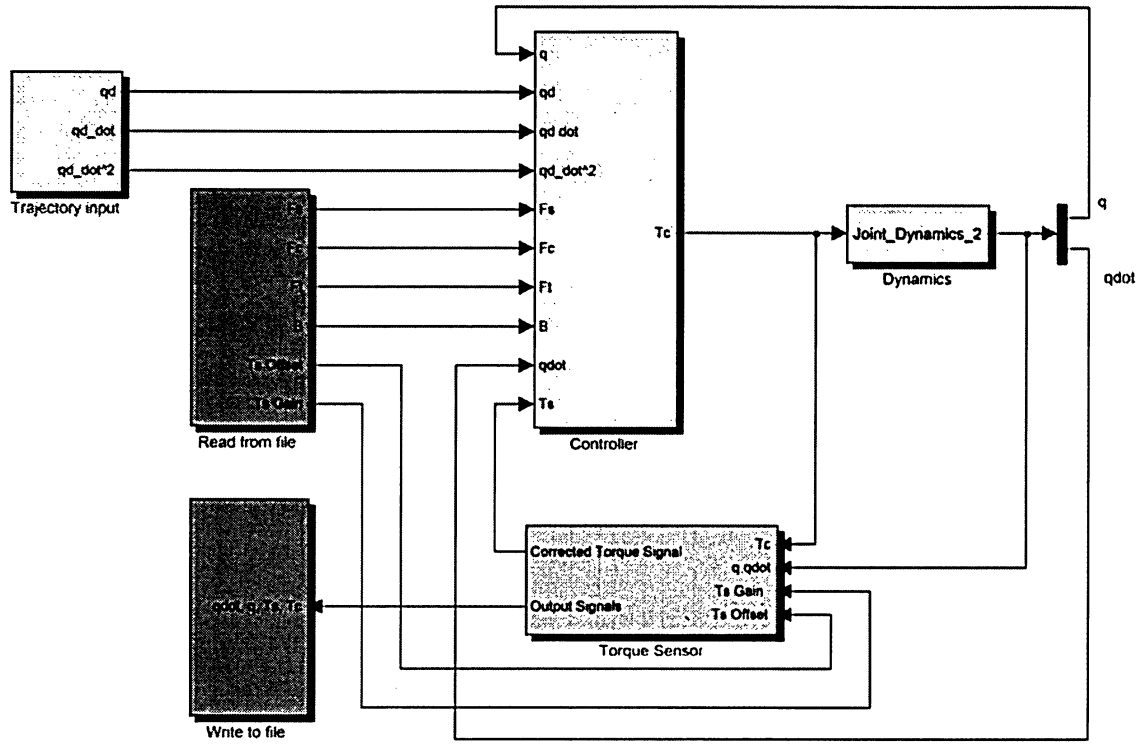


Figure 5-3: Simulink program

5.3 GA Characteristics Selection

Selecting the GA characteristics, i.e., population size, crossover rate, mutation rate, maximum generation etc. is generally one of trial and error. While the literature can give some ideas, generally these parameters are very problem specific. In order to evaluate whether or not the selection of such parameters is suitable or not, there have to be some design constraints introduced. The overall goal is to reduce the tracking error. The control scheme takes care of the stability but requires parameter estimates to perform optimally. In order to test what is the best combination of population size and how many generations to search through, several test cases were considered.

5.3.1 Test Case 1:

The first test case considers a single trial of each population size and generation size. The resultant solution is plotted against a random estimate of the parameters. This may be considered to be the first time the joint is activated and there is no knowledge of any parameter. The GA is then executed and the updated parameter can be expected to enhance the tracking. Figure 5-4 and Figure 5-5 show the results of the GA estimation vs. a random estimation. When looking at the tracking error, two factors should be considered a) the mean distance from zero and b) the peak value of the error.

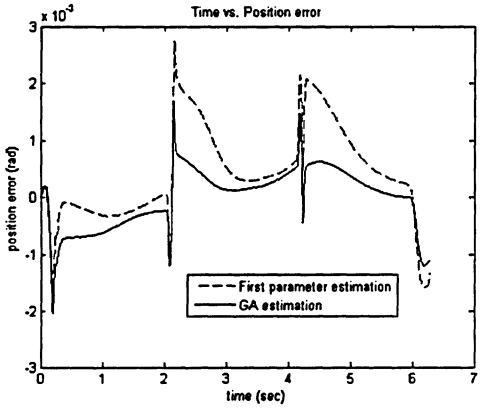


Figure 5-4: Response for 20 population over 100 generations

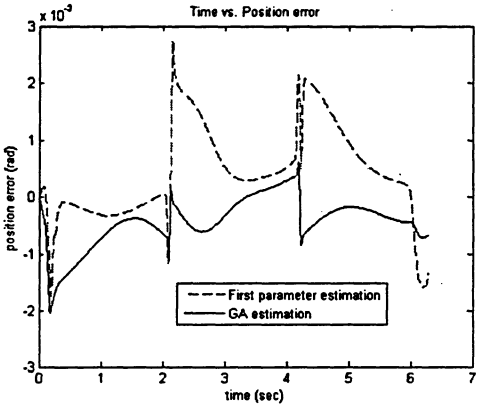


Figure 5-5: Response for 200 population over 100 generations

Two observations can be made from these results. Firstly, the GA does better than a random guess. Even with a small number of solutions explored i.e., a population of 20 over 100 generations or about 2000 solutions, the tracking response is much better. Another expected result is that by exploring the search space with more population over the same number of generations, the result obtained is improved. The next task is to establish an optimal number of population and generations to search to find an adequate estimation of the parameters.

5.3.2 Test Case 2:

For this test case the following situation is considered; the parameters have been identified after an initial run of the experiment. Now it is desirable to improve on the performance which could degrade due to temperature effects or physical wear. In order to understand how the parameters are affected by error, an example will be given as to the effect of parameter estimation, as shown in Figure 5-6 to Figure 5-11.

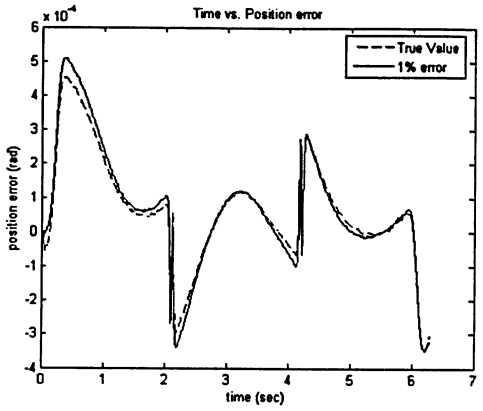


Figure 5-6: Response for 1% error in parameter estimation

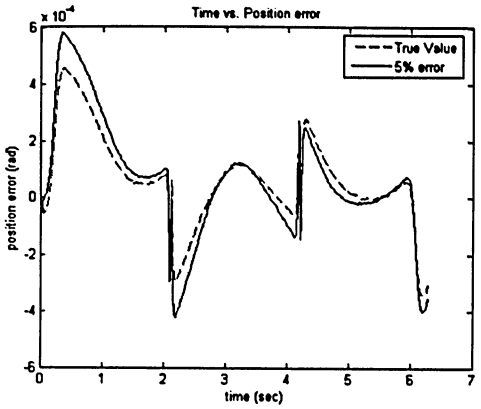


Figure 5-7: Response for 5% error in parameter estimation

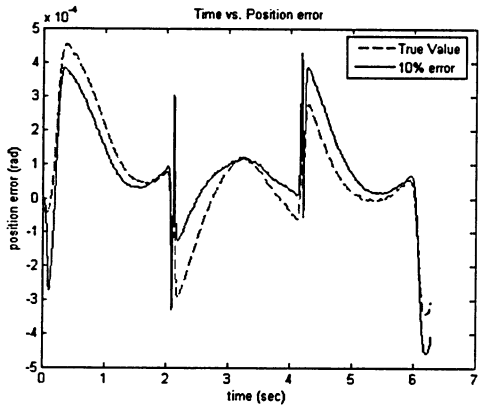


Figure 5-8: Response for 10% error in parameter estimation

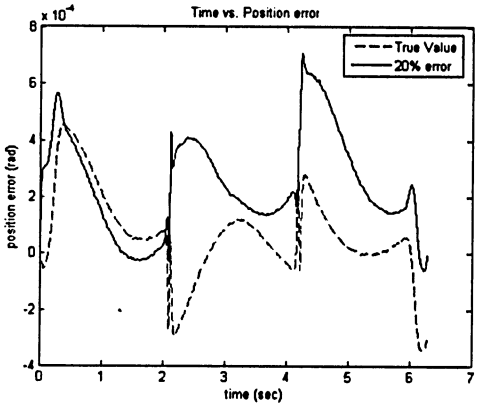


Figure 5-9: Response for 20% error in parameter estimation

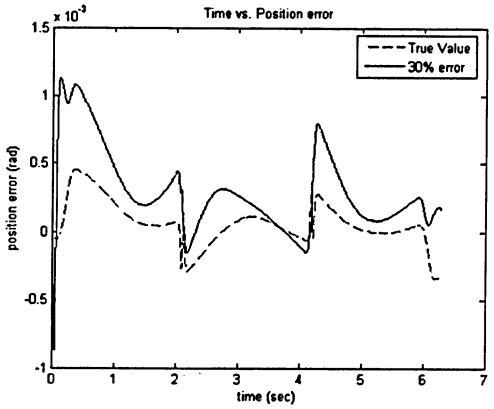


Figure 5-10: Response for 30% error in parameter estimation

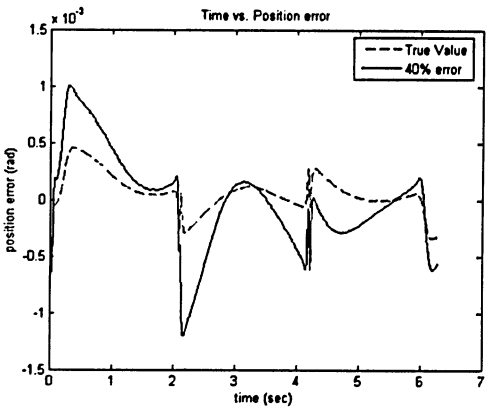


Figure 5-11: Response for 40% error in parameter estimation

A good estimate of the parameters, i.e., parameters randomly selected to within ± 5% of their actual value is compared with the results of 10 trails of the GA results averaged out. The purpose of this test case is to see if the GA is able to achieve better tracking than the randomly selected variables within the ± 5% range. If the GA is able to achieve such results in the simulation, then it is a promising candidate for the experimental test. Rather than comparing the value of each parameter directly, it is better to view the performance. If a single parameter is poorly estimated, it could only affect a small region of the tracking error.

5.4 Simulation Results

Figure 5-12 to Figure 5-17 show the results of a population of 20 over various numbers of generations. In the case of the lower generations, it is clear that the GA does not find an adequate solution. The solutions improve as the number of generations increase, but between 1000 and 1500 generations, the solution does is no longer improving, this suggests that the GA needs to begin the search with more initial solutions.

Table 5-1: GA properties for test case 2 - 1

Population size	Maximum generations	$P_{c,max}$, $P_{c,min}$	$P_{m,max}$, $P_{m,min}$	Trials
20	20-1500	0.95, 0.80	0.08, 0.03	10

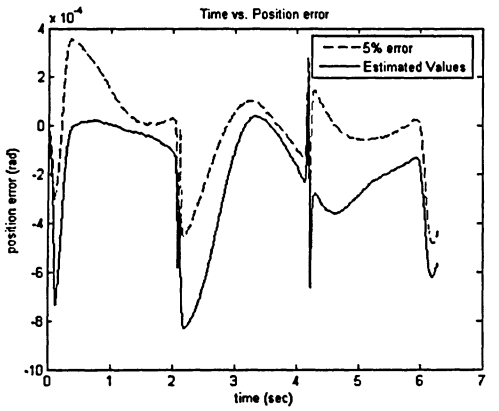


Figure 5-12: Response for 20 population over 20 generations

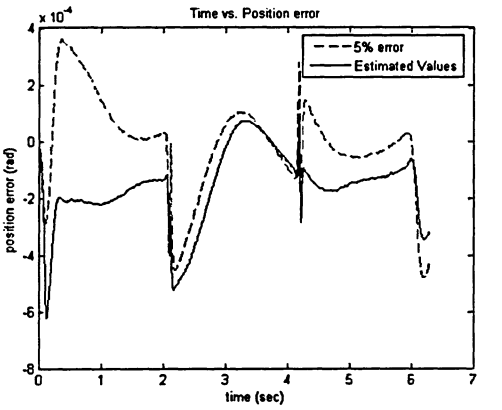


Figure 5-13: Response for 20 population over 50 generations

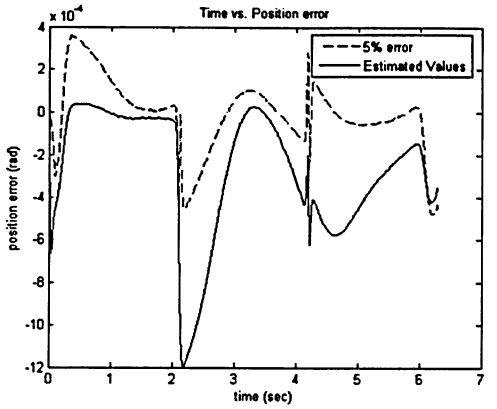


Figure 5-14: Response for 20 population over 100 generations

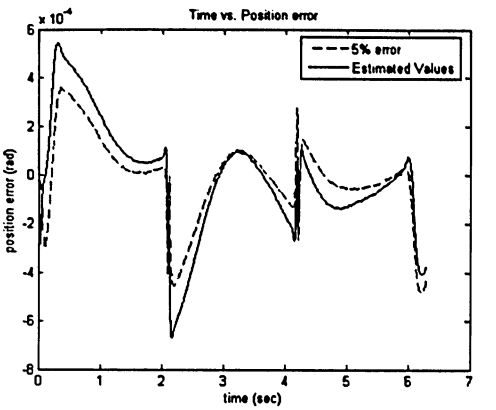


Figure 5-15: Response for 20 population over 500 generations

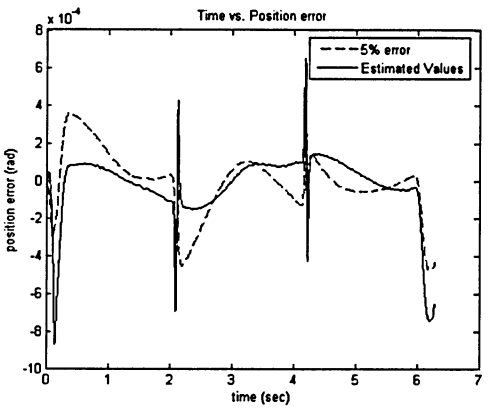


Figure 5-16: Response for 20 population over 1000 generations

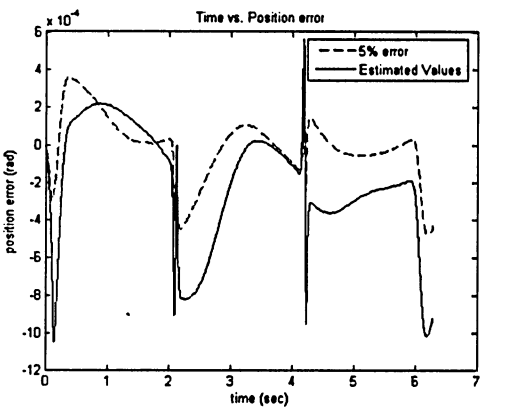


Figure 5-17: Response for 20 population over 1500 generations

Figure 5-18 to Figure 5-23 show the results for a population of 50 over a number of different generations.

Table 5-2: GA properties for test case 2 – 2

Population size	Maximum generations	$P_{c,max}$, $P_{c,min}$	$P_{m,max}$, $P_{m,min}$	Trials
50	20-1500	0.95, 0.80	0.08, 0.03	10

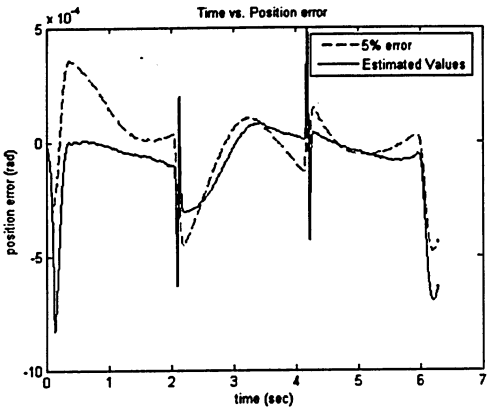


Figure 5-18: Response for 50 population over 20 generations

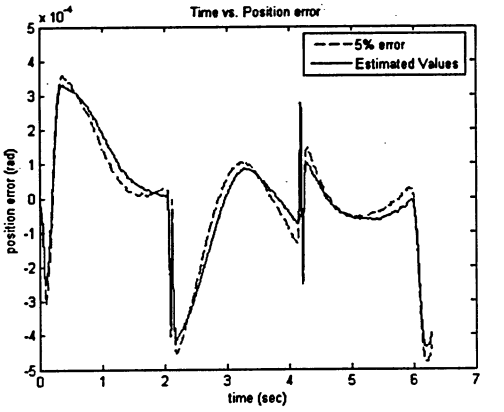


Figure 5-19: Response for 50 population over 50 generations

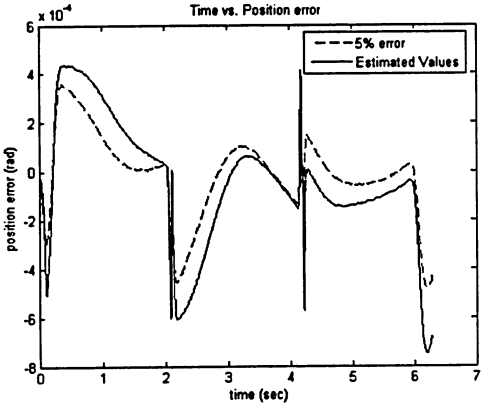


Figure 5-20: Response for 50 population over 100 generations

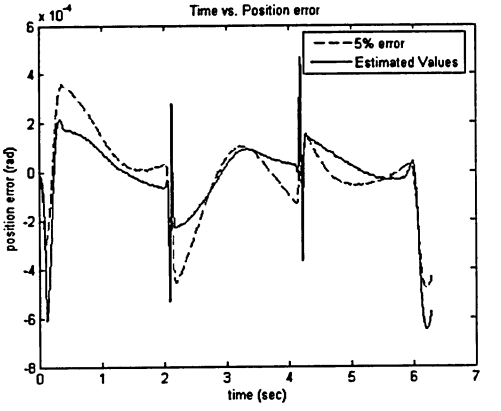


Figure 5-21: Response for 50 population over 500 generations

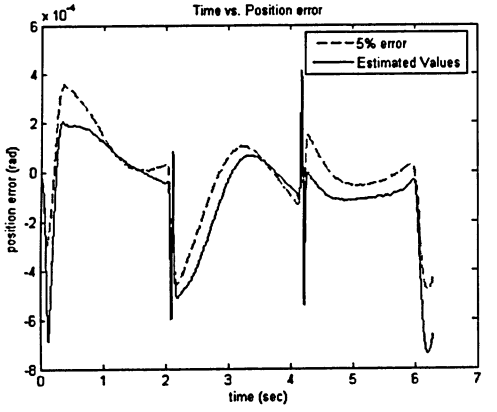


Figure 5-22: Response for 50 population over 1000 generations

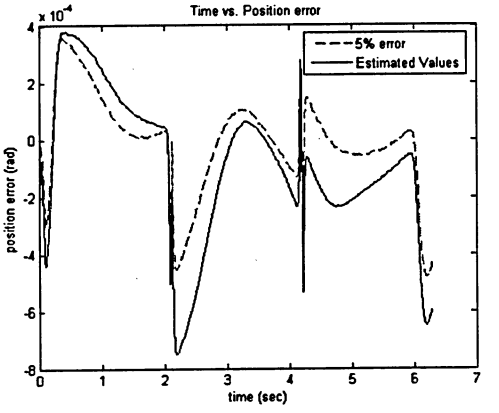


Figure 5-23: Response for 50 population over 1500 generations

For this scenario, the GA once again shows improvement as more generations are used to explore the search space. The solutions still do not fall within the 5% that is desirable for the simulations. Therefore, more space should be explored.

Figure 5-24 to Figure 5-29 show the results of a population of 100 over a number of different generations.

Table 5-3: GA properties for test case 2 - 3

Population size	Maximum generations	$P_{c,max}$, $P_{c,min}$	$P_{m,max}$, $P_{m,min}$	Trials
100	20-1500	0.95, 0.80	0.08, 0.03	10

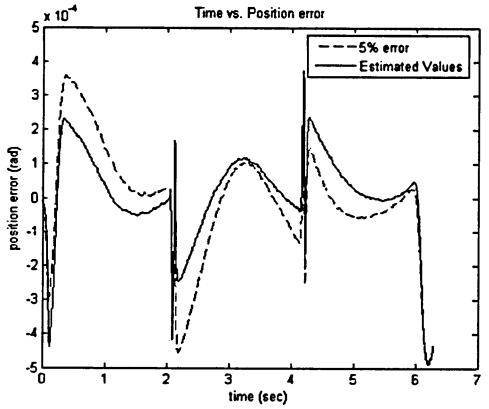


Figure 5-24: Response for 100 population over 20 generations

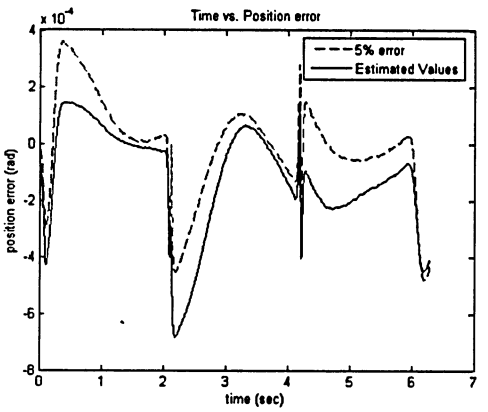


Figure 5-25: Response for 100 population over 50 generations

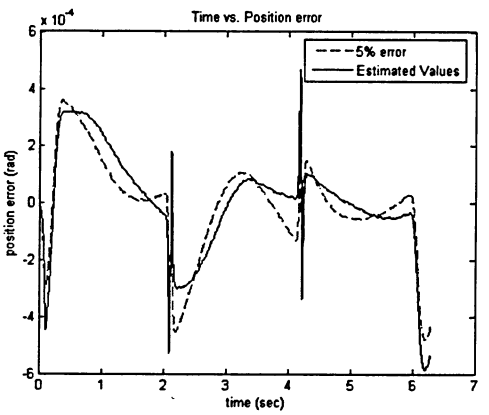


Figure 5-26: Response for 100 population over 100 generations

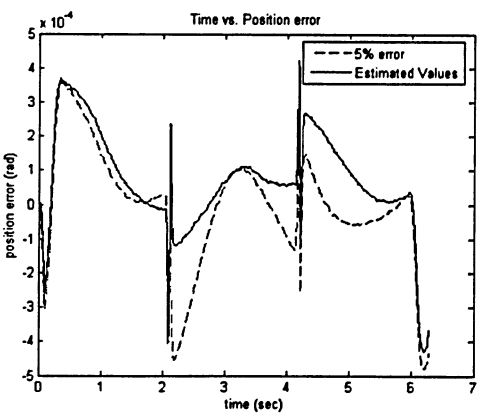


Figure 5-27: Response for 100 population over 500 generations

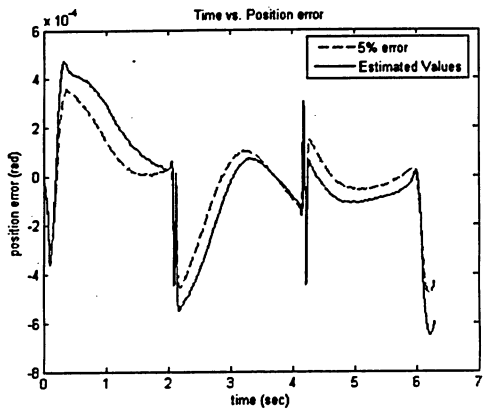


Figure 5-28: Response for 100 population over 1000 generations

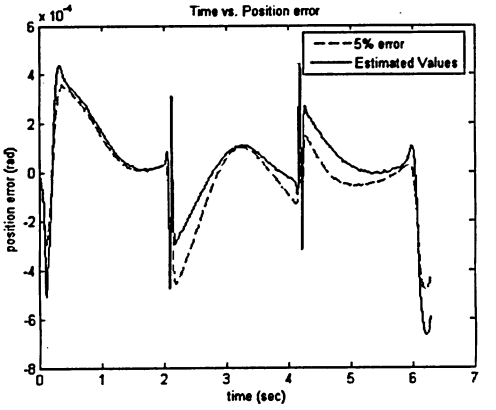


Figure 5-29: Response for 100 population over 1500 generations

Once again, there is a general trend that as the number of generations increases, the overall solution also increases. It is also notable that between 1000 generation and 1500 generations, the solution is not improving a great deal. The solutions are almost within the desired bound but it would seem prudent to search further still.

Figure 5-30 to Figure 5-35 show the results of a population of 200 over a number of different generations.

Table 5-4: GA properties for test case 2 – 4

Population size	Maximum generations	$P_{c,max}$, $P_{c,min}$	$P_{m,max}$, $P_{m,min}$	Trials
200	20-1500	0.95, 0.80	0.08, 0.03	10

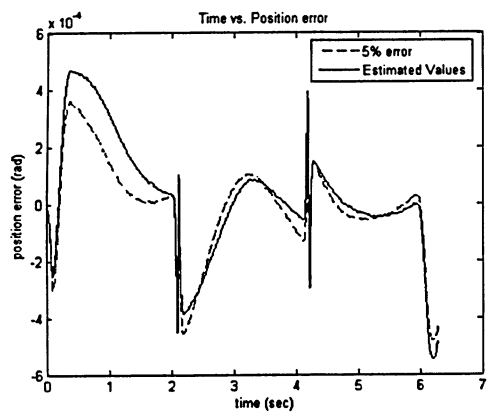


Figure 5-30: Response for 200 population over 20 generations

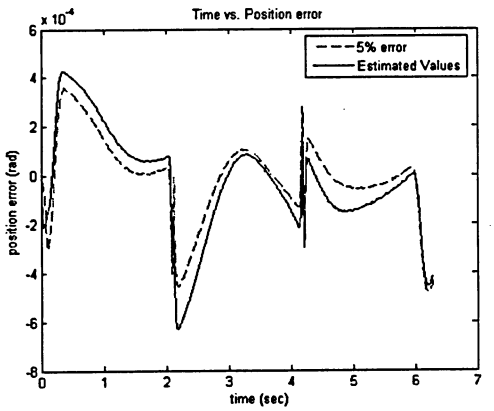


Figure 5-31: Response for 200 population over 50 generations

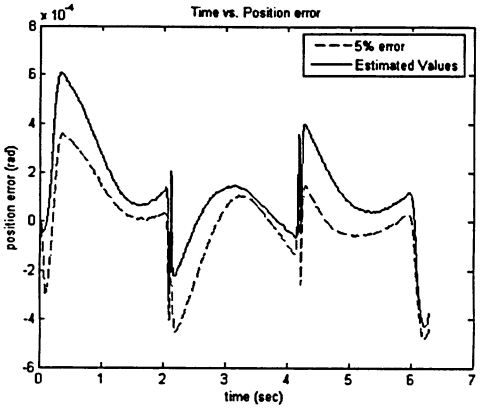


Figure 5-32: Response for 200 population over 100 generations

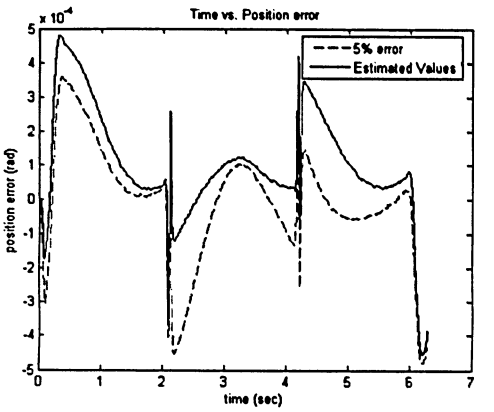


Figure 5-33: Response for 200 population over 500 generations

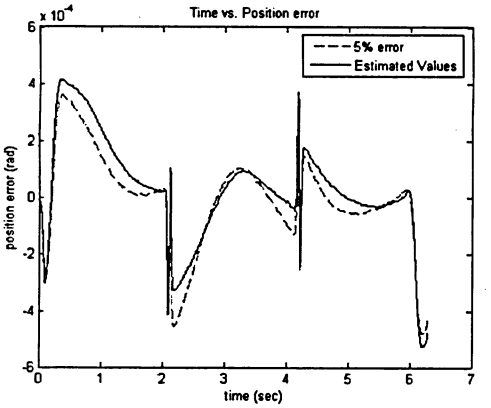


Figure 5-34: Response for 200 population over 1000 generations

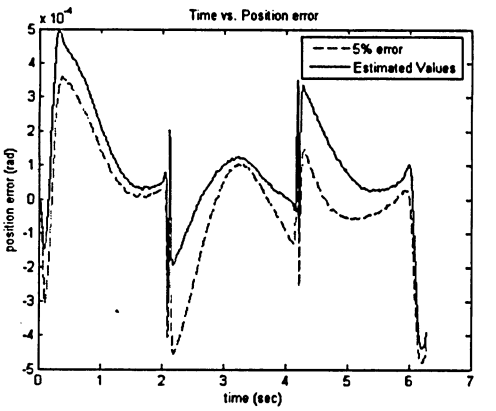


Figure 5-35: Response for 200 population over 1500 generations

The results show that as population increases and the number of generations increase, the parameter estimation improves. The case of an initial population of 200 with 1500 generation has the least amount of tracking error compared to all other cases. The estimated GA tracking error shape also closely resembles the 5% error curve. This would indicate that all of the parameters are being closely identified. In the instances where there is a large spike, it indicates that at least one parameter has been poorly identified.

In all cases, the parameter that is most difficult to estimate is the static friction parameter. This parameter has the greatest effect on the performance at the zero crossing positions. An explanation for the poor estimation of this parameter likely lies in the simulation of the torque sensor. In order to simulate the torque sensor, the velocity signal has to be differentiated. In the simulations, this should not be a problem, as one would expect the velocity signal not to be

noisy, as it would in a real system. However, at very low velocities there some round-off errors which cause the velocity to have small discontinuities. When differentiated, these amplify to somewhat larger discontinuities. While this only occurs for a few milliseconds at the zero crossing, this is the range where the static friction is most dominant in the penalty function, so it is most affected.

5.5 GA vs. Random Search

A GA is a type of random search algorithm. Generally compared to a totally random search, one would expect the GA to be much more efficient. In this case, it was desirable to achieve a heuristic solution that is within 5% of the actual solution. In order to compare the efficiency of the GA to a random search algorithm, it is necessary to break the domain for each parameter up into sections of 5%, then calculate the probability of finding each parameter within 5% of the actual solution. For example, in the GA, the value for Coulomb friction coefficient was thought to be in the domain [0,50] Nm. The value for the simulations was chosen to be 25 Nm. Therefore, any solution in the range of [23.75, 26.25] would be considered as a good solution. If the total domain for that parameter is broken down into sections of 5%, or in this case 2.5 Nm, then there would be 20 possible solutions. This gives a random chance of 1 in 20 of finding a good solution. The other domains are similarly selected such that each has a 1 in 20 chance of being within 5% of the heuristic solution. In order to find a heuristic solution randomly, the probability would be $\left(\frac{1}{20}\right)^6 = \frac{1}{6.4\times10^7}$. The GA with a population of 200 and a maximum generation of 1500 explored 3×10^5 solutions to find the heuristic solution. Therefore, the GA has an order of magnitude of 2 advantage to find the solution. Clearly, this is more efficient than a random search, which is consistently supported in the GA literature.

Chapter 6

Experiment

In order to confirm the hypothesis, it is essential to conduct experiments on a physical plant. The simulations suggest the parameter estimation algorithm will work, but experiments must be done to verify this. The experiments should demonstrate how applicable this algorithm could be to a practical setting and may even suggest a commercial viability.

6.1 Experimental Setup

As mentioned previously, the overall project is on a modular and reconfigurable robot. The experiments conducted for this thesis will involve one joint module only. The joint has several major components that will be outlined. The components will be divided into two major categories, the joint hardware and the joint software and communication. The following sections will give a brief description of each component.

6.1.1 Experimental Hardware

- Motor: Moog brushless DC motor model BN-42-33EU-03. The motor has a peak torque of 8.82 Nm, a rated speed of 4710 RPM, a rated torque of 1.42 Nm and consumes a rated power of 697W.
- Gear: HD Systems harmonic drive CFS-32-100-2A-GR-IV-SP-A1228. The gear has a ratio of 101:1, a rated torque of 137 Nm a max momentary torque of 647 Nm and a maximum input speed of 4800 RPM.
- Encoder: Torque Systems HS15-05/05-2000-0-04-T5-01. The encoder has a resolution of up to 5000 counts per revolution. The input voltage is 5 to 26 volts DC with a frequency of 500 KHz.
- Power Supply: Sorenson model DCS 80-37E. The power supply is capable of a voltage of 0-80, and amperage of 0-37. The input: 190-250 VAC, three phase, 14A typical, 47-63 Hz.

- Joint Torque Sensor: Is composed of 4 full Wheatstone bridges. The strain gauges are from Kyowa Electronic Instruments model KFG-1-120-D16-11. The gauges are 1mm long with a resistance of 120Ω. The gauges have an excitation of +/- 5V and can produces up to 20mV output.
- Amplifier: Phoenix Systems, model PN5603007. Strain Gage amplifier supports strain gages/load cells with resistances from 120 to 20,000 ohms. It includes 1000 V, 3-way galvanic isolation between power supply, input, and output circuits. Module filters and conditions signals to eliminate unwanted signal noise with DIP switch selectable cutoff frequencies of 30 or 5000 Hz. Outputs are provided for voltage (0 to 10, +/-10, 0 to 5, or +/-5 V), and current loop (4 to 20 mA) operation.
- Link: The link is 50 cm long and 15 cm wide and 1.27 cm thick. The total mass of the link is 3.3 kg. The link is designed to carry a payload in order to create a torque signal to feedback to the controller to test to see if the control law is effective.

6.2 Electrical Architecture

The overall electrical architecture is shown in Figure 6-1. The personal computer (PC) is used to program the digital signal processor (DSP) board, which is a distributed controller. Once the program is downloaded into the DSP board, it is capable of controlling the driver, which drives the motor. The motor turns the gear head and produces two feedbacks, an encoder for position and the torque sensor. The encoder is a digital signal and is sent to the driver. The driver uses the position signal to calculate the velocity. These two pieces of information are fed back to the DSP board via a Can bus which is a high speed communication network device. The DSP also receives the analog signal from the amplifier which is necessary to boost the weak torque signal into the +/-3V range that the DSP board needs to do the analog to digital conversion. The DSP board now has the torque signal, the position and velocity signals and is able to calculate how much torque to send to the driver.

6.2.1.1 Driver

The driver selected for this experimental setup is from Elmo Motion Control. The drive is a Cello digital servo drive. The driver is capable of doing position, velocity and current control. This unit was selected because the current loop is open, so it is possible to program a current control law and implement it with this driver. The drive has a standard serial port, RS232 and a

Controller Area Network (CAN) communication port available. For the purposes of the MRR project, only the CAN communications are necessary.

6.2.1.2 DSP Board

The DSP board serves as a distributed controller. For the MRR project, there would be a DSP board on each joint, which would be capable of controlling each joint in a distributed manner. The DSP is responsible for controlling the joint in real-time. The communication from the PC to the DSP board is carried out through the RS232 COM port. Technosoft provides software that is capable of programming the control law which is desired onto the DSP board. The software programming language is C.

6.2.2 CANOpen Protocol

In order to ensure timely control, it is necessary to have high speed communication between the DSP board and the driver. This is accomplished by using the CAN bus and a CANopen protocol. The CANopen protocol has one master, in this case the DSP board, and any number of slave nodes. For the purposes of this experiment, the only slave node needed is the motor drive. The overall communication architecture is shown in Figure 6-1. The CANopen bus speed can be set as high as 1 Mbps, for a single joint this speed is not necessary. The speed that was used was 500 kbps. The limiting factor in determining the speed at which the joint is commanded was determined by the driver. The driver is designed to be a position control but had an option to keep the torque loop open. This allows for a control law to be executed outside of the driver, which is desirable for the research on the MRR. The drawback of this feature is that the drive only reads the CAN bus torque commands in its processor idle loop. The processor idle loop is statistically read every 1.5 ms. While it is possible to broadcast torque commands more frequently, based on the bus speed, the DSP speed, it will not ensure that the drive will execute this command. Therefore, the control law is set to send a command to the drive every 2 ms. This is the shortest amount of time which guarantees a consistent execution.

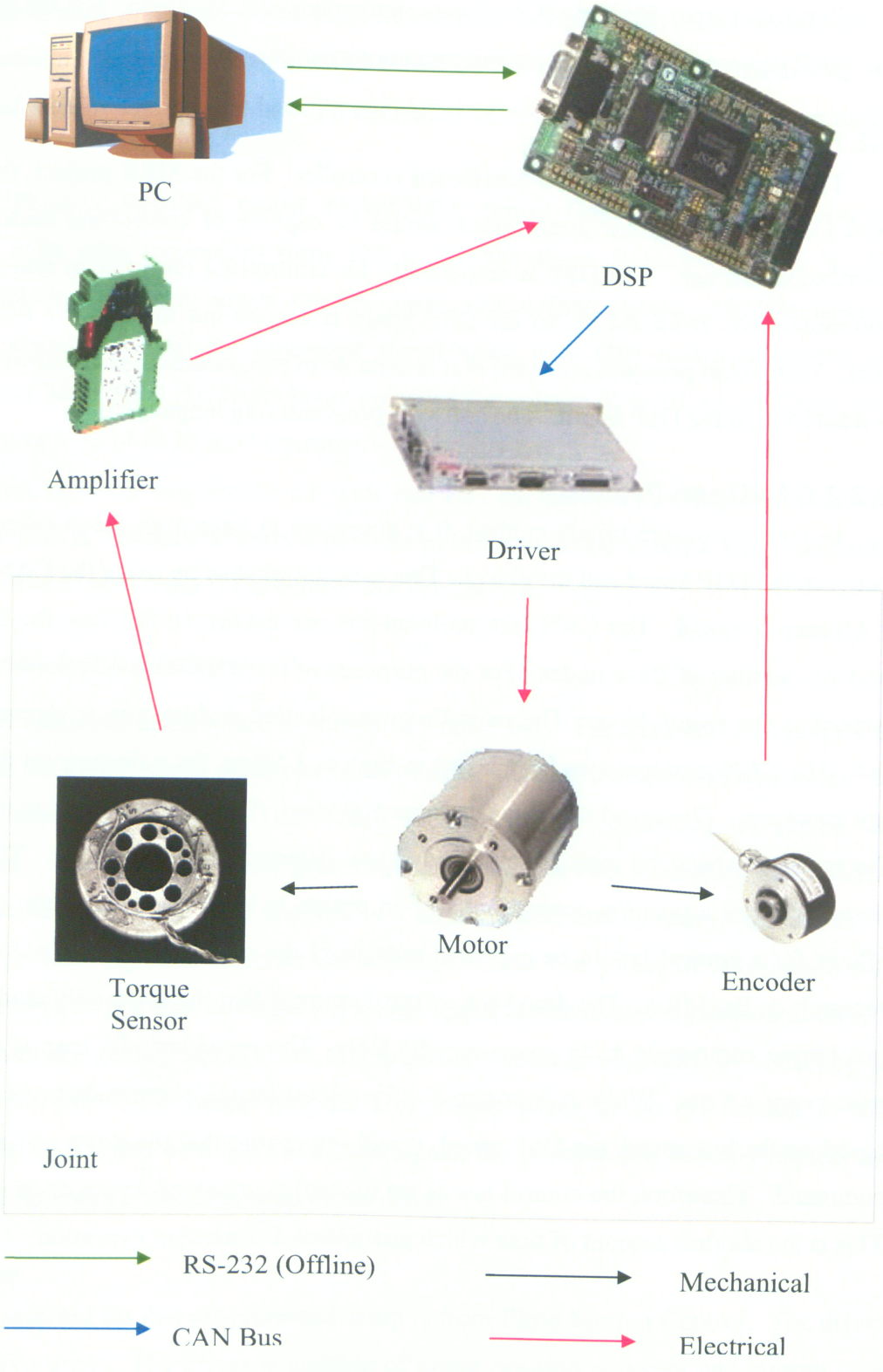


Figure 6-1: Electrical communication

6.3 Sensors

Two sensors are used in the experiment. There is the joint torque sensor, which has been described in section 2.2, and the position sensor, i.e., the incremental encoder. The incremental encoder is used for two purposes. The first is to measure the position. This is procedure is fairly straightforward. The encoder is electrically wired to the DSP and there is a subroutine on the Technosoft DSP board that increments (or decrements depending on the direction of rotation) a register as the motor shaft rotates. The torque sensor is an analog signal. In order for it to be used in the control algorithm, it must be converted to a digital signal. The conversion from analog to digital is also a subroutine supplied by the Technosoft DSP board. The board is capable of converting sixteen different channels. Since this application requires only one channel, the other 15 channels are used to over-sample the signal. Over-sampling the signal and averaging the result dramatically reduces the analog to digital conversion (ADC) noise as seen in Figure 6-2 and Figure 6-3.

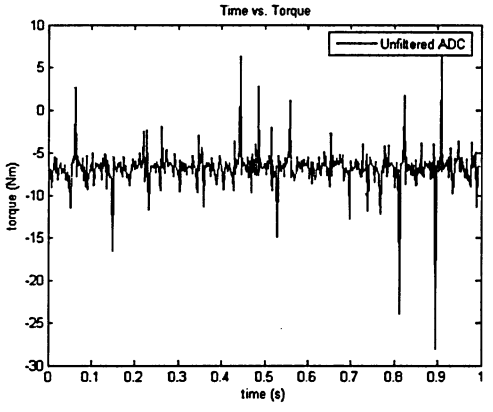


Figure 6-2: Unfiltered ADC signal

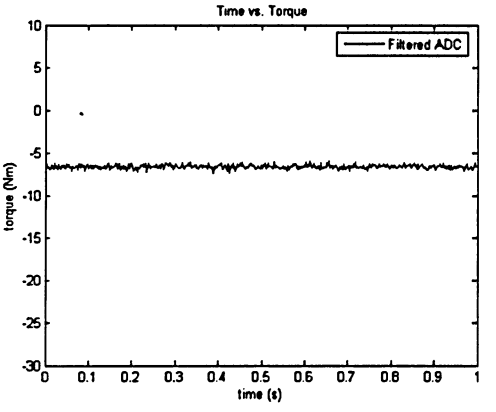


Figure 6-3: Filtered ADC signal

6.3.1 Velocity Estimation

The control law requires the velocity of the joint to be known. A tachometer can be used to directly measure the velocity, but there is not one available in this experimental setup. In order to determine the velocity, the signal from the position encoder is differentiated at each time step. At very low speeds this method can be inaccurate. The definition of very low speed depends on the resolution of the encoder. When only a few pulses are being measured in each time step, then partial increments of the encoder are lost and the result is a loss of accuracy in velocity measurement.

Liu et al. [12] proposed a method to estimate the velocity to overcome this problem. In the proposed method, if the encoder does not increment by a set number of pulses, or the time does not change by a pre-determined amount, then the velocity will remain the same. The resulting velocity is experimentally demonstrated to be more accurate than by directly computing the velocity at each time step. The method also allows the tuning of the estimation parameters in order to trade off the accuracy vs. the time delay. For this joint, it was found that the best estimation of velocity occurred when the minimum number of pulses used to estimate the velocity is set to 5 and the maximum amount of time used to estimate the velocity is 10 ms.

6.4 Experimental Joint

The experimental joint is shown in Figure 6-4. In this configuration, the link rotates in the vertical plane. It is also designed so it can be rotated 90 degrees and the link can be planar. This is part of the reconfigurable characteristics of the joint. The mass of the link is approximately 3 kg and can carry a weight of 100 N. The link was designed to be able to be easily removed and reconfigured to two positions; the position shown in Figure 6-4 and a configuration where the link is attached to the joint in the centre. The advantage of the first position is that weight added to the end will give the maximum amount of torque on the joint. Attaching the link in the middle would effectively cancel out the weight of the link, and allow the user to add the precise amount of weight that is desired. Also, if the joint were to be configured in a planar mode, attaching the link in the centre would cancel out torque created perpendicular to the axis of rotation. This mode could be important if the bearing in the joint is not functioning properly. In all of the experiments conducted for this work, the joint was configured exactly as seen in Figure 6-4.

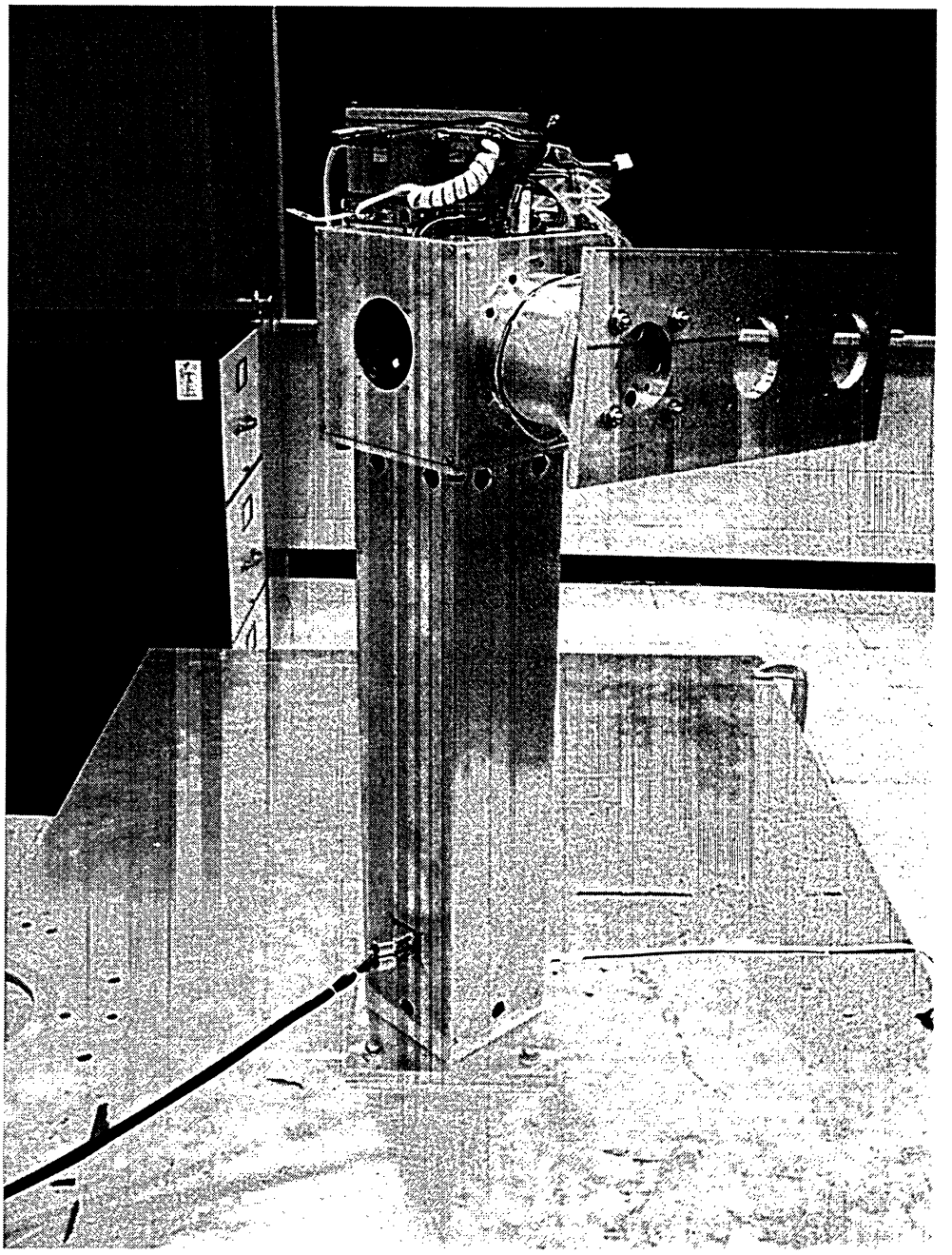


Figure 6-4: Experimental Joint

6.5 Experimental Results

The experiments were carried out with the following position, velocity and acceleration trajectories and are shown in Figure 6-5, Figure 6-6, and Figure 6-7, respectively. The chosen amplitude, A, of the motion is selected to be: $A = 4^{\circ}$.

$$q_d = \frac{A}{1.3}\left(\sin\left(\frac{3t}{2\pi}\right) - 0.5\sin\left(\frac{3t}{\pi}\right)\right), \dot{q}_d = \frac{A}{1.3}\left(\frac{3}{2\pi}\left[\cos\left(\frac{3t}{2\pi}\right) - \cos\left(\frac{3t}{\pi}\right)\right]\right),$$

$$\ddot{q}_d = \frac{A}{1.3}\left(-\left(\frac{3}{2\pi}\right)^2\left[\sin\left(\frac{3t}{2\pi}\right) - 2\sin\left(\frac{3t}{\pi}\right)\right]\right) \quad (6\text{-}1)$$

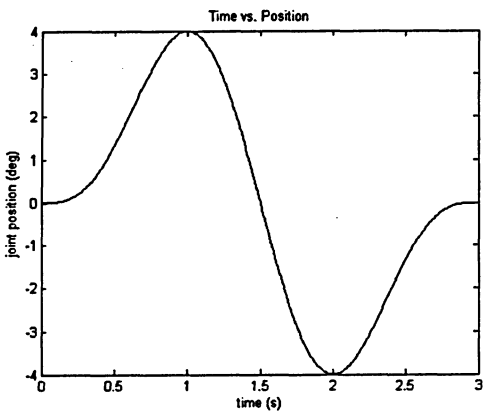


Figure 6-5: Desired joint position

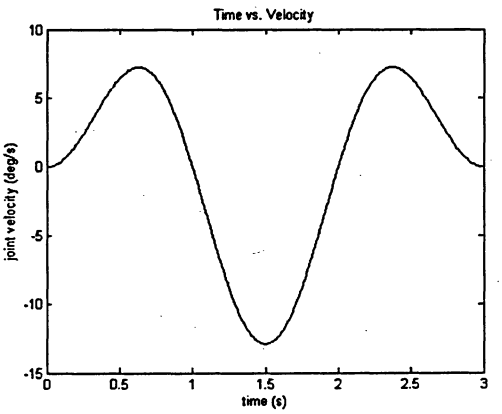


Figure 6-6: Desired joint velocity

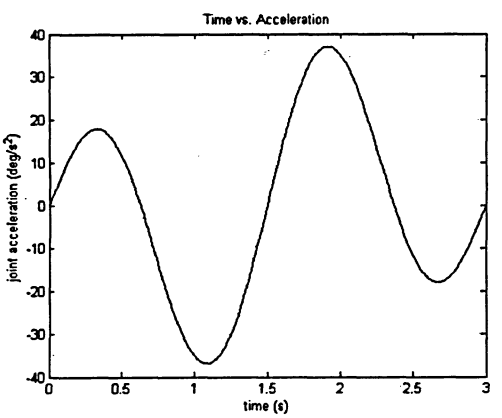


Figure 6-7:Desired joint acceleration

6.6 Experimental Effect of Friction Parameter Identification

As predicted by the simulations, poor parameter estimation adversely affects the experimental results. The control system remains stable, but the position tracking error increases when the parameters are poorly estimated. The control parameters are shown in Table 6-1. For the following experiments, the sensor characteristics gain and offset are assumed to be known, and further discussions are included later. For this experiment, several different combinations of friction parameters were compared. A combination of parameters found to have the best tracking is considered to be the good estimates and for the sake of comparison, another set of

parameters are selected to be the poor estimates, as listed in Table 6-2. The position tracking error for each case is shown in Figure 6-8.

Table 6-1: Experimental Control Parameters

Control Parameters	λ	k	ε	ρ
	80	0.005	0.9	0.05

Table 6-2: Friction Parameters

Parameters	Fc (Nm)	Fs (Nm)	Ft (s^2/rad^2)	B (Nm-s/rad)	Gain	Offset (Nm)
Good Estimates	20	10	110	20	3.2	5.2
Poor Estimates	10	40	300	50	3.2	5.2

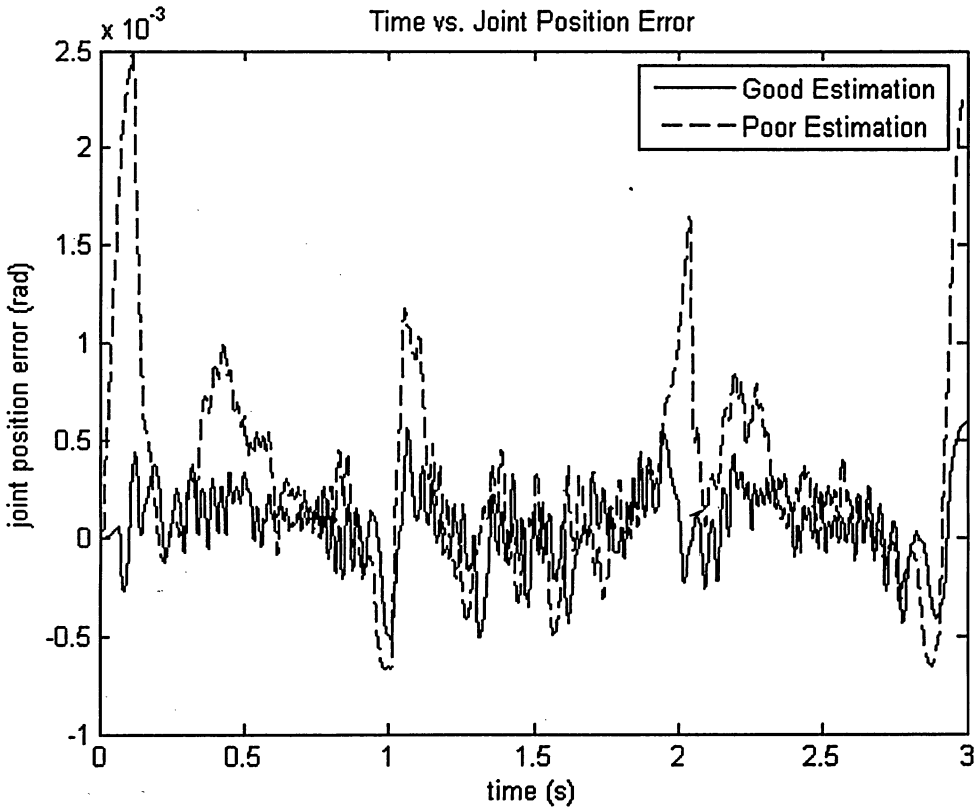


Figure 6-8: Position tracking for good estimation vs. poor estimation.

6.7 Joint Torque Feedback Assisted Control

The motor command is supplemented by including the joint torque sensor signal. The joint torque sensor measures the dynamic torque of the link without needing specific model

parameters pertaining to the link, i.e., the mass, the inertia or the acceleration. In the case where the trajectory is chosen such that gravity dominates, the sensor feedback greatly enhances the tracking control. The joint sensor feeds the signal back and cancels this force before it can cause an error in the tracking control. The effect of adding this term to the control law is shown in the position tracking error in Figure 6-9.

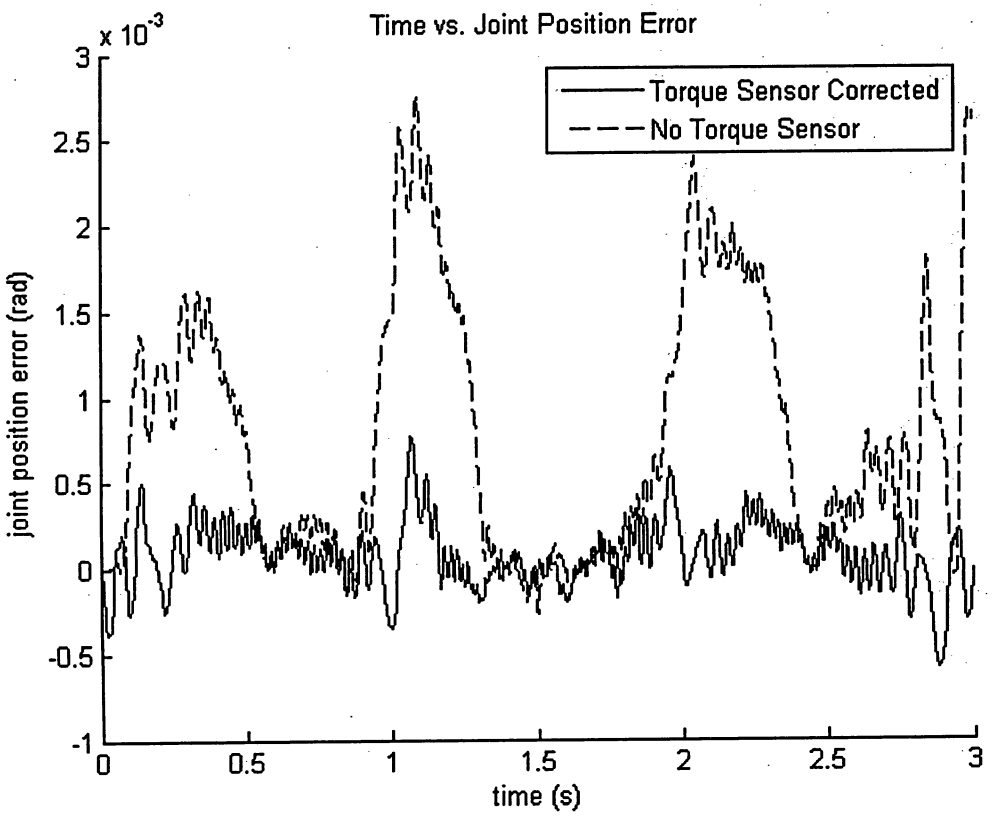


Figure 6-9: Position tracking for torque sensor correction vs. no torque sensor.

6.8 Effect of Poor Joint Torque Sensor Estimation

The joint torque sensor contains two parameters, the sensor gain and the sensor offset that can change over time. As these characteristics change, the performance of the joint is compromised. The sensor measurement used in the controller is no longer accurate.

6.9 GA Identification of Joint Parameters

Consider the situation where the user is first setting up the joint. In order to tune the controller to achieve stability, some simple experiments must be conducted. To implement the controller it is necessary to input some parameters. The initial estimation of the parameters may be based on experience with similar joints, or it may be based on values found in literature. For

the purposes of this discussion, that initial estimate will be referred to as unidentified parameters. Following the successful execution of an experiment, i.e., the experiment is carried out and the tracking error does not diverge, the corresponding data can be analyzed by the GA and the parameters can be identified. The result of this procedure is shown in Figure 6-10. The values for the initial estimate and the GA are shown in Table 6-3. This result suggests that the GA identification method is an effective way to estimate the friction and sensor parameters for the joint.

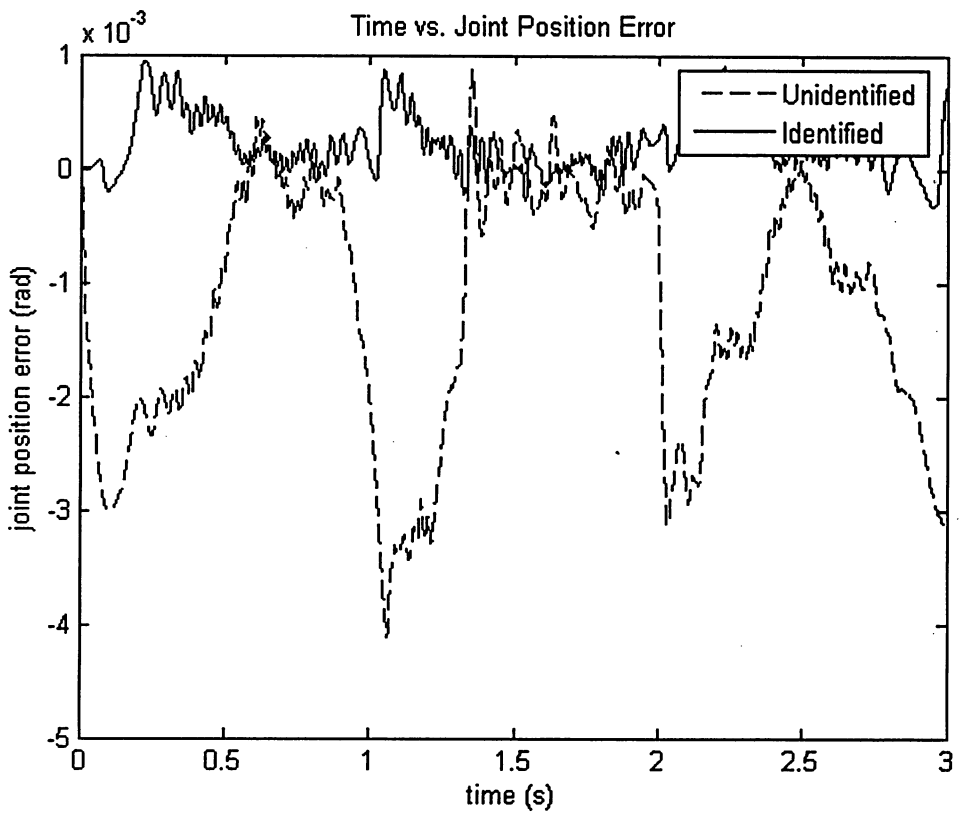


Figure 6-10: Parameter estimates vs. GA estimates

In order to find a baseline comparison for the GA, a series of trial and error experiments were conducted. In these experiments each parameter was varied individually and for each variation an experiment was conducted. The goal of this procedure was to identify the set of parameters with produced the least amount of position tracking error. The results of these experiments is compared with the result of the GA identified parameters in Figure 6-11.

Table 6-3: Friction Parameters

Parameters	Fc (Nm)	Fs (Nm)	Ft (s^2/rad^2)	B (Nm-s/rad)	Gain	Offset (Nm)
Initial Estimates	10	20	200	10	1	0
GA Identified Estimates	25	3	94	19	1.3	2.5

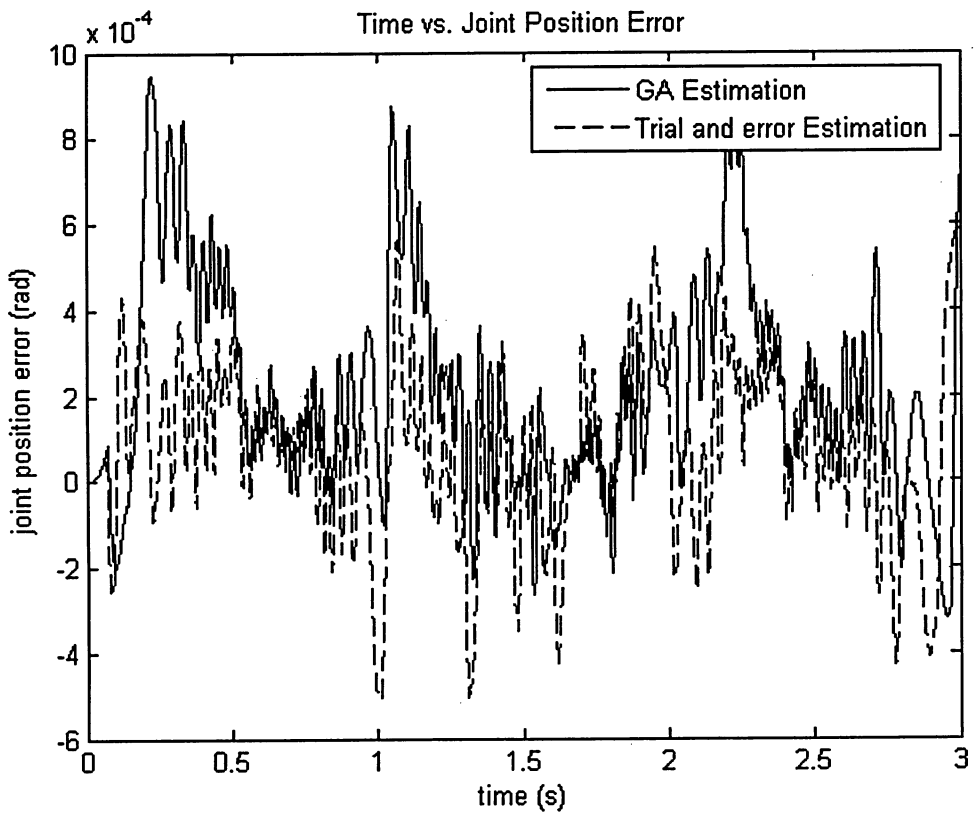


Figure 6-11: GA estimates vs. trial and error estimates.

6.10 Discussions

The genetic algorithm identification technique can be effective in identifying system parameters. However, in order for the algorithm to be effective, appropriate data must be collected. Clearly, the system dynamics must be excited. If the system is not excited appropriately, then the model no longer accurately represents the system, and the identification

process is no longer possible. A clear example of this is using a high speed trajectory; at high speeds Stribeck friction is not present.

As experimental tests were done, some more subtle problems appeared while collecting data, for example, the flexspline and wave generator influence on the joint velocity. While it would be desirable for the joint to move at a steady velocity, this is impossible with a harmonic drive. The continuous flexing of the flexspline causes a speed variation while the joint is in motion. This torque ripple creates a velocity error, this error is fed back through the controller and the controller increases and decreases the torque accordingly. The effect of this ripple can be seen in Figure 6-12 and Figure 6-13. The resulting friction map is shown in Figure 6-14. The GA is still able to find the best fit data which represents the system, despite the friction map containing these ripples.

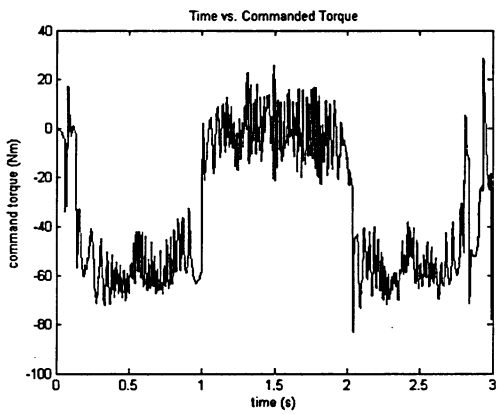


Figure 6-12: Closed loop commanded torque

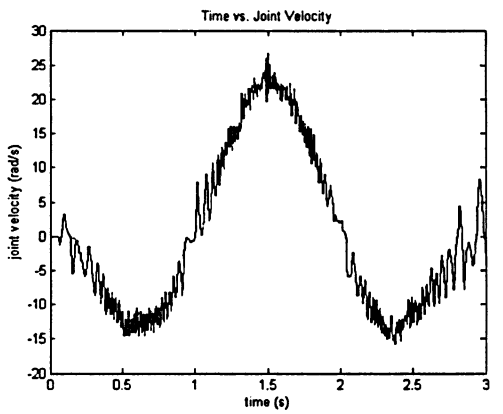


Figure 6-13: Closed loop tracking velocity

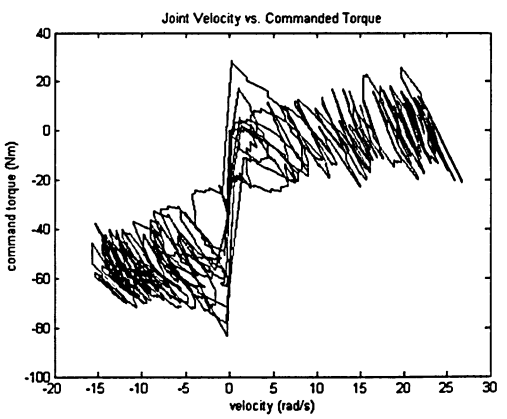


Figure 6-14: Closed loop friction map

One problem that should be addressed is the initial estimation of the parameters. While the control law suggests any estimation should be stable, i.e., the tracking error does not diverge, in

reality of course this is not true. Physical limitations, such as motor saturation, will cause the tracking error to diverge and limit the range over which the parameters can be estimated. In order to get an initial estimate, users can use their experience or it may be possible to run the joint in an open loop configuration. Figure 6-15 and Figure 6-16 show the open loop torque and open loop velocity, respectively. The effect of the torque ripple can still be seen in the figures, but since the velocity error is not being fed-back to the controller, the resulting torque is much smoother. Figure 6-17 and Figure 6-18 show the maps for the command torque versus the velocity and the command torque with the sensor subtracted plotted against the velocity, respectively. Since the torque sensor measures the effect of the link, the map for the command torque without the torque sensor vs. the velocity would be the best representation of the friction in the system.

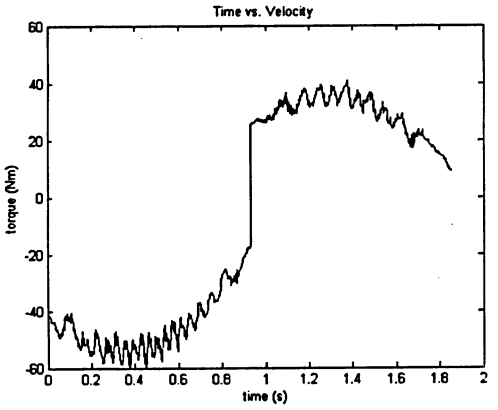


Figure 6-15: Open loop commanded torque

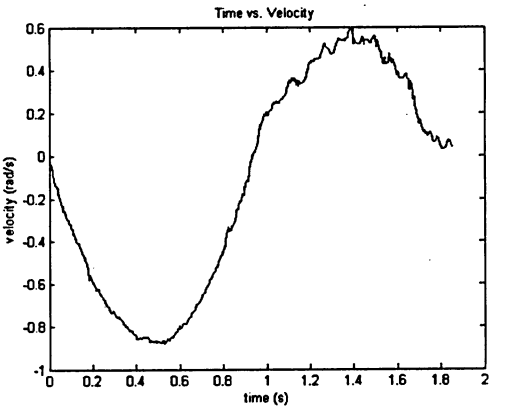


Figure 6-16: Open loop joint velocity

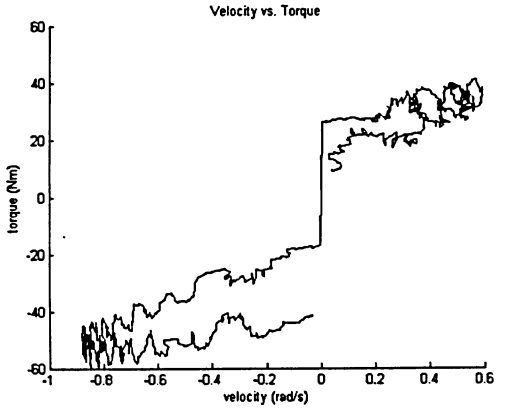


Figure 6-17: Open loop friction map with torque sensor

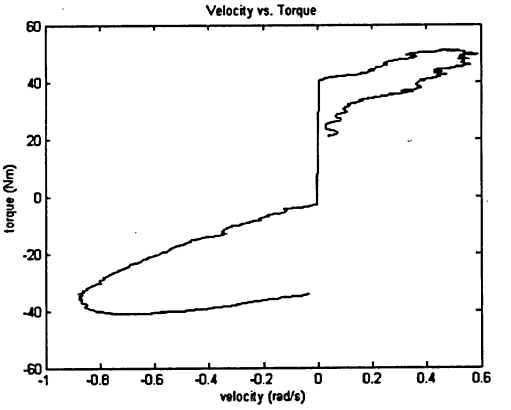


Figure 6-18: Open loop friction map with the torque sensor subtracted

This method is effective for getting the initial estimate but not very practical for updating the parameters. Running the joint in an open loop manner is not desirable and may not even be possible. It requires a great deal of trial and error to find the correct amount of torque to move the joint in the desired fashion. Also, the joint moving in an uncontrolled fashion can be unsafe and has the potential to damage the joint. Once a joint is integrated with control system, it normally runs with closed-loop control only.

Chapter 7

Conclusions and future work

The purpose of this thesis was to determine system parameters of a mechatronic joint via GA. The identification of the system parameters is important to ensure stable, precise motion control. A real coded GA technique was utilized to identify the friction parameters as well as the joint torque sensor parameters. The effectiveness of the technique was demonstrated in a laboratory experiment.

The real coded GA is a much more efficient way to explore the search space compared to a binary coded GA or a random search. The real coded GA used a dynamic adaptive crossover method to determine the probability of crossover. The SBX crossover method was used to generate the children solutions. The fitness function for the GA was designed to have the ability to trade off some of the accuracy in order to save computation time. This feature was found to be useful when evaluating the GA over large time spans of data.

The key roll of the simulations was to develop the fitness function so that it would be suitable for the parameter identification problem. The simulations were a fast and cost effective way to evaluate the behaviour of the GA identification process. Once the fitness function was developed, various parameters of the GA were experimented with to find the most optimal for finding the friction and sensor parameters. The most notable parameters were the GA population and the total number of generations. The simulations indicated that this GA would be capable of identifying the system parameters within 5% of their actual values if the population and generations were selected to be 100 and 1000, respectively. This information provided a starting point for the experiments

The experimental part of this work was essential. Although the experiments are time consuming and expensive, they demonstrate the practicality of this identification technique. The joint was built and the control law was implemented. As predicted by the simulations, the

presence of the joint torque sensor feeding back to assist the compensation technique reduced the tracking error. The tracking error was also effected by the variation of the parameter estimates. First, open loop experiments were conducted on the joint and the parameters were estimated. Once the parameters were used in the closed loop control, the tracking error was found to be reduced. Running a joint in open loop configuration could be impossible, or impractical, even dangerous, so it was necessary to conduct the trial using information from a closed loop test. The closed loop data is not as clean because the torque ripples cause the velocity signal to have a ripple which feeds back to the controller as an error. Despite the noisy friction map, the GA is able to identify the parameters and improve the tracking error.

Part of the work of this thesis was laying the groundwork for future work. The first joint module of the MRR is complete. The identification technique can be updated to include more system parameters. It would be useful to try to use another friction model and develop a new control law and use the parameter estimation technique to identify these other parameters. The LeGru model uses 7 parameters, and the GA could be modified to include these other parameters. While the model used in this thesis assumed the joint is rigid, in fact the flexspline is flexible, which is how it is able to measure torque. Using a more accurate model to represent the joint and implementing this model into the control law could improve the performance of the joint. Of course a model of the joint would require further parameter estimations, such as the spring constant of the flexspline. The GA could be used to identify this parameter as well.

Appendix

MATLAB Script Files for Simulation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simulation of a single degree of freedom joint with friction
% and a link which is affected by gravity
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear
%Friction Values are from Friction_Values.txt, these are for time step %1
only
x=load_values; %Loads estimated friction values from a file which are
% estimated via the GA
Fs_hat=x(1);
Fc_hat=x(2);
Ft_hat=x(3);
B_hat=x(4);
Ts_Offset_hat=x(5);
Ts_Gain_hat=x(6);
%Set the system parameters
TS_Offset=40;
TS_Gain=2.4;
Gear_Ratio=101;
Rotor_Inertia=.0002472;
Full_Scale=650;
Ripple=0.009; % 0.9% ripple
Nonlinearity=0.0014; %0.14% non-linearity
sdnoise =0.5;
Length=0.5; %meters
Mass=3; %kg
Payload_mass=0;
Link_I=1/3*Mass*(1/2)*Length^2;%+Payload_mass*Length;
Gravity=9.817;
sim single_dof_const_grav
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Integrates the torque signal.  Calculates the join dynamics.
% Based on S-Function template.  Input is the torque, the output is the %
velocity and position of the link.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [sys,x0,str,ts] = sfuntmpl(t,x,u,flag)

switch flag,
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
  case 1,
    sys=mdlDerivatives(t,x,u);
  case 2,
    sys=mdlUpdate(t,x,u);
  case 3,
    sys=mdlOutputs(t,x,u);
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9,
```

```
    sys=mdlTerminate(t,x,u);
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
%
%=============================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 2; %position, velocity
sizes.NumInputs      = 1; %command torque
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;   % at least one sample time is needed
sys = simsizes(sizes);
%
% initialize the initial conditions
%
x0  = [0 0];
%
% str is always an empty matrix
%
str = [];
%
% initialize the array of sample times
%
ts  = [0 0];
% end mdlInitializeSizes
%
%=============================================================================
=
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================================
=
%
function sys=mdlDerivatives(t,x,u)
Fc=.22;%Nm
Fs=.33; %Nm
B=.08; %Nm s/rad
Im=.0002472; % Rotor Inertia kg m^2
Ft=150; %s^2/rad^2
Gear_Ratio=1;%101;
Length=.5; %meters
Mass=3; %kg
%Payload_mass=12;
Link_I=1/3*Mass*(1/2)*Length^2;%+Payload_mass*Length;
Gravity=cos(x(1))*Length/2*Mass*9.817;
Friction=(Fc+Fs*exp(-Ft*x(2)*x(2)))*sign(x(2));
Fq=0.05*sin(3*x(1));
q2dot=(u-Friction-Fq-B*x(2)-Gravity)/(Im*Gear_Ratio+Link_I);
sys(1)=x(2);
sys(2)=q2dot;
% end mdlDerivatives
```

```
function sys=mdlUpdate(t,x,u)
sys=[];
% end mdlUpdate

function sys=mdlOutputs(t,x,u)
x;
sys = x;
% end mdlOutputs

function sys=mdlGetTimeOfNextVarHit(t,x,u)
sampleTime = 1;     %  Example, set the next hit to be one second later.
sys = t + sampleTime;
% end mdlGetTimeOfNextVarHit

function sys=mdlTerminate(t,x,u)
sys = [];
% end mdlTerminate



% noise function : gaussian distribution function
function [sys,x0,str,ts]  = noise_func(t,x,u,flag,sd_sig)
%VDPM Example M-File S-function implementing the Van der Pol equation
%   See sfuntmpl.m for a general S-function template.
%
%   See also SFUNTMPL.

%   Copyright 1990-2001 The MathWorks, Inc.
%   $Revision: 1.19 $

switch flag,

  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0
    [sys,x0,str,ts] = mdlInitializeSizes(sd_sig);

  %%%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%%
  case 1,
    sys = mdlDerivatives(t,x,u);

  %%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%
  case 3,
    sys = mdlOutputs(t,x,u,sd_sig);

  %%%%%%%%%%%%%
  % Terminate %
  %%%%%%%%%%%%%
```

```
  case 9
    sys = []; % do nothing

end
 %
%=============================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================
%
function [sys, x0,str,ts] = mdlInitializeSizes(sd_sig)

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

sys = simsizes(sizes);

x0  = [];
str = [];
ts  = [0 0];  % continuous sample time: [period, offset]

%end mdlInitializeSizes
%=============================================================
% mdlDerivatives
%=============================================================
function sys = mdlDerivatives(t,x,u)
%end mdlDerivatives

function sys = mdlOutputs(t,x,u,sd_sig)
% sd_sig: standard deviation

theta = 8*atan(1.0)*rand(1);
temp = -1;
while (temp<=0) temp = 1-rand(1);
end
r = sqrt(-2*log(temp));
sys=sd_sig*r*cos(theta);

%end mdlOutputs
```

MATLAB Script Files for genetic algorithm

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function is designed to use the information collected by either %the
simulation or experimental sensors and use it to estimate the %parameters.
It calls the GA as well as plots the frictional maps
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num_trials=3; %Set the number of trials to be averaged
totalpop=[10]; %a vector of the number of populations
totalmaxgen = [20]; %a vector of the number of maximum generations
for u=1:size(totalpop,2)
```

```
    popsize=totalpop(u)
    for r=1:size(totalmaxgen,2)
        maxgen=totalmaxgen(r)
        for m=1:num_trials
         close all
            j=0;
            figure(1);
            plot(Velocity,Torque,'k');
            %Evaluate the GA and estimate the parameters
            tic
            %to see a graph or check the fitness, comment out the GA call and
            substitute the parameters directly
            [Parameter_Values] = multi_parameter_real_grav(Velocity,Position,
            Sensed_Torque, Torque, time, popsize, maxgen,mutation_rate);
            %Parameter_Values=[Fc Fs Ft B Offset Gain]
            toc
            j=0;
            Friction=f_model(Parameter_Values,Velocity,Position, Torque
            ,time);
            hold on;
            plot(Velocity,(Torque-Sensed_Torque),'g')
            plot(Velocity,Friction,'r');
            legend('Command Torque','TC-TS','F');
            xlabel('Vel(rad/s)');
            ylabel('Torque (Nm)');
            Estimated_Torque=integral_friction_model_grav(Parameter_Values,Ve
            locity,Position, Torque ,time);
           % Plot the results to compare how accurate the GA is
            Ts_Int=TS_Integral(Sensed_Torque,time);
            Tc_Int=TS_Integral(Torque,time);
            figure(2);
            plot(time,Ts_Int);
            hold on;
            plot(time,Estimated_Torque,'r');

            x=obj_func_grav(Parameter_Values,Velocity,Position,Sensed_Torque,
            Torque,time)
            for k=1:6,
                if m>1
                  avg_values(m,k)=avg_values(m-1,k)+Parameter_Values(k);
                else
                    avg_values(m,k)=Parameter_Values(k);
                end
            end
        end
        for k=1:6
            parm_avg(k)=avg_values(m,k)/m;
            total_parm_avg(u,r,k)=parm_avg(k);
        end
        parm_avg
        %total_parm_avg(u,r)=parm_avg;
    end
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Genetic Algorithm
% Inputs are: velocity, position, command torque, sensed torque time,
```

```
% population size, and the maximum generations.
% Ouputs are: The estimated parameter values
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Parameter_Outputs] = multi_parameter_grav(Velocity,Position,
Sensed_Torque, Torque, Time, popsize, maxgen)
%%%%%%%%%%%%%%%%%%Parameter Definition%%%%%%%%%%%%%%%%%%%%%%

NParm=6;  %convention of N* is "Number of" ie NParm = Number of Parameters

%%%%%%%%%%%%%%%%%%Population Definition%%%%%%%%%%%%%%%%%%%%%%
%-----------Crossover parameter--------------------------------------
neta=4;
%----------Mutuation Parameters (non-uniform)-----------------------
mutation_rate=.03;
B=2;
Vmin=0.7;
Vmax=0.95;
Prob_cross(1)=0.80;
Prob_cross(2)=0.9;
Prob_mut(1)=0.03;
Prob_mut(2)=0.15;

%%%%%%%%%%%%%%%%%%%%%%%%%%Solution%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------Define Search Domain-----------------------------------
Total_Scale=0;
Parm_Scale(1,1)=0;%Coloumb
Parm_Scale(1,2)=50;
Parm_Scale(2,1)=0;%Static
Parm_Scale(2,2)=40;
Parm_Scale(3,1)=0; %stribeck
Parm_Scale(3,2)=200;
Parm_Scale(4,1)=0;%viscous
Parm_Scale(4,2)=40;
Parm_Scale(5,1)=-30;%Sensor Offset
Parm_Scale(5,2)=30;
Parm_Scale(6,1)=.1;%Sensor Gain
Parm_Scale(6,2)=5;
for i=1:NParm
    Max_Parm(i)=Parm_Scale(i,2);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Initialize%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Population%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pop.sumfitness=0;
gen=1;
for i=1:popsize,
    pop(i).fitness=0;
    for m=1:NParm
        pop(i).chrom(m)=Parm_Scale(m,1)+rand*(Parm_Scale(m,2)-
(Parm_Scale(m,1)));
    end
    pop(i).fitness=obj_func_grav(pop(i).chrom,Velocity,Position,
Sensed_Torque, Torque, Time);

    fitness(i)=pop(i).fitness;
    pop(1).sumfitness=pop(i).fitness+pop(1).sumfitness;
```

```
end
avg(gen)=pop(1).sumfitness/popsize;
pop(2).sumfitness=0;
for i=1:popsize, %normalize fitness
    pop(i).fitness=(1-pop(i).fitness/pop(1).sumfitness)*pop(1).sumfitness;
    fitness(i)=pop(i).fitness;
     if (i==1)
        Most_Fit(gen)=pop(i).fitness;
        Global_Fit=pop(i).fitness;
        Global_Max=pop(i).chrom;
    else
        if (pop(i).fitness>Most_Fit(gen))
            Most_Fit(gen)=pop(i).fitness;
            Most_Chrom(gen,1:NParm)=pop(i).chrom;
        end
    end
    pop(2).sumfitness=pop(2).sumfitness+pop(i).fitness;

end
total_relfit=0;
for i=1:popsize
    rel_fitness(gen,i)=pop(i).fitness/pop(2).sumfitness;
end
%%%%%%%%%%%%%%Generation Iteration%%%%%%%%%%%%%%%%%%%%
j=1;
while (gen<maxgen),
    gen=gen+1
    %avg(gen)=sumfitness/popsize;
    %Determine Probability of Mutation and Crossover Based on Genetic
Diversity
    Gdm=(pop(1).sumfitness-Most_Fit(gen-1))/(avg(gen-1));
    gdmmm(gen)=Gdm;
    newpop(1).sumfitness=0;
    newpop(2).sumfitness=0;
    [Pc Pm] = nonuniform(Prob_cross, Prob_mut, Vmin, Vmax, Gdm);
    %%%%%%%%%%Roulette Wheel Selection%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:2        :popsize,
        mate1=select(popsize, pop(2).sumfitness, pop);
        mate2=select(popsize, pop(2).sumfitness, pop);
        mates(gen, i)=pop(mate1).fitness;
        mates(gen, i+1)=pop(mate2).fitness;
%%%%%%%%%%%%%%%%Crossover%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %         Crossover each parameter of the selected mates.
        if (flip(Pc)==1)
            for n=1:NParm
                [newpop(i).chrom(n) newpop(i+1).chrom(n)]=
dynamic_adaptation_crossover(min(pop(mate1).chrom(n), pop(mate2).chrom(n)),
max(pop(mate1).chrom(n), pop(mate2).chrom(n)),neta,Max_Parm,n);
            end
        else
            newpop(i).chrom=pop(i).chrom;
            newpop(i+1).chrom=pop(i+1).chrom;
        end
    end

    for i=1:popsize,
```

```
        %%%%%%%%%%%%%%%%%%%%%%%%Mutation%%%%%%%%%%%%%%%%%%%%%%%%
        if (flip(Pm)==1)

            newpop(i).chrom=mutate_real(newpop(i).chrom,Parm_Scale,gen,maxgen
            ,B,NParm);
        end
        %%%%%%%%%%%%%%%%%%%%%%%%Fitness%%%%%%%%%%%%%%%%%%%%%%%%

      newpop(i).fitness=obj_func_grav(newpop(i).chrom,Velocity,Position,Sense
      d_Torque, Torque, Time);
        fitness(i)=newpop(i).fitness;
        newpop(1).sumfitness=newpop(1).sumfitness+newpop(i).fitness;
    end
    avg(gen)=newpop(1).sumfitness/popsize;

    %%%%%%%%%%%%%%Normalize the Fitness%%%%%%%%%%%%%%%%%%%%
    for i=1:popsize
        newpop(i).fitness=(1-
newpop(i).fitness/newpop(1).sumfitness)*newpop(1).sumfitness;
        newpop(2).sumfitness=newpop(2).sumfitness+newpop(i).fitness;

        %%%%%%%%%%%Find the Best Fit for this Generation%%%%%%%%%%%%
        if (i==1)
            Most_Fit(gen)=(newpop(i).fitness);
            Most_Chrom(gen,1:NParm)=newpop(i).chrom;
        else
            if newpop(i).fitness>Most_Fit(gen)
                Most_Fit(gen)=newpop(i).fitness;
                Most_Chrom(gen,1:NParm)=newpop(i).chrom;
            end
        end
    end
    pop=newpop;
end

%=========================GA Finished==============================

%%%%%%%%%%%%%%%%%%%%%%%%Find Global Max%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:maxgen
    if i==1
    Global_Fit=obj_func_grav(Most_Chrom(i,1:NParm),Velocity,Position,
Sensed_Torque, Torque, Time);
    Global_Chrom=Most_Chrom(i,1:NParm);
    else
        Fitness=obj_func_grav(Most_Chrom(i,1:NParm),Velocity,Position,
Sensed_Torque, Torque, Time);
        if Fitness<Global_Fit
            Global_Fit=Fitness;
            Global_Chrom=Most_Chrom(i,1:NParm);
        end
    end
end

%-----------------------Return to program----------------------
for i=1:NParm,
    Parameter_Outputs(i) = Global_Chrom(i);
```

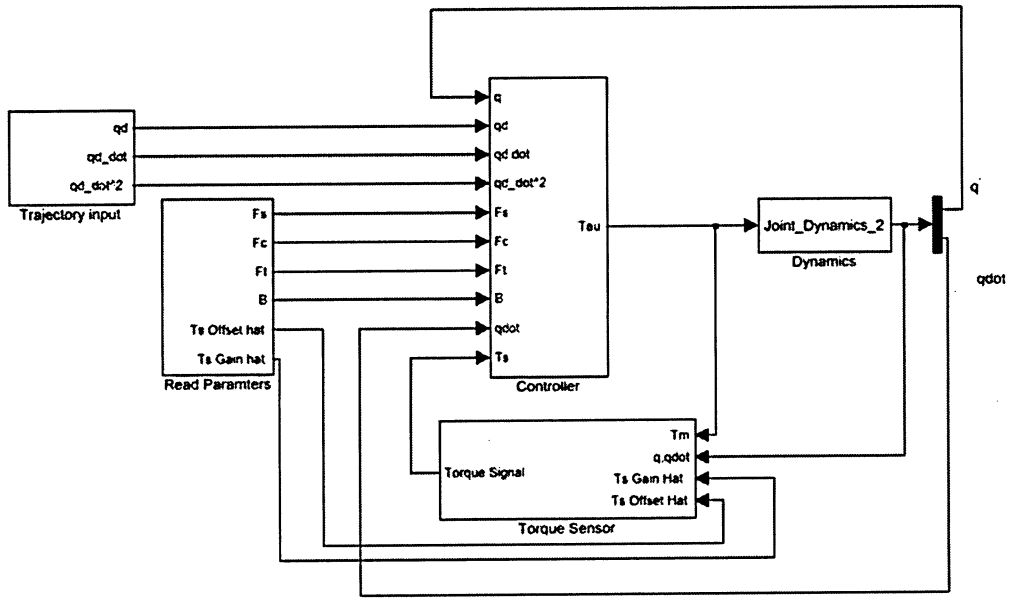
```
end
%-----------------------Write to File-----------------------------
save Friction_Values.txt Parameter_Outputs -ascii
%-----------------------Plot Items--------------------------------

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Selection algorithm.  Probability of selection increases with high
% fitness
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [j] = select (popsize, sumfitness, pop)
partsum=0;
j=0;
random=rand*sumfitness;
for j=1:popsize,
    partsum=partsum+pop(j).fitness;
    if (partsum>=random)
        break;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Random selection of 1 or 0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [bool] = flip(probability)
a=rand;
if (probability==1)
    bool=1;
elseif (probability<a)
    bool=0;
else
    bool=1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dynamic Adaptation Crossover method
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [x1, x2] = dynamic_adaptation_crossover (min, max, n, Scale,Parm)
u=rand;
if u<=0.5
    Bq=(2*u)^(1/(n+1));
else
    Bq=(1/(2*(1-u)))^(1/(n+1));
end

x1=abs(0.5*((1+Bq)*min+(1-Bq)*max));
x2=abs(0.5*((1-Bq)*min+(1+Bq)*max));

if x1>Scale(Parm)
    x1=Scale(Parm);
end
if x2>Scale(Parm)
    x2=Scale(Parm);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Real coded mutation algorithm
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [new_value] = mutate_real(chrom,Scale,gencount,maxgen,B,NParm)
parameter=1+floor(rand*NParm);
min=Scale(parameter,1);
max=Scale(parameter,2);
p=(1-gencount/maxgen)^B*rand;
new_value=chrom;
if (round(rand)==1)
    new_value(parameter)=abs((1-p)*chrom(parameter)+p*min);
else
    new_value(parameter)=abs((1-p)*chrom(parameter)+p*max);
end

if new_value(parameter)>max
    new_value(parameter)=max/2;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Integration of the system model
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Sensed_Value] = friction_model_grav(friction_parameters,qdot,q,
Torque, time)

Fc=    (friction_parameters(1));
Fs=    (friction_parameters(2));
Ft=    (friction_parameters(3));
B=     (friction_parameters(4));
Offset=(friction_parameters(5));
Gain= (friction_parameters(6));
%pshyical parameters
Im=0.0002472;
dt=time(2)-time(1);
Sensed_Value(1)=Torque(1);
%%%%%%Trapazoidal integration
for i=1:size(time,1),
    if i>1 & (time(i)-time(i-1)<0.02)
        Viscous=B*(q(i)-q(i-1));
        Coloumb=Fc*dt;
        Exponential=Fs*(exp(-abs(Ft*qdot(i)^2))+exp(-abs(Ft*qdot(i-
1)^2)))/2*dt;
        Friction=Viscous+(Coloumb+Exponential)*sign(qdot(i));
        Dynamic=(101*101*Im)*(qdot(i)-qdot(i-1));
        Input_Torque=(Torque(i)+Torque(i-1))/2*dt;
        Int_Gain=Gain*dt;
        Int_Offset=Offset*dt;
        Sensed_Value(i)=(Input_Torque-Dynamic-Friction);
        Sensed_Value(i)=(Sensed_Value(i)+Int_Offset)/Gain;
        Sensed_Value(i)=Sensed_Value(i)+Sensed_Value(i-1);
    else
        dt=time(i+1)-time(i);
        Sensed_Value(i)=Torque(1)*dt;
    End

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Objective function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

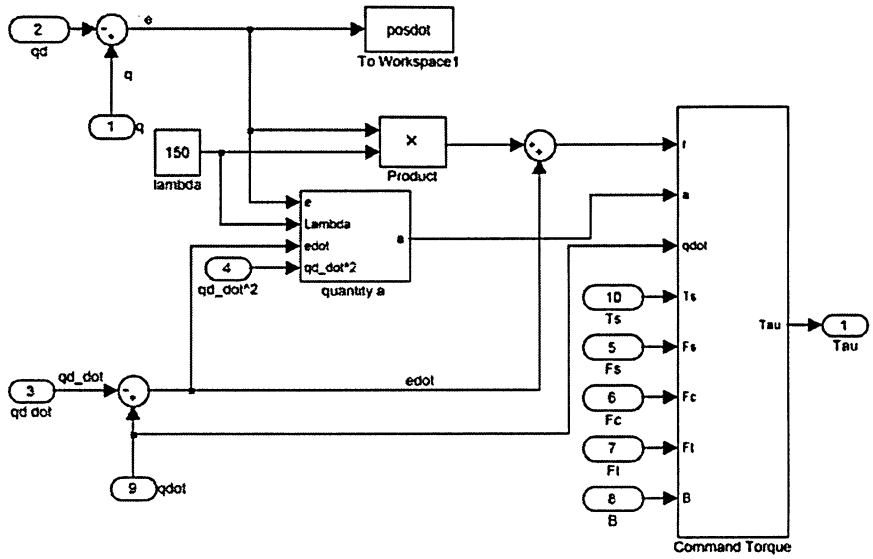
```
function [sum] =
obj_func_grav(parameter_values,velocity,position,Torque_sensor,Torque,Time)
x=0;
TS=TS_Integral(Torque_sensor,Time);
model_value=integral_friction_model_grav(parameter_values,velocity,position,T
orque,Time);
x=(TS-model_value).^2;
sum=0;
for i=1:size(x,2)
    sum=sum+x(i);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Non uniform crossover and mutation probability calculation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Pc, Pm] = nonuniform(Prob_cross, Prob_mut, Vmin, Vmax,gdm)
if (gdm<Vmin)
    Pc=Prob_cross(2);
    Pm=Prob_mut(1);
elseif (gdm>Vmax)
    Pc=Prob_cross(1);
    Pm=Prob_mut(2);
else
    Pc=-((Vmin-gdm)*(Prob_cross(2)-Prob_cross(1))/(Vmin-Vmax)-Prob_cross(2));
    Pm=(gdm-Vmin)*(Prob_mut(2)-Prob_mut(1))/(Vmax-Vmin)+Prob_mut(1);
end
```

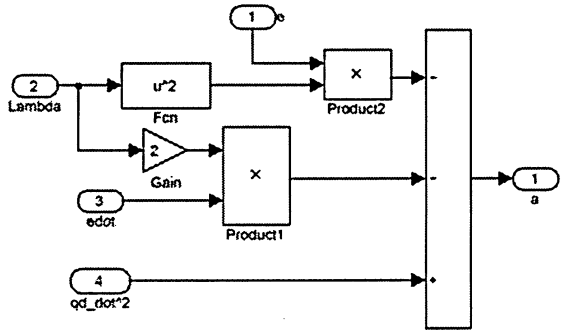
Simulink Block Diagrams for Simulation of 1 DOF Joint and Link with Torque Sensor and Friction

Overall System

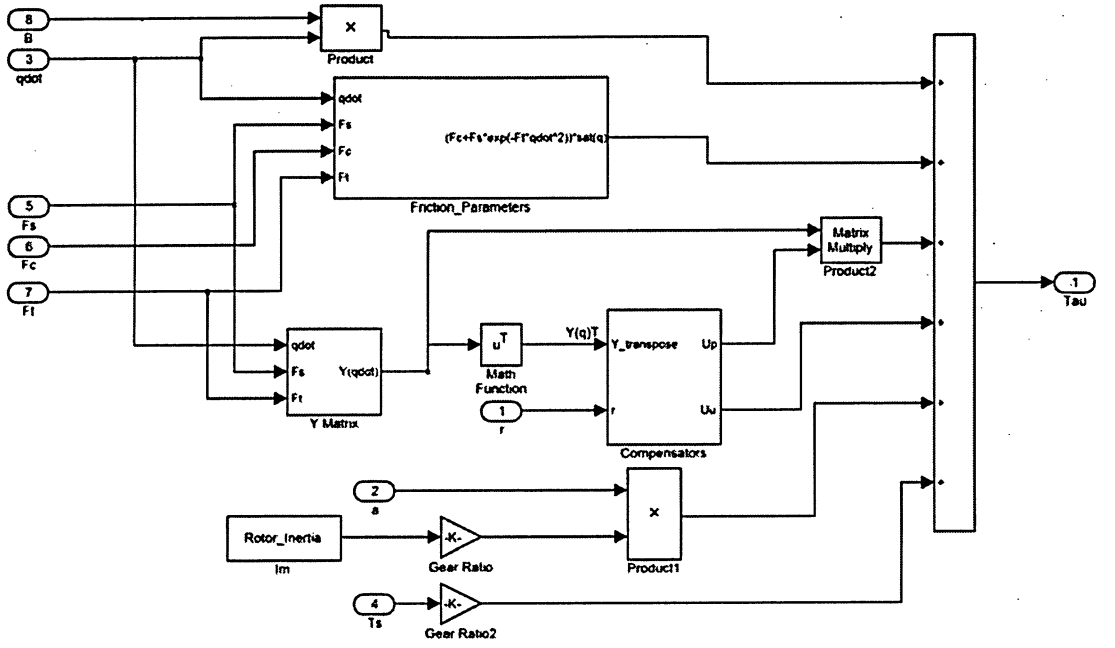


Controller

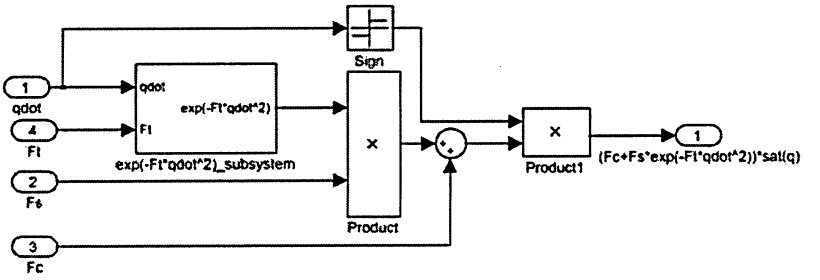




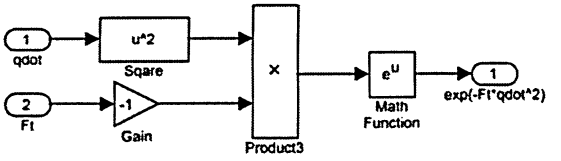
quantity a



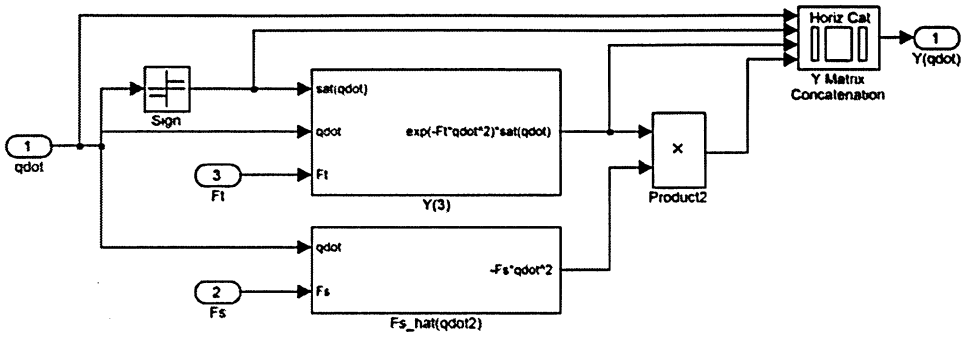
Command Torque



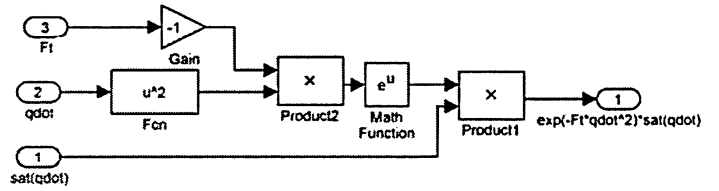
Friction Parameters



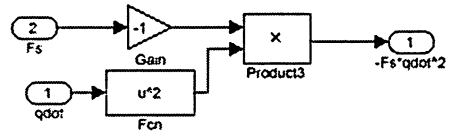
exp(-Ft*qdot^2)



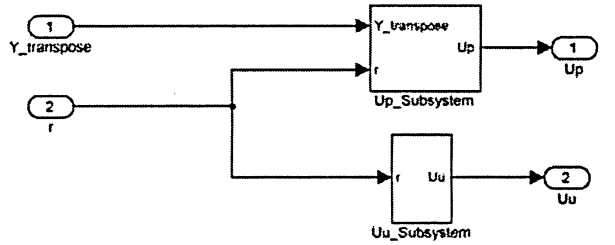
Y matrix



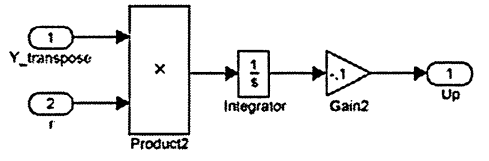
Y(3)



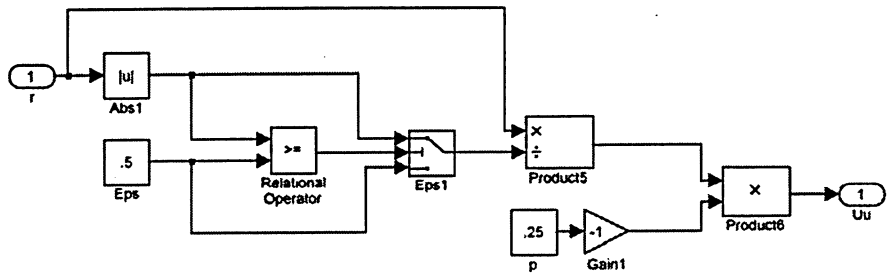
Fs_hat(qdot2)



Compensators

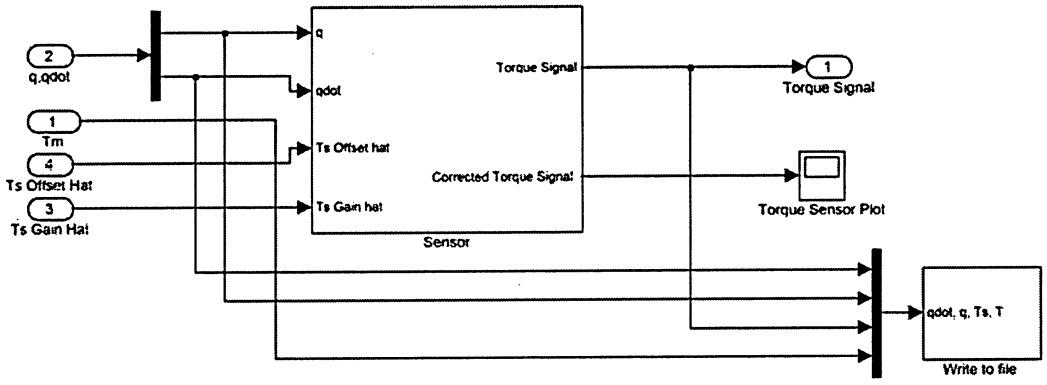


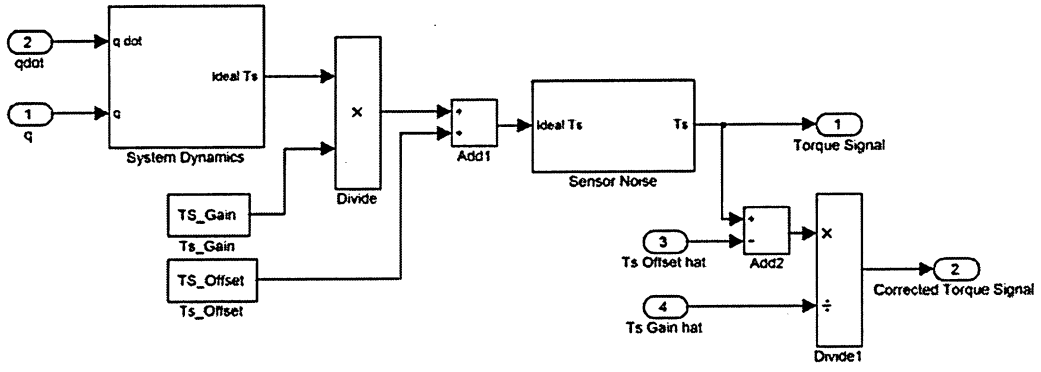
Up subsystem



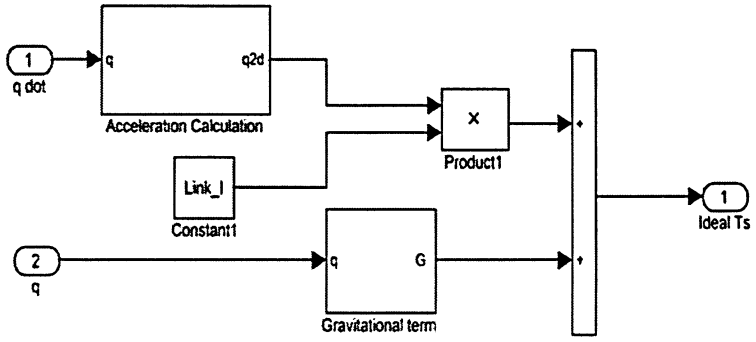
Uu subsystem

Torque Sensor

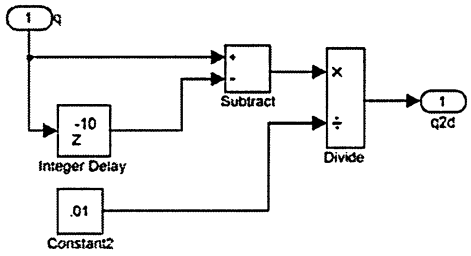




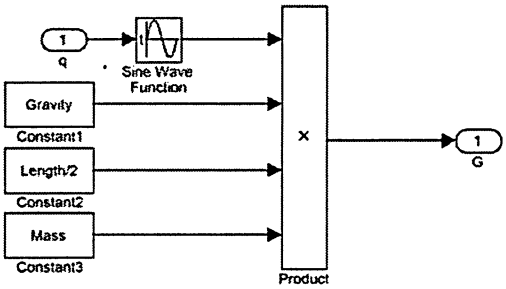
Sensor



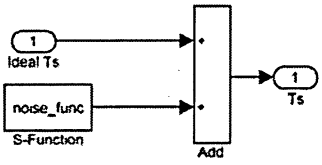
System Dynamics



Acceleration Calculation

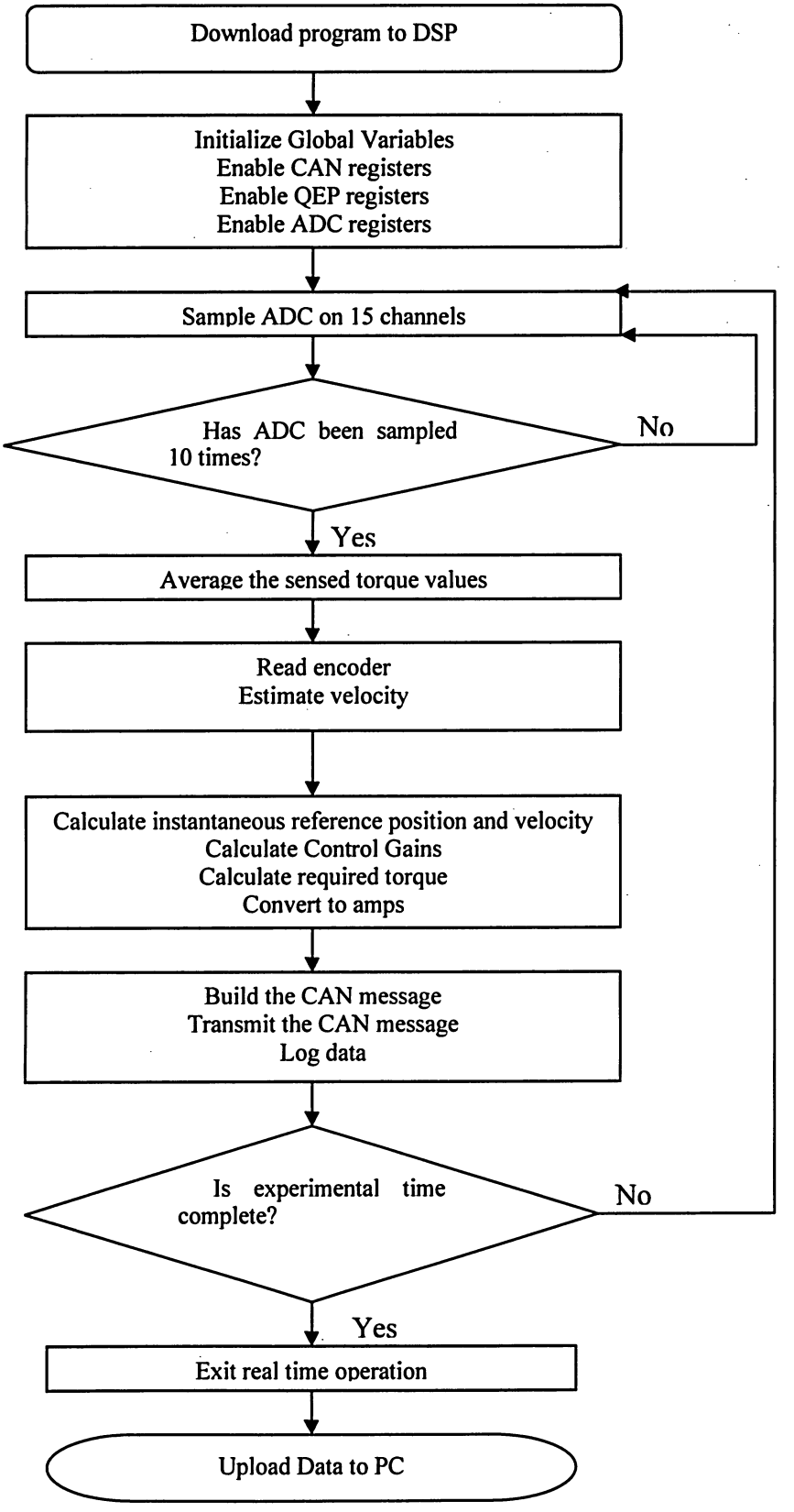


Gravitational Term



Sensor Noise

Real Time Control Algorithm



DSP Program C Code

```
/*********************************************************************
 File name   :    can.c
 Project     :    MRR Joint Prototype
 Originator  :    M. Adamson, S. Abdul
 Target Sys  :    MSK2812 board
 Description :    C file for CAN demo modified to include ADC, QEP.

 This application controls a single joint via current commands sent to an
elmo driver.

 What to do? Set trajectory settings, compile, download and run the
application. Upload trace variables to see performance.
 -------------------------------------------------------------------
 Status            :    OK
 Last Update : Nov 6, 2006
---------------------------------------------------------------------
 Include required header files:
    LF281x_Regs.h : bit definitions of peripheral registers.
    RegDef.h      : definition of Global Peripheral Variables.
    can.h           :   specific for the given example.
      qep.h : encoder
      adc.h : analog to digital conversion
      math.h : math functions
---------------------------------------------------------------------*/
#include "..\LF2812_Regs.h"
#include "..\RegDef.h"
#include "math.h"
#include "can.h"
#include "qep.h"
#include "adc.h"

// Variables must be declared globally in order to trace them or log them
i.e. download them to PC after program has run.
float pos_error=0;
long Position=0;
int Transmit_Count=0;
int error_count=0;
float Prev_Pos=0;
float Velocity=0, Vel_rad=0;
float ref_vel=0;
float ref_pos=0;
float Prev_ref_pos=0;
int strExMsg[4];
float vel_error=0;
float ref_acc=0;
float prev_vel=0;
float Nom_Tau=0;
float sumY1=0, sumY2=0, sumY3=0, sumY4=0;
float r=0;
float Start_Pos;
int Step_Est=0;
float Vel_Avg=0;
float Torque_Sensor;
float Nom_acc;
float Gain, Offset;
```

```
float a_1;
float t_1;
//float Torque_filt; // Filtered Torque Value
float Up=0, Uu=0;
float Torque_Corr;// position dependent correction on sensor reading
float Error_Torque;
float time_s;
float Pos_Rad;
void main(void)
{
     struct ECAN_REGS ECanaShadow;
/******** Global variable and function initilization  ***********/
      init_logger();
      Transmit_Count=0;
      error_count=0;
      pos_error=0;
      DemoQEP.eva_evb = 0;
      DemoQEP.position = 0x0000;
      DemoCAN.stop = 0;
      DemoADC.count_adc=0;
      DemoADC.count_saved=0;
      DemoADC.ADC_Avg=0;

//===========================================================
/*************Initilize CAN variables***************************/
//===========================================================
      DemoCAN.bit_tim_param = 0x000B007F; // 500Kbs
      DemoCAN.op_mode = 0; //Network Mode
      DemoCAN.rx_format_id = 0x0000;
      DemoCAN.rx_std_id =0X04FF;
      DemoCAN.rx_xtd_id = 0x1FFFFFFF;
      DemoCAN.tx_format_id = 0x0;
      DemoCAN.tx_std_id = 0x037F;
      DemoCAN.tx_xtd_id = 0x1FFFFFFF;
      DemoCAN.std_accept_mask = 0x0; do not mask out any bits
      DemoCAN.xtd_accept_mask = 0x1FFFFFFF;
      DemoCAN.tx_data_len = 0x8;
      DemoCAN.tx_data_val[0] = 0x1234;
      DemoCAN.tx_data_val[1] = 0x5678;
      DemoCAN.tx_data_val[2] = 0x9ABC;
      DemoCAN.tx_data_val[3] = 0xDEF0;
      DemoCAN.data_len = 0x0;
      DemoCAN.tx_msg_rq = 0x1; //0x0 = do not send
      DemoCAN.rx_msg_rq = 0x0;
      DemoCAN.pos_count = 0;
      DemoCAN.bitvalue=2;

/************** Configure CAN bits *****************/
// Enable CAN clock
      EALLOW;
      SysCtrlRegs.PCLKCR.bit.ECANENCLK=1;
      EDIS;

// Configure CAN pins using GPIO regs here
      EALLOW;
      GpioMuxRegs.GPFMUX.bit.CANTXA_GPIOF6 = 1;
      GpioMuxRegs.GPFMUX.bit.CANRXA_GPIOF7 = 1;
```

```
    EDIS;

// Configure the eCAN RX and TX pins for eCAN transmissions
    EALLOW;
    ECanaShadow.CANTIOC.all = ECanaRegs.CANTIOC.all;
    ECanaShadow.CANTIOC.bit.TXFUNC = 1;
    ECanaRegs.CANTIOC.all = ECanaShadow.CANTIOC.all;

    ECanaShadow.CANRIOC.all = ECanaRegs.CANRIOC.all;
    ECanaShadow.CANRIOC.bit.RXFUNC = 1;
    ECanaRegs.CANRIOC.all = ECanaShadow.CANRIOC.all;
    EDIS;

    EALLOW;
      ECanaShadow.CANMC.all = ECanaRegs.CANMC.all;
// select the operation mode
      if (DemoCAN.op_mode == 0)
            ECanaShadow.CANMC.bit.STM = 0;      // set the CAN Network Mode
(op_mode = 0)
      else
            ECanaShadow.CANMC.bit.STM = 1;      // set the Self Test Mode
(op_mode = 1)
      ECanaRegs.CANMC.all = ECanaShadow.CANMC.all;

      ECanaShadow.CANMC.all = ECanaRegs.CANMC.all;
      ECanaShadow.CANMC.bit.DBO = 1;      // configure data byte order as
0,1,2,..,7
      ECanaShadow.CANMC.bit.ABO = 1;      // set Auto Bus On bit
      ECanaShadow.CANMC.bit.CCR = 1;      // CPU requests write access to
Configuration Registers CANBTC
      ECanaRegs.CANMC.all = ECanaShadow.CANMC.all;
      EDIS;
// Wait until the CPU has been granted permission to change the configuration
registers
    do
    {
            DemoCAN.ret_val_mon = (*callmon2812)();  // call monitor
            ECanaShadow.CANES.all = ECanaRegs.CANES.all;
      } while(ECanaShadow.CANES.bit.CCE != 1 );  // Wait for CCE bit to be
set..
// Configure the eCAN timing paramenters and baud rate prescaler
      EALLOW;
      ECanaShadow.CANBTC.all = DemoCAN.bit_tim_param;
      ECanaRegs.CANBTC.all = ECanaShadow.CANBTC.all;
// CPU requests normal operation
      ECanaShadow.CANMC.all = ECanaRegs.CANMC.all;
      ECanaShadow.CANMC.bit.CCR = 0 ;           // Set CCR = 0 to disable the
write access to Configuration Registers
      ECanaRegs.CANMC.all = ECanaShadow.CANMC.all;
      EDIS;
// Wait until the CPU no longer has permission to change the
// configuration registers
    do
    {
            DemoCAN.ret_val_mon = (*callmon2812)();     // call monitor
            ECanaShadow.CANES.all = ECanaRegs.CANES.all;
```

```
    } while(ECanaShadow.CANES.bit.CCE != 0 ); // Wait for CCE bit to be
cleared..

// clear transmission acknowledge bit for mailbox 5
      ECanaRegs.CANTA.bit.TA5 = 1;
// clear receive message pending bit
      ECanaRegs.CANRMP.bit.RMP0 = 0;

      ConfigRxMailBox();
/*=====================================================================*/
/****************Initilize QEP Settings*****************************/
/*=====================================================================*/
// configure shared pins as capture input pins
      EALLOW;
      GpioMuxRegs.GPAMUX.all = EVA_QEPENMSK; // EVA CAP/QEP  pins
      EDIS;
// Enable EVA clock
      SysCtrlRegs.PCLKCR.bit.EVAENCLK=1;
// Setup Timer 2 Registers (EV A)
      EvaRegs.GPTCONA.bit.T2TOADC = 0;     // configure GPTCONA not to start
ADC on GPT2 Event
      EvaRegs.T2PR = T2PR_INI;       // initialize GPT2 timer period
      EvaRegs.T2CNT = TxCNT_INI_QEP;       // reset GPT2 counter register
// set GPT2 configuration register
      EvaRegs.T2CON.all = TxCON_INI_QEP;  // T2CON prepared to use for QEP,
x/1
// start timer 2
      EvaRegs.T2CON.bit.TENABLE = 1;             // Start GPT2 counter to
count QEP circuit pulses
// configure Capture Control Register CAPCONA
      EvaRegs.CAPCON.bit.CAPQEPN = 3;            // enable QEP in CAPCONA
register
/*=====================================================================*/
/-****************Initialize ADC Settings***********************/
/*=====================================================================*/
/*To transmit every 2ms, must account for delay of transmition*****/
/**147 microseconds 10 times=1.47ms + transmit time = approx. 2ms per current
message*/
      DemoADC.timer_period = 0X56B8;  //147 micro seconds
      DemoADC.n_samples = 8; // transmit time = (n_samples X
timer_period)+transmit time
//******configure ADC to read Channel 6 on all samples (channel 6 has torque
sensor input)
      DemoADC.ch_sel1 = 0x6666;
      DemoADC.ch_sel2 = 0x6666;
      DemoADC.ch_sel3 = 0x6666;
      DemoADC.ch_sel4 = 0x6666;
      DemoADC.max_ch = 15;
  // Enable ADC clock
      EALLOW;
      SysCtrlRegs.PCLKCR.bit.ADCENCLK=1;
      EDIS;
   // Set HSPCLK to SYSCLKOUT = 150MHz
      EALLOW;
      SysCtrlRegs.HISPCP.all = 0;    // HSPCLK = SYSCLKOUT / 1 = 150MHz
      EDIS;
      AdcRegs.ADCTRL3.bit.ADCCLKPS = 3;    // ADCCLK = HSPCLK/6
```

```
      AdcRegs.ADCTRL1.bit.CPS = 0;         // ADCCLK = HSPCLK/6

// Power up the reference and bandgap circuits
      AdcRegs.ADCTRL3.bit.ADCBGRFDN = 0x3;      // Power up bandgap/reference
circuitry
      DELAY_US(DELAY5000);       // Delay before powering up rest of ADC
      AdcRegs.ADCTRL3.bit.ADCPWDN = 1;          // Power up rest of ADC
      DELAY_US(DELAY20);              // Delay after powering up ADC
  // Assign ADC Interrupt Service Routine
      EALLOW;     // This is needed to write to EALLOW protected registers
      PieVectTable.ADCINT = &adc_isr;
      EDIS; // This is needed to disable write to EALLOW protected registers
// Enable ADCINT in PIE
    PieCtrlRegs.PIEIER1.bit.INTx6 = 1;
// Enable CPU Interrupt 1
    IER |= M_INT1;                                                    //
Enable Global INT1
// Enable global Interrupts and higher priority real-time debug events:
      EINT;        // Enable Global interrupt INTM
      ERTM;        // Enable Global realtime interrupt DBGM
// set Maximum Conversion Channels Register
      AdcRegs.MAXCONV.all =DemoADC.max_ch;      // Setup the number of conv's
on SEQ1
// Select Cascaded mode
      AdcRegs.ADCTRL1.bit.SEQ_CASC = 1;         // Cascaded mode
// Initialize ADC Input Channel Select Sequencing Control Register
      AdcRegs.CHSELSEQ1.all = DemoADC.ch_sel1; // Setup the 1st to 4th SEQ1
conv.
      AdcRegs.CHSELSEQ2.all = DemoADC.ch_sel2; // Setup the 5th to 8th SEQ1
conv.
      AdcRegs.CHSELSEQ3.all = DemoADC.ch_sel3; // Setup the 9th to 12th SEQ1
conv.
      AdcRegs.CHSELSEQ4.all = DemoADC.ch_sel4; // Setup the 13th to 16th SEQ1
conv.
      AdcRegs.ADCTRL2.bit.EVA_SOC_SEQ1 = 1;  // Enable EVASOC to start SEQ1
      AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1;  // Enable SEQ1 interrupt (every
EOS)
// Configure EVA
// Assumes EVA Clock is already enabled
      EvaRegs.T1CMPR = 0x0080;               // Setup T1 compare value
      EvaRegs.T1PR = DemoADC.timer_period; // Setup period register
      EvaRegs.GPTCONA.bit.T1TOADC = 1;       // Enable EVASOC in EVA
      EvaRegs.T1CON.all = 0x1042; // Enable timer 1 compare (upcount mode)
/*=================================================================*/
//****Main Rouitne.  Executes interrupt until Experiment is over****/
/*=================================================================*/
      while (!DemoCAN.stop)
      {
            DemoCAN.ret_val_mon = (*callmon2812)();
      }
      DINT;
      EALLOW;
      SysCtrlRegs.SCSR.bit.WDOVERRIDE =1; // Set WDOVERRIDE bit to allow the
Watchdog enable
      SysCtrlRegs.WDCR = 0x000F; // Enable Watchdog and write incorrect WD
Check Bits
}
```

```
//*****************************************************************
void SendMessage(void)
{
      ECanaRegs.CANTRS.bit.TRS5 = 1;             // enable transmission of the
mailbox 5
}
//*****************************************************************
void ReadMessage(void) //unused in this version.  only needed for reading CAN
messages on DSP
{ /*
      if(DemoCAN.rx_format_id == 0) // test reception mailbox format:
0=standard format, 1=extended format
      // read standard identifier
            DemoCAN.rx_std_id = ((int)ECanaMboxes.MBOX0.MID.bit.MSGID_H) >>
2;
      else
            // read extended identifier
            DemoCAN.rx_xtd_id = ECanaMboxes.MBOX0.MID.all & 0x1FFFFFFF;
      // load data length
      DemoCAN.rx_data_len = ECanaMboxes.MBOX0.MCF.bit.DLC;
      if(DemoCAN.rx_data_len != 0)  //  test if data was transmitted
      {
            // convert bytes in words
            DemoCAN.data_len = ( (DemoCAN.rx_data_len + 1) >> 1 ) - 1;
            // read received data
            switch (DemoCAN.data_len)
            {
                  case 3:
                        DemoCAN.rx_data_val[3] =
ECanaMboxes.MBOX0.MDRH.bit.HI_WORD;
                  case 2:
                        DemoCAN.rx_data_val[2] =
ECanaMboxes.MBOX0.MDRH.bit.LOW_WORD;
                  case 1:
                        DemoCAN.rx_data_val[1] =
ECanaMboxes.MBOX0.MDRL.bit.HI_WORD;
                  case 0:
                        DemoCAN.rx_data_val[0] =
ECanaMboxes.MBOX0.MDRL.bit.LOW_WORD;
                  default: ;
            }
      DemoCAN.position=DemoCAN.rx_data_val[1]//=/[DemoCAN.pos_count]=DemoCAN.
rx_data_val[1];
      DemoCAN.position//[DemoCAN.pos_count]=DemoCAN.position[DemoCAN.pos_coun
t]<<16;
      DemoCAN.position[DemoCAN.pos_count]+=DemoCAN.rx_data_val[0];
            DemoCAN.timestamp[5]=ECanaMots.MOTS0;
            DemoCAN.pos_count+=1;
      DemoCAN.position=DemoCAN.rx_data_val[1];
//          DemoCAN.position=DemoCAN.position<<16;
//          DemoCAN.position+=DemoCAN.rx_data_val[0];
//          DemoCAN.timestamp=ECanaMots.MOTS0;
            //Position=DemoCAN.position;
      //    logger();

      }*/
}
```

```
//*************************************************************
void ConfigRxMailBox(void)
{
// configure reception mailbox0
      ECanaRegs.CANME.bit.ME0 = 0; // disable mailbox 0

      if(DemoCAN.rx_format_id == 0)
      // reception mailbox format: 0=standard format
      {
            // set local acceptance mask
            ECanaLam.LAM0.bit.LAM_H = DemoCAN.std_accept_mask;
            // set standard identifier
            ECanaMboxes.MBOX0.MID.bit.MSGID_H = DemoCAN.rx_std_id << 2;
            ECanaMboxes.MBOX0.MID.bit.IDE = 0;  // configure to Standard
Identifier
      }
      else
      // reception mailbox format: 1=extended format
      {
            // set local acceptance mask
            ECanaLam.LAM0.bit.LAM_L = (int) (DemoCAN.xtd_accept_mask &
0xFFFF);
            ECanaLam.LAM0.bit.LAM_H = (int) (DemoCAN.xtd_accept_mask >> 16);
            // set extended identifier
            ECanaMboxes.MBOX0.MID.bit.MSGID_L = (int)(DemoCAN.rx_xtd_id &
0xFFFF);
            ECanaMboxes.MBOX0.MID.bit.MSGID_H = (int)(DemoCAN.rx_xtd_id >>
16);
            ECanaMboxes.MBOX0.MID.bit.IDE = 1;  // configure to Extended
Identifier
      }

      ECanaMboxes.MBOX0.MID.bit.AME = 1;  // enable the usage of the
acceptance mask

      ECanaRegs.CANMD.bit.MD0 = 1;        // Config as receive mailbox
      ECanaRegs.CANME.bit.ME0 = 1;    // enable mailbox 0
}
//*************************************************************
void ConfigTxMailBox(void)
{
// configure transmission mailbox5
      ECanaRegs.CANME.bit.ME5 = 0; // disable mailbox 5
      if(DemoCAN.tx_format_id == 0)
      // transmission mailbox format: 0=standard format
      {
            // set standard identifier
            ECanaMboxes.MBOX5.MID.bit.MSGID_H = DemoCAN.tx_std_id << 2;
            ECanaMboxes.MBOX5.MID.bit.IDE = 0;  // configure to Standard
Identifier
      }
      else
      // transmission mailbox format: 1=extended format
      {
            // set extended identifier
            ECanaMboxes.MBOX5.MID.bit.MSGID_L = (int)DemoCAN.tx_xtd_id;
```

```
            ECanaMboxes.MBOX5.MID.bit.MSGID_H = (int)(DemoCAN.tx_xtd_id >>
16);
            ECanaMboxes.MBOX5.MID.bit.IDE = 1;  // configure to Extended
Identifier
      }
      ECanaRegs.CANME.bit.ME5 = 1;          // enable mailbox 5
      ECanaMboxes.MBOX5.MCF.bit.RTR = 0; // no remote frame requested
      if(DemoCAN.tx_data_len != 0)  // test if data  be transmitted
      {
            ECanaMboxes.MBOX5.MCF.bit.DLC = DemoCAN.tx_data_len;

            // convert bytes in words
            DemoCAN.data_len = ( (DemoCAN.tx_data_len + 1) >> 1 ) - 1;

            // put data for transmission
            switch (DemoCAN.data_len)
            {
                  case 3:
                        ECanaMboxes.MBOX5.MDRH.bit.HI_WORD =
DemoCAN.tx_data_val[3];
                  case 2:
                        ECanaMboxes.MBOX5.MDRH.bit.LOW_WORD =
DemoCAN.tx_data_val[2];
                  case 1:
                        ECanaMboxes.MBOX5.MDRL.bit.HI_WORD =
DemoCAN.tx_data_val[1];
                  case 0:
                        ECanaMboxes.MBOX5.MDRL.bit.LOW_WORD =
DemoCAN.tx_data_val[0];
                  default: ;
            }
      }
}

/*=====================================================================*/
/***Convert Torque to floating point and transmit over CAN bus*******/
/*=====================================================================*/
void TCommand(float TC)
{
   Uint32 Bin=0;
      float mantissa;
      long temp[4];
      float Tc_Dec;
      int bin_flag;
      int i;
      long bin_power=0;
      int sign_flag=0;
      bin_flag=1;
      //*******Convert floating point number to IEEE floating point format
(as required by Elmo)*****/
      if (TC<0)
      {
            sign_flag=1;
            TC=TC*-1;
      }
      Tc_Dec=TC-((int)(TC));
      if (TC==0) Bin=0;
```

```
            else
            {
                  if (TC>=2)
                   for (i=1;i<4;i++)
                   {
                   if ( ( (int)(TC)/(int)(pow(2.,i)) ) >=1)
                               bin_power+=1;

                   }
                  else if (TC<1)
                   for (i=0;i<8;i++)
                         if (Tc_Dec-pow(2.,-i)>=0) bin_flag=0;
                         else if (bin_flag) bin_power-=1;
                  mantissa=(TC)/pow(2.,bin_power);

                  if (mantissa>=1) mantissa-=1;

                  for (i=1;i<5;i++)
                  {
                        Bin=Bin<<1;
                        if (mantissa-pow(2.,-i)>=0)
                        {
                              mantissa=mantissa-pow(2.0,-i);
                              Bin+=1;
                        }
                  }
                  Bin=Bin<<19;
                  Bin=Bin+((127+bin_power)<<23);
                  if (sign_flag) Bin=Bin+2147483648;
            }
            temp[0]=Bin&0x000000FF;
            temp[1]=(Bin&0x0000FF00)>>8;
            temp[2]=(Bin&0x00FF0000)>>16;
            temp[3]=(Bin&0xFF000000)>>24;

      /*********************Build the CAN message*********************/
      DemoCAN.tx_data_val[0] = 0x4354;  // 'T'=0x54, 'C'=0x43
      DemoCAN.tx_data_val[1] = 0x8000;  // tells Elmo driver to expect
floating point number in IEEE format
      DemoCAN.tx_data_val[2] = ((temp[1]<<8)+temp[0]);
      DemoCAN.tx_data_val[3] = ((temp[3]<<8)+temp[2]);
      ECanaRegs.CANTA.bit.TA5 = 1;// clear transmission acknowledge bit
      ConfigTxMailBox();      // configure MailBox 5
      SendMessage();    //transmit message
      while (ECanaRegs.CANTA.bit.TA5==0) {}

      return;
}
//===========================================================
//    Timer 2 period interrupt service routine
//===========================================================
interrupt void adc_isr(void)
{
      float TC,K_P;
      float lambda,  a=0, abs_r=0;
      float Fs_hat, Ft_hat, Fc_hat, B_hat;// Offset_hat, Gain_hat;
      float Im;
```

```
      float Ku, eps;
      float count_rad=3.14159/4000;
      int sign_vel=0;
      int Pos_Diff;
      float Y1=0, Y2=0, Y3=0, Y4=0;
      //    float nd[2],dd[2];    // filter coefficients
      float Amplitude,Frequency;
      int MAXCOUNT;
      int Data_Log_Interval;
      float Start_Link;       // starting position of link in degrees
      float Cycles;

      /*******  Trajectory Settings.  Experiment Duration ************/
      MAXCOUNT=6000;
      Start_Link=90;
      Start_Link=Start_Link/180.*3.14159+0.6454;
      Amplitude=20; //link in degrees
      Amplitude=Amplitude*101*200/9; //convert link in degrees to motor in
counts
      Amplitude=Amplitude/1.3; //repeating sine
      Cycles=2;
      Frequency=2*Cycles*3.142/MAXCOUNT;  // sampling time of 2ms
      Data_Log_Interval=MAXCOUNT/1500;

      /*****************  Control Gain Settings ********************/
      K_P=.001
      Ku=0.05
      eps=0.9;
      lambda=80;
      /******************** Parameter Settings ********************/
      /*                 in terms of motor units                    */
      Fc_hat=.18;
      Ft_hat=.040;
      Fs_hat=.1;
      B_hat=13/101/101;
      Im=0.0002472;
      Gain=1/0.3;
      Offset=4;
      /**********  Low pass filter for Torque Sensor Signal  ******/
      /*   For experiments on filtered torque signal only */
      /*    nd[0]=.09091; //100Hz
            nd[1]=.09091;
            dd[1]=-.8182; *
            nd[0]=.04762; //50Hz
            nd[1]=.04762;
             dd[1]=-.9048; */

      if (DemoADC.count_adc < DemoADC.n_samples)    // if count_adc <
n_samples, new acquisition
      {
            // read and store A/D conversion results
            switch (DemoADC.max_ch)
                  {
                        case 15:
                        {
                              DemoADC.count_saved++;
                              DemoADC.ADC_Avg+=(AdcRegs.RESULT15>>4);
```

```
}
case 14:
{
      DemoADC.count_saved++;
      DemoADC.ADC_Avg+=(AdcRegs.RESULT14>>4);
}
case 13:
{
      DemoADC.count_saved++;
      DemoADC.ADC_Avg+=(AdcRegs.RESULT13>>4);
}
case 12:
{
      DemoADC.count_saved++;
      DemoADC.ADC_Avg+=(AdcRegs.RESULT12>>4);
}
case 11:
{
      DemoADC.count_saved++;
      DemoADC.ADC_Avg+=(AdcRegs.RESULT11>>4);
}
case 10:
{
      DemoADC.count_saved++;
      DemoADC.ADC_Avg+=(AdcRegs.RESULT10>>4);
}
case 9:
{
      DemoADC.count_saved++;
      DemoADC.ADC_Avg+=(AdcRegs.RESULT9>>4);
}
case 8:
{
      DemoADC.count_saved++;
      DemoADC.ADC_Avg+=(AdcRegs.RESULT8>>4);
}
case 7:
{
      DemoADC.count_saved++;
      DemoADC.ADC_Avg+=(AdcRegs.RESULT7>>4);
}
case 6:
{
      DemoADC.count_saved++;
      DemoADC.ADC_Avg+=(AdcRegs.RESULT6>>4);
}
case 5:
{
      DemoADC.count_saved++;
      DemoADC.ADC_Avg+=(AdcRegs.RESULT5>>4);
}
case 4:
{
      DemoADC.count_saved++;
      DemoADC.ADC_Avg+=(AdcRegs.RESULT4>>4);
}
case 3:
```

```
                        {
                              DemoADC.count_saved++;
                              DemoADC.ADC_Avg+=(AdcRegs.RESULT3>>4);
                        }
                        case 2:
                        {
                              DemoADC.count_saved++;
                              DemoADC.ADC_Avg+=(AdcRegs.RESULT2>>4);

                        }
                        case 1:
                        {
                              DemoADC.count_saved++;
                              DemoADC.ADC_Avg+=(AdcRegs.RESULT1>>4);
                        }
                        case 0:
                        {
                              DemoADC.count_saved++;
                              DemoADC.ADC_Avg+=(AdcRegs.RESULT0>>4);

                        }
                        default: ;
                  }

            DemoADC.count_adc++;    // increment counter of A/D made
conversions
      }
      else
      {     /*=============================================================*/
      /********************** Control Law ***************************/
      /*=============================================================*/
      /********Calculate the reference trajectories******************/
      /* Several reference trajectories are available.  Uncomment the desired
      trajectory for the experiment and comment the unused ones.  Using a
      reference velocity will improve the accuracy if an analytical solution
      is readily available.***********************/
     ref_pos=-Amplitude*(sin(Frequency*Transmit_Count)-
0.5*sin(2*Frequency*Transmit_Count));
      //    ref_vel=-500*Amplitude*Frequency*(cos(Frequency*Transmit_Count)-
cos(2*Frequency*Transmit_Count))*count_rad;
      //Sigmoid
      //    ref_pos=Amplitude/2*(1-cos(3.14159*(2*Transmit_Count-
1)/(2*(MAXCOUNT-1))));
      //    ref_vel=count_rad*Amplitude/3.142*sin(3.14159*(2*Transmit_Count-
1)/(2*(MAXCOUNT-1)));
      //Step
      /*    if (Transmit_Count<200)
                  ref_pos=0;
            else
                  ref_pos=1000;
            ref_vel=0;  */

      //*********************************************************//
      //**Store varibles from previous step for reference velocity and
acceleartion***///
            if (Transmit_Count>1) //used to generate the reference
acceleration
```

```
                prev_vel=ref_vel;
            else
                prev_vel=0;
            time_s=(float)(Transmit_Count)/1000.*2;
      //Keep track of previous reference position for reference velocity if
trajectory is not known
      if (Transmit_Count>0)
            {
                ref_vel=((ref_pos-Prev_ref_pos)/.002)*count_rad; //Use if
ref velocity is unknown
                Prev_Pos=Position;
            }
            else
                ref_vel=0;
            Prev_ref_pos = ref_pos;
            //Trapazoid trajectory
      /*    y1=20000; //in motor counts
            y2=40000;
            y3=60000;
            time1=500;
            time2=750;
            time3=1250;
            if (Transmit_Count<time1)
            {
            ref_pos=y1*Transmit_Count*Transmit_Count/(time1*time1);
            ref_vel=2*y1*Transmit_Count/time1*count_rad;
            //    ref_acc=2*y1/time1/time1;
            }
            if ((Transmit_Count>=time1)&&(Transmit_Count<=time2))
            {
            ref_pos=(Transmit_Count-time1)/(time2-time1)*(y2-y1)+y1;
            ref_vel=2*(y2-y1)*count_rad;
      //          ref_acc=0;
            }
            if (Transmit_Count>time2)
            {
                ref_pos=-((y3-y2)*(time3-Transmit_Count)/(time3-
time2)*(time3-Transmit_Count)/(time3-time2))+y3;
                ref_vel=2*(y3-y2)*(time3-Transmit_Count)/(time3-
time2)*count_rad;
      //          ref_acc=-2*(y3-y2)/(time3-time2)/(time3-time2);
            }*/
      /*************************************************************/
      /********* Obatain Position from encoder in counts*************/
            if (Transmit_Count<1)
                EvaRegs.T2CNT=0;
            DemoQEP.position += (int)(EvaRegs.T2CNT);
            EvaRegs.T2CNT=0;
            Position=DemoQEP.position;
      /*********** Torque sensor calculation*************************/
            DemoADC.ADC_Avg=DemoADC.ADC_Avg/DemoADC.count_saved;
            DemoADC.Volts=DemoADC.ADC_Avg*3/4095;
      Torque_Corr=0.2+0.09*cos((float)(Position)*2*3.142/(8000.0*101.0)+Start
_Link);//position based torque correction
            Torque_Sensor=((DemoADC.Volts-1.5)/1.5); //217.34
            Torque_Sensor=Torque_Sensor-Torque_Corr;
```

```
          Torque_Sensor=(217.34*Torque_Sensor-Offset)/Gain; //adjust for
offsets and gains (217.34 is the amplifier gain and represents max torque)
      /*****Initialze varibles in the first step of the experiment***/
            if (Transmit_Count<1)
                  {
                        Velocity=0;
                        Step_Est=0;
                        a_1=Torque_Sensor;
                        t_1=a_1;
                  }
      /* Find the Reference Acceleration for the nominal torque value*/
            if (Transmit_Count>1)
            {
                  ref_acc=(ref_vel-prev_vel)/.002;
            }
            else ref_acc=0;

      /************** Velocity Estimation routine***************/
            if (Step_Est==0)
             Start_Pos=Position;
            Pos_Diff=Position-Start_Pos;
            if ((Step_Est==2) || (abs(Pos_Diff)>3))
                  {
                        Velocity=Pos_Diff/(Step_Est*.002)*count_rad;
                        Step_Est=0;
                  }
            else Step_Est++;
      /**Calculate velocity and position errors as well as mixed tracking
errors**/
            vel_error=(-ref_vel+Velocity);
            pos_error=(-(ref_pos)+Position)*count_rad;
            r=vel_error+lambda*pos_error;
            a=ref_acc-2*lambda*vel_error-(lambda*lambda)*pos_error;
            Pos_Rad=(float)Position*count_rad;
            //Find the sign of the velocity signal
            if (Velocity ==0)
                  sign_vel=0;
            else if     (Velocity<0)
                  sign_vel=-1;
            else
                  sign_vel=1;

      /***************Calculate the nominal torque****************/
            Nom_Tau=Im*a+B_hat*Velocity+(Fc_hat+Fs_hat*exp(-
Ft_hat*Velocity*Velocity))*sign_vel;
            Nom_Tau=Nom_Tau/0.198; //convert from Nm to Amperes
      /***************Calculate Parametric Control Value*************/
            Y1=Velocity;
            Y2=sign_vel;
            Y3=exp(-Ft_hat*Velocity*Velocity)*sign_vel;
            Y4=(-Fs_hat*Velocity*Velocity)*exp(-
Ft_hat*Velocity*Velocity)*sign_vel;
            if (Transmit_Count>1)
            {
                  sumY1+=r*Y1*.002;
                  sumY2+=r*Y2*.002;
                  sumY3+=r*Y3*.002;
```

```
                sumY4+=r*Y4*.002;
            }
            else
            {
            sumY1=0;
                sumY2=0;
                sumY3=0;
                sumY4=0;
            }
            Up=sumY1*Y1;
            Up+=sumY2*Y2;
            Up+=sumY3*Y3;
            Up+=sumY4*Y4;
            Up=Up*(-K_P);
            UP=Up/0.198;//Convert from Nm to Amps store in UP, the current
control input
      /*************Calculate Robust Control Value*****************/
            if (r<0) abs_r=r*-1;
            else abs_r=r;
            if (abs_r>=eps)
                Uu=-Ku*r/abs_r;
            else
                Uu=-Ku*r/eps;
            Uu=Uu/0.198;//  Convert from Nm to Amps
      /**************Cacluate the Filtered Torque**********************
            Torque_filt = nd[0]*Torque_Sensor+nd[1]*t_1-dd[1]*a_1;
            a_1=Torque_filt;
         t_1=Torque_Sensor;  */
      /************** Torque Calculation*****************************/
            TC=Nom_Tau+Uu+UP+Torque_Sensor/(101*0.198);
      /********* Saturation Bounds for saftey ***********************/
            if (TC>=8)
            {
                TC=8;
                error_count++;
            }
            if (TC<=-8)
            {
                TC=-8;
                error_count++;
            }

            if ((Transmit_Count>=MAXCOUNT) || (error_count>20))
                TC=0;
            DemoCAN.Torque_Command=TC*0.198*101;
            Error_Torque=(Uu+UP+Nom_Tau)*.198*101;
/****************** Transmit Control Message **********************/
            if (Transmit_Count>MAXCOUNT)
            {
                DemoADC.stop = 1;                                          //
else, set stop index, last acquisition has been made
            }
            else
            {
                TCommand(TC);
                Transmit_Count++;
                if (Transmit_Count%Data_Log_Interval==0)
```

```
                    logger();
                /****Reset ADC variables for next iteration***/
                DemoADC.count_adc=0;
                DemoADC.count_saved=0;
                DemoADC.ADC_Avg=0;
            }
        }
// Reinitialize for next ADC sequence
      AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;               // Reset SEQ1
      AdcRegs.ADC_ST_FLAG.bit.INT_SEQ1_CLR = 1; // Clear INT SEQ1 bit
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;         // Acknowledge
interrupt to PIE
}
/*=============================================================
 Delay Function
---------------------------------------------------------------
 There is a 18/19 cycle overhead and each loop takes 14 cycles.
 The LoopCount is given by the following formula:
  DELAY_CPU_CYLES = 18 + 14*LoopCnt
  LoopCnt = (DELAY_CPU_CYCLES - 18) / 14
 =============================================================*/
void Delay(unsigned long LoopCnt)
{
      unsigned long i;

      for (i = 0; i < LoopCnt; i++){}
}
```

References

[1] Armstrong-Helouvry, B., Dupont, P., and Canudas de Wit, C. "A survey of models, analysis tools and compensation methods for the control of machines with friction," *Automatica*, vol. 30, no. 7, pp. 1083 – 1138, 1994.

[2] Dahl P.R. "Solid friction damping of mechanical vibrations." *AIAA J.*, vol. 14, no. 12, pp. 1675 –1682, 1976.

[3] Hsieh, C. Pan, Y.-C. "Dynamic behavior and modelling of the pre-sliding static friction," *Wear* vol. 242, pp. 1-17. 2000.

[4] Armstrong B., Dupont P., Hayward V.. "Single State Elasto-Plastic Models for Friction Compensation," IEEE Transactions on Automatic Control vol. 47, issue 5 pp. 787 – 792. 2001.

[5] Canudas de Wit, C., Olsson, H., Astrom, K. J. and Lischinsky, P. "A new model for control of systems with friction," *IEEE Transactions on Automatic Control*, vol. 40, no. 3, pp. 419 – 425, 1995.

[6] Papadopoulos E. and Chasparis G. "Analysis and model-based control of servomechanisms with friction," *Proceedings of IEEE/RSJ IntInternational Conference on Intelligent Robots and System*, vol. 3, 2002, pp. 2109 – 2114.

[7] Canudas de Witt. C. and Lischinsky P. "Adaptive friction compensation with partially known dynamic friction model," *Inter. J. of Adaptive Control and Signal Processing*, vol.11, pp. 65 – 80, 1997.

[8] Southward S., Radcliffe C., and MaCluer C. "Robust nonlinear stick-slip friction compensation," *ASME Journal of Dynamic Systems, Measurement and Control*, vol. 113, no. 4, pp. 639 – 645, 1991.

[9] Kang M. S. "Robust digital friction compensation," *Control Engineering Practice*, vol. 6, pp. 359 – 367, 1998.

[10] Song G., Wang Y., Cai L. and Longman R.W. "A sliding-mode based smooth adaptive robust controller for friction compensation," *Proceedings of the American Control Conference* vol. 5 pp. 3531 – 3535. 1995.

[11] Liu G. "Decomposition-based friction compensation of mechanical systems," *Mechatronics*, vol. 12, no. 5 pp.755 – 69 2002.

[12] Liu G., Goldenberg A.A., Zhang Y. "Precise slow motion control of a direct-drive robot arm with velocity estimation and friction compensation," *Mechatronics,* vol. 14, pp. 821–834, 2004.

[13] Goldberg D. *Genetic Algorithms in Search, Optimization and Machine Learning.* MA: Addison-Wesley, 1989.

[14] Hashimoto, M. "Advanced robot position control using joint torque sensors," *IEEE International Workshop on Intelligent Robots* Oct. 31 – Nov. 2, pp. 503 – 508. 1988.

[15] Taghirad H. "On the robust H-infinity torque control of harmonic drive systems," Ph. D. Thesis, McGill University, 1997.

[16] Hashimoto M., Kiyosawa Y., and Paul R. P., "Torque sensing techniques for robots with harmonic drives," IEEE Transactions on Robotics and Automation, vol. 9, no. 1, pp. 108 – 116, 1993.

[17] Pfeffer L. E., Khatib O., and Hake J. "Joint torque sensory feedback in control of a PUMA manipulator," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 4, pp. 418 – 425, 1989.

[18] Hashimoto M., and Kiyosawa Y. "Experimental study on torque control using harmonic drive built-in torque sensors," *Journal of Robotic systems*, vol. 15, no. 8, pp. 435 – 445, 1998.

[19] Morel G., Iagnemma K., and Dubowsky S. "The precise control of manipulators of high join-friction using base force/torque sensing," *Automatica*, vol. 36, pp. 931 – 941, 2000.

[20] Imura J., Sugie T., Yokokohji Y. and Yoshikawa T. "Robust control of robot manipulators based on joint torque sensor information," *IEEE International Workshop on Intelligent Robots,* Nov. 2 – 3, pp. 344 – 349, 1990.

[21] Kosuge K., Takeuchi H. and Furuta K. "Motion control of a robot arm using joint torque sensors," *IEEE Transaction on Robotics and Automation*, vol. 6, no. 2, pp. 258 – 263, 1990.

[22] Godler I., Ninoiya T., Horiuchi M., and Hashimoto M. "Ripple compensation of harmonic drive built-in torque sensing," *IEEE International Symposium on Industrial Electronics*, vol. 2, pp. 888 – 892, 1999.

[23] Taghirad H. D., and Belanger P. R. "Torque ripple and misalignment torque compensation for the built-in torque sensor of harmonic drive systems," *IEEE Transactions on Instrumentation and Measurement*, vol. 47, no. 1, pp. 309 – 316, 1998.

[24] Tuttle T. D., and Seering W. P. "A nonlinear model of a harmonic drive gear transmission," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 3, pp. 368 – 373, 1996.

[25] Kennedy C. W., and Desai J. P. "Modeling and control of the Mitsubishi PA-10 robot arm harmonic drive system," *IEEE/ASME Transactions on Mechatronics*, vol. 10, no. 3, pp. 263 – 275, 2005.

[26] Deb K. and Beyer, H. G. "Self-adaptive genetic algorithms with simulated binary crossover," *Evolutionary Computing*, vol. 9, no. 2, pp. 197 – 221, 2001.

[27] Vasconcelos J. A., Ramirez J. A., Takahashi, R. H. C. and Saldanha R. R. "Improvements in genetic algorithms." *IEEE Transactions on Magnetics*, vol. 37, no. 5 pp. 3414 – 3417, 2001.

[28] Herrera F., Lozano M. and Verdegay J.L. "Tackling real coded genetic algorithms: Operators and tools for behavioural analysis," *Artificial Intelligence Review*, vol. 12, pp. 265 – 319, 1998.

[29] Muhlenbein H. and SchlierkampVoosen D. "Predictive Models for the Breeder Genetic Algorithm I. Continuous Parameter Optimization," *Evolutionary Computation,* vol. 1, no. 1, pp. 25 – 49. 1993.

[30] Wright, A. Genetic *Algorithms for Real Parameter Optimization. Foundations of Genetic Algorithms 1*, G.J.E Rawlin (Ed.) Morgan Kaufmann, San Mateo, pp. 205 – 218, 1991.

[31] Eshelman L.J. and Schaffer J.D. "Real coded genetic algorithms and interval schemata," *Foundation of Genetic Algorithms 2*, L.Darrell Whitley (Ed.), Morgan Kaufmann Publishers, San Mateo, pp. 187 – 202, 1993.

[32] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs.* SpringerVerlag, 1992.

[33] Lih-Chang L., Ywh-Jeng L. "Fuzzy enhanced adaptive control for flexible drive system with friction using genetic algorithms," *Journal of Intelligent and Robotic Systems*, vol. 23, pp. 379 – 405, 1998.

[34] Davidor Y. *Genetic algorithms and robotics, a heuristic strategy for optimization*, World scientific series in robot and automated system, vol. 1, 1991.

[35] Liaw D. and Huang J. "Contact friction compensation for robots using genetic learning algorithms," *J. of Intelligent and Robotic Systems*, vol. 23, pp. 331 – 349. 1998.

[36] Aghili F. and Namvar M. "Adaptive control of manipulators using uncalibrated joint-torque sensing," *Proc. of the IEEE Int. Conf. on Robotics & Automation*, pp. 1706 – 1711, 2005.

[37] Aghili F., Buehler M. and Hollarbach J. M. "Motion control system with H-infinity positive joint torque feedback," *IEEE Transaction on Control Systems Technology*, vol. 9, no. 5, pp. 685-695, 2001.

BL-IV-30