ABSTRACT

Title of Thesis:	MATHEMATICAL MODEL OF ADAPTIVE MOTOR CONTROL
Degree candidate:	Makiko Kosha
Degree and year:	Master of Science, 1999
Thesis directed by:	Associate Professor Robert M. Sanner Department of Aerospace Engineering

An adaptive control law incorporating a biologically inspired neural networks for robot control is used as a mathematical model of human motor control and the motor control adaptation. Modeling human motor control strategy is made difficult due to the redundancies in the human motor control system. This control model is able to overcome the difficulties of the human motor control modeling, and include the learning capability of the motor control strategy which was omitted in human motor control studies until now. By adaptively piecing together a collection of elementary computational elements, the proposed model develops complex internal models which are used to compensate for the effects of externally imposed forces or changes in the physical properties of the system.

In order to examine the form of human motor control adaptation in detail, a computer simulation was developed with a two dimensional model of the human

arm which utilized the proposed adaptive motor control model. The simulation result show that the model is able to capture the characteristics of the motor control adaptation seen in human experiments reported by [14], [46]. For continuation of this research, an experimental apparatus was designed and built for the human motor control study. This apparatus is a cable driven, two-dimensional manipulator which is used to apply specified disturbance forces to the human arm. The preliminary experiment conducted with this test apparatus show a strong correlation to the simulation data and other experimental data reported on human reaching motions.

MATHEMATICAL MODEL OF ADAPTIVE MOTOR CONTROL

by

Makiko Kosha

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland at College Park in partial fulfillment
of the requirements for the degree of
Master of Science
1999

Advisory Committee:

Associate Professor Robert M. Sanner, Chairman/Advisor
Associate Professor David L. Akin
Research Associate Craig Carignan

DEDICATION

To my mom, Aunty Toyo, Aunty Kyouko, and Uncle Glen for absolutely everything.

To Kenny for all of the love and support.

ACKNOWLEDGEMENTS

Thanks to my family, you did everything to make sure that I had great opportunities. And thank you for always giving me love and support with occasional kick in my butt when I needed it.

Thanks to Kenny for always being supportive, and taking care of our little furry friends for last two months.

Thanks to Rob, my advisor. You have taught me so much and gave me an exciting research opportunity. I especially want to thank you for last few weeks. You have given me tons of your time to edit my thesis and help me prepare for my defense.

Thanks to Russ and Craig, you always made time to answer my questions, and supplied me all those excess equipment, so that I could build TSUNAMI at a minimum cost.

Thanks to Joe Graves who always gave me a different approach to attacking my problems. I really appreciate those all nighters you pulled with me just to help me.

Thanks to J.M. who also pulled several all nighters with me to help me with my project.

Thanks to Glen and Stephen for all those software help you gave me.

Thanks to Beth, Laurie, Julianne, and Steve for looking after me at the lab and outside of the lab. You guys were always ready to jump in and help me with whatever I was doing.

Thanks to Dave who gave me a wonderful opportunity to work and learn at the SSL. You have provided an amazing place for so many people to learn and to grow. Even though we may complain a lot, we do appreciate it.

Thanks to all my friends at the SSL. I would not be here finishing my thesis (at early morning of the due date) without your help.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Motivation

The early experiences of astronauts on numerous space walks during Extravehicular Activity (EVA) and on the moon in the 1960's and the 70's demonstrate the difficulties experienced by humans performing tasks in reduced gravity environments. The first successful EVA operation was accomplished by Edwin "Buzz" Aldrin during Gemini 12 mission. Aldrin was also first astronaut to perform EVA with training in neutral buoyancy in preparation for the flight. Before this mission, astronauts were not capable of conducting smooth movements in space; instead they struggled and ended the EVA completely fatigued [40]. Prior to the Gemini 12 mission, EVA training was performed solely on KC-135 flights, where the simulation of microgravity alternates with periods of hypergravity of approximately 2-g's. These altered gravity environments last respectively, only thirty seconds and fifteen seconds. The training only in the KC-135 microgravity simulation aircraft proved insufficient for learning to execute tasks in a challenging environment as microgravity.

In the mean time, wealth of knowledge has been gained during space opera-

tions. The time astronauts spent on the moon, the data collected on the human motion on the Skylab and the Mir, and even the time spent to train the astronauts for space conducted in a simulation of microgravity, such as neutral buoyancy and KC-135, have helped to illustrate the challenges that humans face when they are required to work and function in unfamiliar dynamic environments. Although humans possess remarkable capacity to learn and adapt to new situations and environments, the experiences in the space operations teaches the importance of large number of carefully planned training sessions.

With the advent of the International Space Station (ISS), the space program is faced with new challenges. The current design of space station relies heavily on EVA for initial construction and external maintenance of the station. This required number of EVA's grossly exceed National Aeronautical and Space Agency's (NASA) capability [1], thus making space operations, scheduling, and training play an even more critical role in our space program.

Human capacity to adapt to new dynamic environments allows us to operate in unfamiliar territory, such as in space and under water. Human motor control and adaptation have been studied for decades, but the level of understanding on how the adaptation and learning process occurs is quite limited and it remains a topic of current research. An improved understanding of human adaptation to altered dynamic environments would be tremendously useful to improve both the success of operational capabilities and improve the training of the astronauts required to meet the needs of the future space program. Better understanding of the motor control and the adaptation mechanism would prove invaluable in various fields of research. These include orthopedics, child development, and, as mentioned earlier, subject training in altered environments. These are a few

of the most obvious areas to benefit from a better understanding of the human motor control mechanism.

Although observed in everyday life, the adaptation mechanism of human motor control has not been modeled adequately. This thesis attempts to elucidate the structure of human motor control strategies and model how they learn to accommodate different dynamic environments, setting them in a mathematical framework amenable to simulation and analysis. In the development below, "neural" networks are used as a component of the proposed mathematical model.

The use of neural networks to model human learning has been promising, but so far, scientists have not been able to implement it in the frame work of a motor control model in any meaningful manner. Vast number of neurons form the nervous system of the human body. Furthermore, the large number of different types of neurons which make up the nervous system and its intricate and specific network connections have made the understanding of motor control impossible thus far. The precise modeling of the connection of billions of neurons is not only unfeasible, but has very little meaning. The neural connections are very redundant, the precise connections may be different in each and every person. Furthermore, precise connections change continuously, and in some cases drastically. For example, it is known that in the cases of brain surgery performed on the patients with sever seizures, large portions of the brain may be removed, but the functions computed in those areas are learned to be computed in other parts of the brain.

Until recently, researchers have treated the modeling of neural science using neural networks as a black box, only able to connect an input to an output without any considerations for what the body represented by that black box is

doing, since it is impractical to attempt to model what every neuron is doing. But, by definition, this type of approach does little to reveal what the human body is actually doing. In addition, the network used might be able to create a sensorimotor map, but if any small changes to the system are made, then nothing can be said of the next condition.

It is beyond our current computational power to try to model each and every neuron, how they are connected to each other, and how the connections result in human behavior. Instead of trying to solve the problem by mapping from behavioral goals of motor control to the neural input activating the muscles, the understanding of motor control can be gained progressively by establishing a hierarchy of motor control levels [6] [51]. This approach allows the investigator to address more meaningful questions while still advancing the knowledge of motor control study. Although, bridging the gap between motor control levels will be difficult, many questions can be addressed by studying each levels individually. The proposed model takes the understandings contributed by several previous experiments, analyzing each level, and then closes one of the gaps of hierarchy. Specifically by providing a model of the observed plasticity of motor control strategies.

1.2 Organization of Thesis

This thesis opens with the motivation for the investigation of human motor control and its adaptation mechanism. In Chapter 2, the current understanding of the nervous system and the muscle physiology is presented. The basic knowledge of how neurons communicate with each other, how networks are built, and how neurons activate muscle contractions is described. The background on physiol-

ogy of motoneurons and muscles provided in this chapter assists in evaluating the model and offers insight into the difficulties of this task. Chapter 3 reports the results of current motor control research. The basic principles of movement organization provide us with an approach to appropriately structure the model that is capable of capturing the characteristics of the observed movement. Chapter 4 follows with the description of the proposed mathematical model, along with the capabilities and the limitations of the model. This is then followed by the results from a simulation of this model which is shown to provide good qualitative agreement with previously reported experiments on human subjects. In chapter 5, the design and construction of a haptic device capable of performing further experiments to validate the proposed model are described. Preliminary test results obtained with this device are then presented. Chapter 6 reviews the research results, offers some recommendations to improve the test apparatus, and concludes with a discussion of the future research.

Chapter 2

The Physiology of Limb Motion

2.1 Introduction

Similar to a robotic system, human limb motion can be viewed in an engineering sense with the human arm modeled as a manipulator and the central nervous system (CNS) playing the role of the manipulator controller structure. The CNS commands efferent signals to the muscles to execute motion while an afferent signal is sent back from the muscles to transfer sensory information such as position, velocity, and force data. At the higher level of the CNS, commands are calculated by a 'joint controller' which computes the necessary torque to follow a specified reference trajectory. The reference trajectory is similarly calculated by a trajectory planning module to accomplish a specific task.

The mathematical model proposed in this thesis addresses the modeling of the CNS, and the detailed discussions are covered in chapters 3 and 4. This chapter presents the reader with a basic understanding of the human neural, muscular, skeletal system and upper limb motor control. A more complete information of this chapter can be found in [25]. This chapter attempts to reveal the difficulties of modeling human motor control system and give insight on the relevance of the

adaptive motor control model presented in this thesis.

2.1.1 Anatomical Organization of the Nervous System

The command center of the human body, the CNS, consists of the brain, the brain stem, and the spinal cord. It is responsible for the complex tasks of the human mind, such as pattern recognition, cognition, and long term memory. These functions, which are localized to discrete regions in the brain, are not actually complex faculties of mind, but rather elementary operations [25]. More elaborate faculties are constructed from the serial and parallel interconnections of several brain regions.

The neuronal functions are hierarchically and anatomically organized. Complex mental functions are organized in the brain, while the most elementary functions of motor coordinations, for example spinal reflex, are calculated in the spinal cord. Neural circuits in the spinal cord play an essential role in motor coordination. These spinal reflexes are very fast because the networks are connected simply and the information travels short distances. Although the neuronal commands mediating the spinal reflex are very simple, the higher centers can influence the spinal reflexes to generate more complex behavior. Reflex circuits provide the nervous system with a set of elementary patterns of coordination that can be activated either by sensory stimuli or by descending signals from the brain stem and cerebral cortex. Reflex circuits provide the higher centers with a set of elementary patterns of coordination. These wired-in movement patterns are nevertheless remarkably adaptable to current conditions. For execution of motion, the speed of motion is improved greatly in this way. This occurs because environmental information, such as the initial position of the body segments and

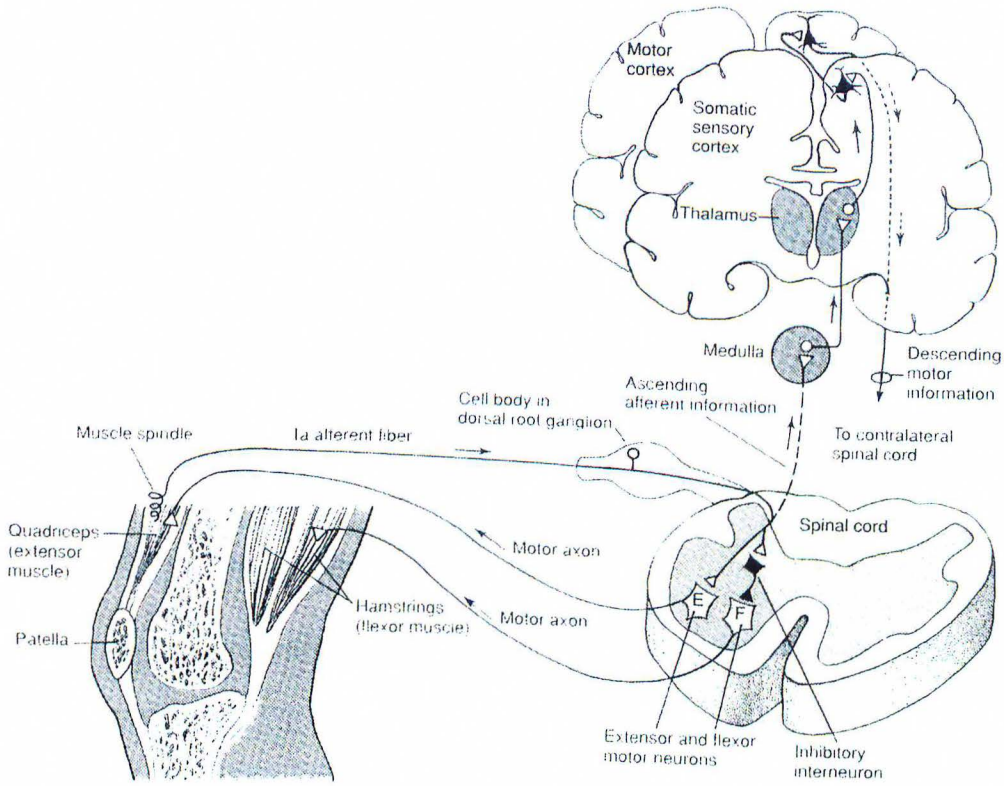

Figure 2.1: A sketch of a brain, spine, and muscle. Reproduced from Principles of Neural Science.

external loads acting to oppose movement, reaches the lower levels directly. The higher centers can activate these reflex circuits to produce voluntary movement patterns.

2.1.2 Advantage of Networks

The architecture of the human brain is comprised of basic elements. But despite the simplicity and the similarities of the basic properties of different types of nerve cells, the precise network of countless nerve cells can produce many complex and

intricate behaviors. The best estimate today of the number of neurons in the human brain is approximately 10^{11} neurons. The means by which these neurons are connected can be modified with experience.

Though there are as many as 10,000 different types of nerve cells, they all share several common features. The functional similarities of the nerve cells are described in the following sections.

2.2 Physiology of Neurons

2.2.1 Neuronal Function

There are two fundamental principles governing neuronal function. The principle of dynamic polarization states that information flows in a predictable and consistent direction within each nerve cell. In most cases, the signals travel from the dendrites to the axon hillock, where the action potential is generated. When the action potential is triggered, it travels through the axon unidirectionally toward the presynaptic release site.

The second principle, the principle of connectional specificity, states three important considerations about the neural connections. First, there is no cytoplasmic continuity between nerve cells. Second, the nerve cells do not connect indiscriminately to one another. Third, each cell makes specific connections at precise and specialized points of synaptic contact.

There are three types of neurons: afferent, efferent, and interneurons. The afferent neurons carry the information provided by the sensory neurons into the nervous system. The motor signal, which is also called the efferent signal, is the information carried by the motor neurons into the muscles and glands. The in-

terneurons constitute the largest number of neurons, and they connect all types of neurons to each other. There are two ways the neuronal connections can improve communications between neurons; these are neuronal diversions and neuronal conversions. Neuronal diversion refers to a situation where a single neuron branches many times and terminates on many target cells. Neuronal diversions allows a single neuron to exert a widespread influence by distributing its signals to many target cells. Neuronal conversion refers to a situation where many sensory cells terminate on single motor cell. This allows a target cell to integrate various information from many sources.

2.3 Human Arm Motion

Activation of Muscles

An action potential is generated in a motoneuron by the spatial and temporal summation of incoming action potentials from other neurons. Action potentials travel in about 100 milliseconds. The action potential travels from the motoneuron's axon and invades the several branches of the postsynaptic neuron, each with a terminal that is closely apposed to a skeletal muscle fiber. The action potential arriving at the neuromuscular junction triggers a sequence of biochemical events which results in muscle contraction [25].

2.3.1 Muscle Function

Production of Force

The smallest subunit that can be controlled is called the motor unit. The motor unit consists of a single motoneuron and the muscle fibers it innervates. The

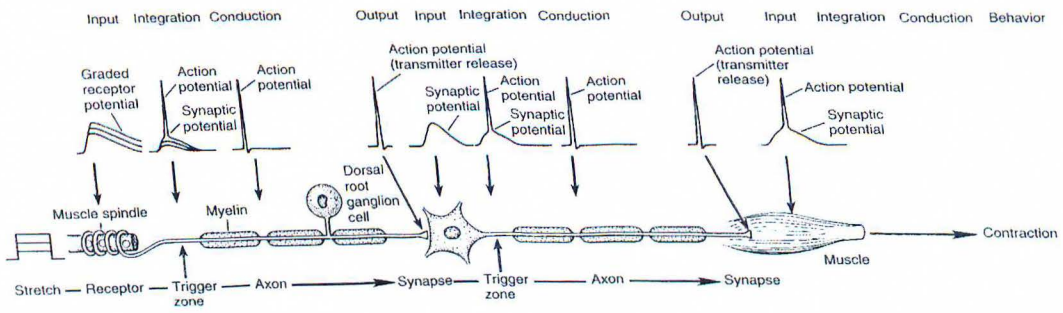

Figure 2.2: Transmission of action potentials. Reproduced from Principles of Neural Science.

innervation ratio (number of muscle fibers innervated by one motoneuron) is roughly proportional to the size of the muscle. Low innervation ratio means that there is finer control of the muscle's total force. Muscles work a lot like a spring. Muscles generate restoring force when they are stretched beyond their resting length. At a point where muscle is stretched beyond the resting length, the muscle starts to produce restoring force proportional to the muscle length. When muscle is stimulated by a motoneuron, the contractile elements shorten similar to excising the slack of a rubber band. When a rubber band is compressed, it just becomes slack. Like a rubber band, if the muscle contraction did not take up the slack, the muscle also would be slack when it is compressed shorter than the set point (resting length). Therefore, the muscle contraction causes restoring force to be generated at a much shorter muscle length. If a load acts on the muscle as if there were a weight attached to the end of the muscle, the weight would be pulled up progressively as the rate of motor neuron stimulation to the muscle is increased, until the weight equaled the muscle restoring force. The length at which the muscle comes to rest for a given stimulation and load is called the equilibrium point. This is the basis of the equilibrium point hypothesis discussed

in the next chapter.

Muscles contract slowly relative to the time course of a single neuron. Therefore, the train of impulses may activate the muscle fiber. Then, the action potentials temporally sum until a plateau of muscle force is reached. The plateau is called tetanus. If the stimulation is relatively slow, and reaches the muscle after the peak force of the twitch is reached, the tension has a ripple and is called unfused tetanus. But if the impulse comes in so fast that the average force increases progressively to a maximum value, the tension becomes smooth because the individual twitches can no longer be distinguished. This is called fused tetanus. The force production depends on the pattern of stimulation, as well as the rate at which muscle fibers are stimulated. The rate of action potential does not correspond linearly to the produced tension. Rather, an insertion of a single extra action potential in a low-frequency train of action potentials profoundly enhances the tension [25]. Following an extra impulse, muscles can be transiently activated at a higher rate than required for the maintenance of a steady level of force. The neuromuscular system takes advantage of this to make quick movements.

The nervous system controls the force of muscle contraction in two ways: by size and by rate modulation. The size principle states that when a motor neuron pool is activated, the smallest cell bodies are recruited first. As the synaptic inputs increase, progressively larger motor neurons are recruited. Rate modulation states that the nervous system can vary force by modulating the rate of firing of motor neurons. Increases in force with increasing firing frequency allows successive twitches to summate more effectively. The recruitment and rate modulation are not mutually exclusive, but the nervous system controls force output by using both methods.

Limitations of Muscle as an Actuator

The muscle properties described above limit the control strategies in two substantial manners. First, instead of controlling the length of contraction or the force produced by the muscles directly, the motor systems control the stiffness and the set point (resting length) of the muscle. The joints in the human body are actuated with two opposing muscles, and there are two strategies by which a joint can be actuated with two contracting muscles. First, a joint can be controlled by reciprocal innervation, where the agonist muscle increases the stiffness and contracts to cause a joint motion, while the inhibited antagonist muscle relaxes. Reciprocal innervation is energy efficient, but it requires that loads are accurately known by central processes involved in planning the movement, because the stiffness of the antagonist muscle, which is analogous to the gain of the PD controller, is decreased. On the other hand, in co-contraction both agonist and antagonist muscles to contract. The actuation takes place because the agonist muscle contracts harder than the antagonist muscle. This method uses more energy but does it increases the stiffness of the joint, analogous to increasing the gain of the PD controller, therefore does not require loads to be known precisely. This method also potentially provides more accurate motion and greater adaptability during unanticipated changes in external loads. When contracting two opposing muscles, the CNS can vary the stiffness and the angle of the joint. Like the increased gains of a controller, the increased stiffness of a joint causes it to be influenced less by an unanticipated external force.

Another limitation of the muscle is its slow response to neural activation. Muscles act like low pass filters, and the nervous system does not control skeletal muscles through a linear relationship. Changes in muscle tension represent

a transformation of the frequency of neural impulses. When the frequency of the action potential train is increased, the magnitude of the change in tension decreases and the change in tension lags progressively more. The muscle can not transform high frequencies in the modulation of neural impulses into fluctuations in force. To produce a fast rising force, motor systems activate the agonist muscle to a greater degree and more quickly than normal and also use the antagonist muscle to slow down. The change in muscle length depends on multiple factors: neural drive, initial length of the muscle, and external loads.

2.3.2 Muscle Receptors

To make movements, the human body needs to know information about its kinematic state and its environment. Proprioception is mediated by muscle spindles and Golgi tendons. The functional difference in response of muscle spindles and Golgi tendons can be explained by their different anatomical arrangements within the muscles. The Golgi tendons are connected in series to the extrafusal muscle fibers, while muscle spindles are connected in parallel to the intrafusal muscle fibers. Because Golgi tendons are much stiffer than extrafusal muscle fibers, during tension most of the stress is taken up by the extrafusal fibers. Therefore, the response of Golgi tendons during stretching is weak and inconsistent. During muscle contraction, the extrafusal muscle fibers pull on the Golgi tendon, resulting in a strong response to compression. By contrast, the intrafusal fibers which are innervated by the muscle spindles are connected in parallel to the muscle fibers. For this reason, as the extrafusal muscle fibers are loaded under tension, the intrafusal fibers are also stretched, resulting in a strong responses from the muscle spindles. On the other hand, when the muscles contract, intrafusal fibers

slacken and the muscle spindles produce no response. The tendon organs provide supplemental information to the muscle spindles about state of muscles: muscle spindles sense relative position of the limb segments while the tendon organs sense tension level of the muscle. In other words, the muscle spindles provide position feedback while Golgi tendon organs provide force feedback.

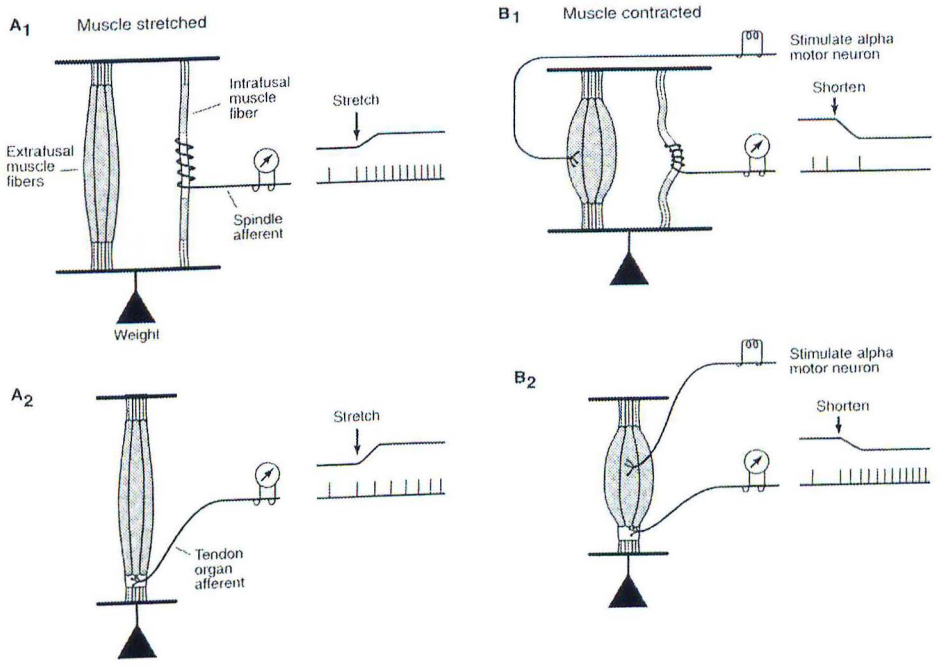

Figure 2.3: Connections of Golgi tendon organ and muscle spindles. Reproduced from Principles of Neural Science.

2.4 Summary

The control of the actuation is controlled by the CNS. There is a hierarchical organization of the nervous system. The complex computations takes place in the brain, the central processor. The signals commanded by the brain travel

through the conduit of the spinal cord which also contains a number of elementary computational elements. These signals from the brain commands how the computational elements in the spinal cord execute actuation by activating the muscles. Embedded within the muscles are muscle spindles and Golgi tendon organs which feed back the velocity information and the force information respectively to the spinal cord, and successively to the brain.

Chapter 3

Abstract Models of Motor Control

The most difficult problem with human motor control study is that human motion is plagued with redundancies. The excess degrees of freedom exist at many levels, and this problem is known as Bernstein's problem [6]. First, the solution of muscle activation used to produce a joint torque is not unique. Because most limb segments are actuated by several muscles, and since some of these muscles actuate multiple joints, multiple combinations of the muscle activations can produce the same net joint torque. Another redundancy exists at the kinematic level; several combinations of joint angles can produce the same hand position and orientation. The third redundancy exists at the task level. A specific task can be accomplished in many different ways. For example, to accomplish a goal such as point-to-point reaching, the hand can follow many different paths and still end up at the desired target position. Therefore, Bernstein's problem poses a real challenge to any motor control study [51]. Researchers have therefore approached this problem in the study of motor control by attempting to gain knowledge at individual levels of motor control organization. Some of the current understandings about the hierarchical nature of the human motor control system are presented in this chapter.

3.1 Virtual Trajectory

Although there are redundancies at the task level,the kinematic level, and joint level, there is still a distinct method by which a person completes a task under various conditions. For example, a person's handwriting looks very similar regardless of whether it is written on a paper with the use of finger and wrist motions or if it is written on the black board with the use of the upper limb motions. It is unlikely, given the size of the memory, that there are multiple models of the movements which accomplish a specific task in every possible situation. Therefore, the similarity in the outcome suggests an existence of a global rule at the task level [51]. This also suggests that the performance of every motion incorporates both the abstract information derived from long-term memory and the available information about the current task [51].

The objective of the studies of planning and control of multi-joint movement is to identify a common kinematic feature or stereotypical patterns of muscle activation which characterize a typical motor behavior. This presents two fundamental questions. What coordinate frame does the brain use to represent motion? And, how does the brain choose a certain trajectory from the infinite number of possibilities? To answer these questions, researchers have set out to conduct an experiment to observe the patterns of invariance in behavior. Some researchers found that the trajectories are planned in joint variables [13], [50] while others have argued that simplicity of motor control is achieved by planning hand trajectories in extracorporeal space [6]. Joint rotations are then tailored to produce the desired hand trajectory. Experiments conducted on both monkey and human reaching motions [2], [14], [32] indicate that for point-to-point reaching motion, the subjects tended to generate hand paths which are roughly straight lines to

the target with single-peaked, bell-shaped speed profiles. These movement characteristics were evident independent of the part of the work-space in which the reaching movement was performed. This result indicates that common features existed in extracorporeal space. The plots showing a typical point-to-point reaching path and velocity profile are in Figure 3.1.

Natural movements of a person is smooth and graceful, instead of awkward and jerky. To describe 'smoothness' mathematically, an optimization function was selected to match the observed motion of Figure 3.1 [23]. The cost function minimizes jerk, the derivative of acceleration:

$$C_T = \frac{1}{2}\int_0^{t_f}\left[\left(\frac{d^3x(t)}{dt^3}\right)^2 + \left(\frac{d^3y(t)}{dt^3}\right)^2\right]dt \tag{3.1}$$

where the movement duration ranges from 0 to t_f, and $x(t)$ and $y(t)$ are the position of the hand in Cartesian coordinates during the motion. Based on this definition of smooth motion, planning of the movement depends on the initial and final position of the movement. The following equations (3.2) express 'minimum' jerk trajectory [23].

$$\begin{aligned} x(t) &= x_o + (x_f - x_o)(6(\frac{t}{t_f})^5 - 15(\frac{t}{t_f})^4 + 10(\frac{t}{t_f})^3) \\ y(t) &= y_o + (y_f - y_o)(6(\frac{t}{t_f})^5 - 15(\frac{t}{t_f})^4 + 10(\frac{t}{t_f})^3) \end{aligned} \tag{3.2}$$

The Figure 3.1 shows notable similarities between the simulation and the experimental data. This model exhibits three characteristics. The trajectory of the hand follows a straight path; the velocity profile of the trajectory is smooth and unimodal; and the shape of trajectory is invariant under translation, rotation, amplitude and speed scaling [51]. Experimental results exhibit the same characteristics when the above parameters are varied.

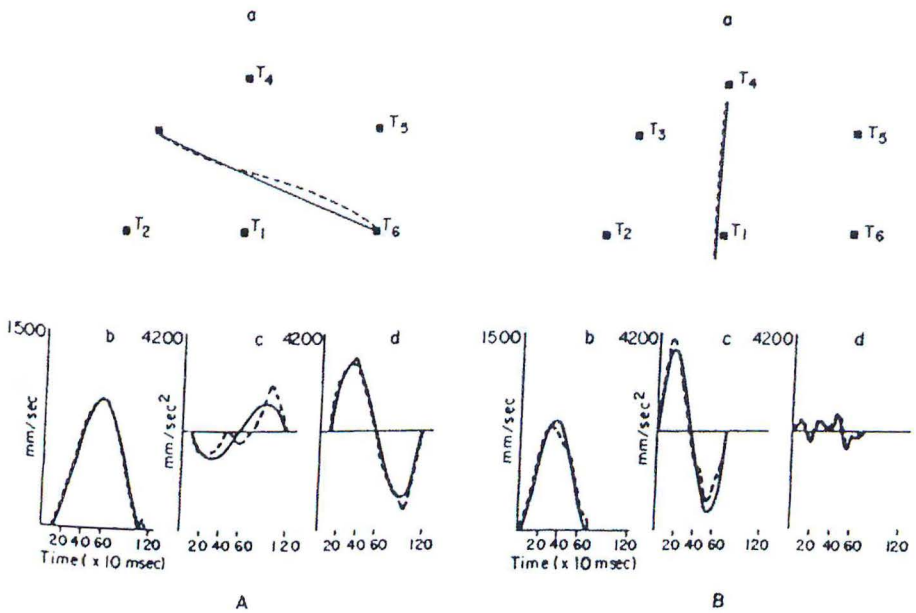

Figure 3.1: Experimental result and simulatin result of virtual trajectory compared, where solid line is the simulation data and dashed is the experimental data. This figure shows the hand paths, speeds, and acceleration components in y axis and acceleration components in x axis for two reaching motions. This figure reproduced from Principles of Neural Science.

These studies do not indicate that the human body measures jerk and calculates the trajectory equation. This theory only supports that there is a certain task dependent reference trajectory used during reaching motions which is captured by this model, the 'minimum jerk'. It may be that the smoothness of motion is an outcome of the intrinsic properties of the neural and musculo-skeletal hardware. However, this finding is important because, it suggests that the reference motion signal and the low level control structure which 'tracks' this reference can

be addressed independently.

3.2 Task Execution

Now that the task level redundancy is addressed, how does a biological system control actuation and calculate the necessary torque to execute motion? The following two sections discuss the results which reveal some of the actuation characteristics and the possible motor control strategies.

Several researchers have conducted experiments to observe the brain activity of a monkey while executing reaching motions [19]. The experiment result indicated that there is a correlation between the neural activity in the brain and the direction of reaching motion. Furthermore, Bizzi et. al. summarize the conclusions made by [19], [20], [24] that the movement control is represented in an extracorporeal space in the cortical cells, and that the exact manner of how the muscles are activated is not represented in the motor cortex [7]. These studies inspired a new experiment to investigate the manner in which the CNS transform the planned movements into muscle activation. This question has to handle the redundancy problem at the actuation level and the kinematic level.

3.2.1 Equilibrium Hypothesis

According to the equilibrium hypothesis, the combination of intrinsic muscle properties and the contraction of the muscle from the motor neuron activation exerts a restoring force. An experiment was conducted on a large number of bullfrogs where the restoring force of the leg muscle was measured at several coordinate locations when a microstimulation was applied to the spinal cord [7].

The force field compiled from a group of restoring force vectors converges to an equilibrium point. The equilibrium hypothesis states that the limb posture is maintained by specifying the equilibrium point.

Equilibrium trajectory hypothesis states that once the equilibrium point is achieved, the limb movements result from a shift in the neurally specified equilibrium point [12]. This theory claims that limb motion resists external perturbations by providing instantaneous corrections force when the limb is away from the intended trajectory. Corrective force provided by the elastic properties of the muscle.

The controller using an equilibrium hypothesis was investigated by creating a simulation modeling the human motor control during an execution of reaching motion in a well known environment (3.3) [14].

$$\tau = -\boldsymbol{K}_P(t)(\mathbf{q} - \mathbf{q}_{ref}) - \boldsymbol{K}_D(t)\dot{\mathbf{q}}_{ref} \tag{3.3}$$

Controller expressed with this equation matches well with the experimental human data as seen in Figure 3.2.

3.2.2 Muscle Properties

Mussa-Ivaldi et al. determined the elastic and viscous properties of muscle in their experiments. The stiffness matrix calculated from the experimental result is nearly symmetric, therefore the muscles behave primarily like springs [35]. Flash compared the muscle model using the elastic and viscous muscle properties, and found out that simulation using a PD controller-like muscle properties during upper limb reaching motions compared well to the experimental results [14]. This study supports the hypothesis that the PD style control strategy can be used to

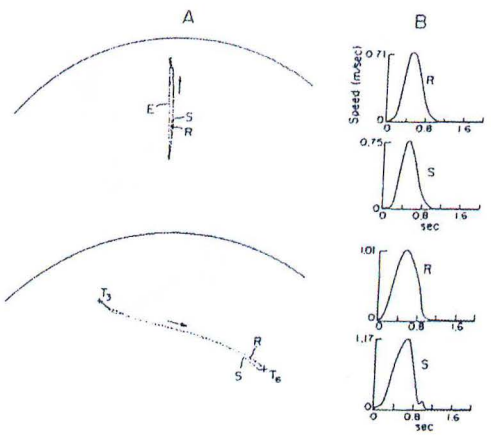

Figure 3.2: Equilibrium trajectory: simulation data is labeled S and the human data is labeled R. The two reaching paths are shown in A, while velocity data are shown in B. Reprinted from [14]

model the control strategy utilized during an execution of point-to-point reaching motion.

3.2.3 Low-level Computational Elements

To provide a more detailed picture of exactly how the CNS motor commands are structured, Bizzi et al. conducted an experiment where several microstimulations were provided to the spinal cord of a bullfrog. At the same time, the isometric force produced by the leg muscle was measured at the ankle. The measurements indicated that for each site stimulated in the spinal cord, there exists a structured pattern of force vectors referred to as the convergent force field (CFF). The experimental setup and the CFF are shown in Figure 3.3. This study also showed that the simultaneous stimulation of two distinct areas produce a force equivalent to the linear combinations of the force fields obtained from the individual

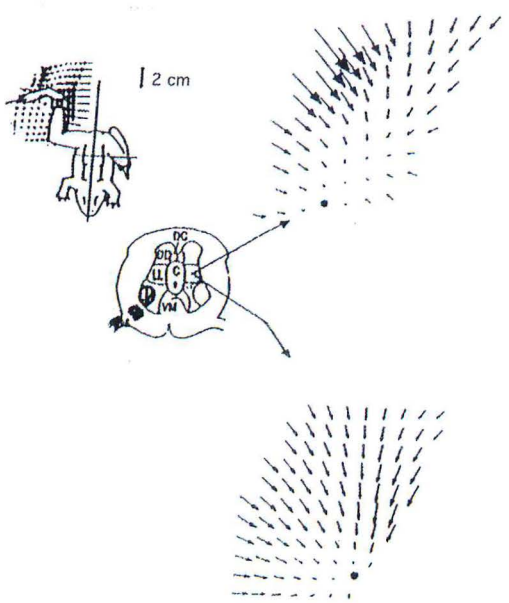

Figure 3.3: Evidence of low-level computational elements. Reprinted from [7]

stimulations of the two areas [7]. This result is also confirmed by several other researchers [36]. From this result, several researchers have hypothesized that the 'PD'-like motor control strategy is created by various linear combinations of these simple, low-level computational elements found in the spinal cord.

Figure 3.4 shows the result from the simultaneous excitation of two distinct sites within the spinal cord. Section A and B are the resulting CFF's from two independent applications of microstimulations at two different sites in the spinal cord. The CFF produced by the costimulation of the tow sites is shown in section C, while the summation of the two measured force fields is shown in section D. Note the strong correlation between C and D.

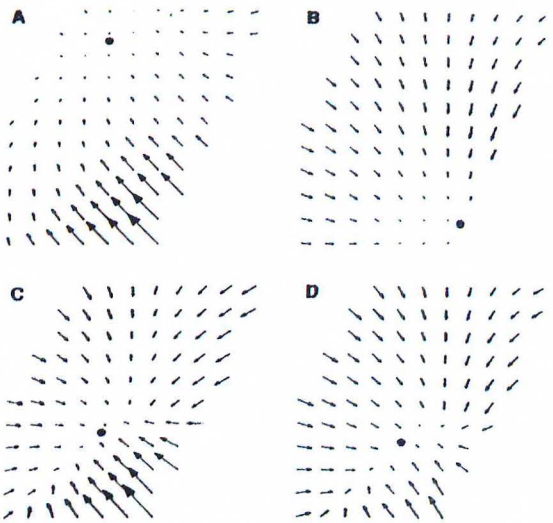

Figure 3.4: Convergent force fields: section A and B are the resulting CFF's from two independent applications of microstimulations at two different sites in the spinal cord. The CFF produced by the costimulation of the two sites is shown in section C, while the summation of the two measured force fields is shown in section D. Note the strong correlation between C and D. Reprinted from [36]

3.3 Internal Model

A simple PD control algorithm is able to simulate successfully the reaching motion executed in a familiar environment, but how does the execution of motion change in a presence of unknown disturbance forces? How does the trajectory respond to a systematic disturbance force if the motion was executed repeatedly? A study conducted by Shadmehr and Mussa-Ivaldi [46] suggests that instead the CNS builds an internal model of the disturbance forces, which is used in the motor control strategy. The internal model is the mechanism the nervous system uses to predict the forces that would be acting on the hand as it performs the task.

In this experiment of adaptive motor control strategy, the test subject was

seated in front of a manipulandum. The elbow of the subject was suspended with a harness to maintain the motion in a two-dimensional plane. The test subject practiced the reaching motion grasping the manipulandum in null field, which is a condition of no disturbance force applied to the test subject. After the practice phase, the test subjects were instructed to make reaching motions in a learning workspace. The reaching motions were conducted mostly in the disturbance force field, and few motions were conducted in the null field. The subjects were moved to make reaching motions in another workspace to study if the learning is generalizable. In the second work space, the subjects were making reaching motions in null field, force field 1, and force field 2. The first type of force field, referred to as force field 1, is a group of forces which was computed as a function of the velocity of the hand (3.4).

$$f = \boldsymbol{B}\dot{x} \tag{3.4}$$

where f represents the force acting on the end-point of the arm. The force field matrix, $\boldsymbol{B}$ represents the viscous, environmental force applied on the hand during reaching motion expressed in the Cartesian hand coordinate. The second force field, referred to as the force field 2 is shown in equation (3.5).

$$\tau = \boldsymbol{W}\dot{q} \tag{3.5}$$

where τ is the torques acting on the elbow and the shoulder joints during motion. This force field, $\boldsymbol{W}$, is the viscous environmental force field which is a function of joint velocities $\dot{q}$. $\boldsymbol{W}$ is calculated so that the two effective disturbance force fields are the same in the learning workspace. If the internal model is built in extrinsic coordinate frame of the hand, then the force field 1 appear same in different workspace but force field 2 will apply different force field in different

workspace , while if the internal model is built in intrinsic coordinate frame, the force field 2 appear same in different workspace and force field 1 appear different in different workspace.

3.3.1 Reference Trajectory

This experiment analyzed the point-to-point reaching motions conducted by the test subjects in a horizontal plane with and without the disturbance force field. The analysis of reaching motion executed in the null field exhibit the same trajectory characteristics as described by the 'minimum jerk trajectory'. Under the influence of disturbance, deviation from the straight path of the trajectory is observed at the initial exposure to the force field. It appears that the unexpected disturbance force push the arm away from the straight path. In response, the subject corrects the hand motion, causing a the reaching motion to look like a hook.

3.3.2 Learning

Evidence of progressive learning is seen in Figure 3.5. With practice, the deviation in the reaching motion become smaller and smaller, and the path approaches the original reaching motion evident in the trajectory in the null field. This learning pattern suggests that indeed there is a desired, minimum jerk trajectory that the reaching motion prefers, and even in the disturbance force field, the controller attempts to return to the desired trajectory by compensating for the disturbance force.

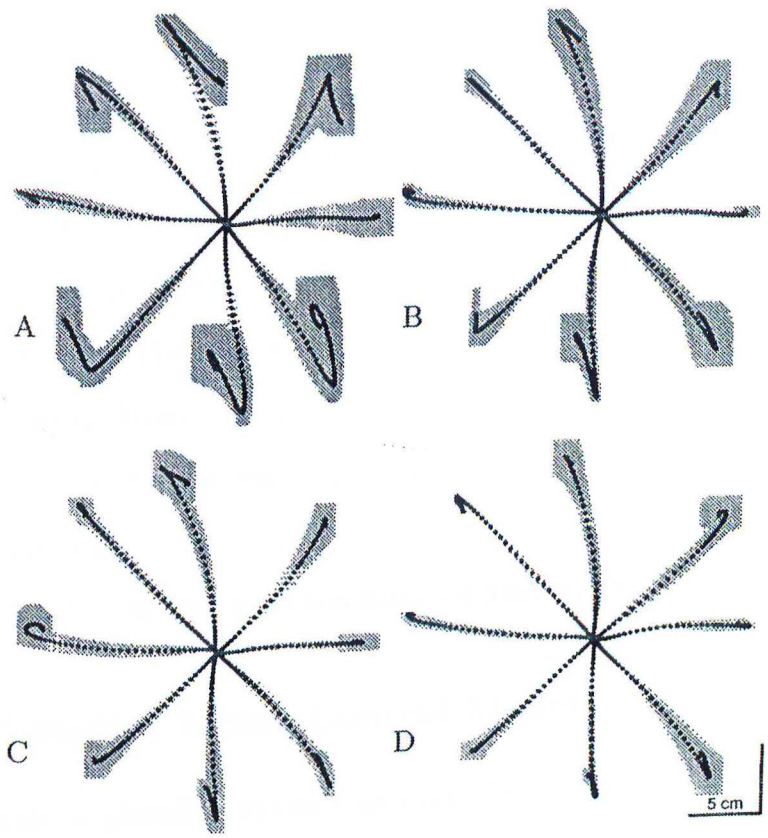

Figure 3.5: Experiment result: hand trajectories with disturbance force field during the learning period. The dark line is the average value while the gray area is the standard deviation. Path of the reaching motion after (A) 250 reaching motions (B) 500 reaching motions (C) 750 reaching motions (D) 1000 reaching motions. This figure and one that follows are reproduced from [46]

3.3.3 Aftereffect

When the force field was unexpectedly removed, instead of producing the original reaching trajectory of straight motion to the target, the subjects produced a trajectory with a deviation from the path. The hook found in the reaching path is opposite direction from the movement deviation seen during the initial exposure to the disturbance force field. This evidence of aftereffect shows that the force compensation is not simply handled with stiffer muscles, but an internal model is being built for the disturbance force. The aftereffect is the signature of the internal model [46]. The aftereffect is also evident outside of the workspace where learning occurred. From comparing the evidence of learning in two types of force fields, it is evident that the learning is generalized in intrinsic coordinate system. Also it indicates that the learning is not accomplished in the form of a look up table, but it is by building a generalizable internal model.

3.3.4 Adaptive Motor Control Model

To investigate a plausible model capturing the evidence of learning during a trajectory following task of reaching motion, another control model was proposed by Shadmehr and Mussa-Ivaldi [46]. Instead of the simple PD-control model, Shadmehr and Mussa-Ivaldi suggested an adaptive computed torque controller as a model of motor control which captures the evidence of the internal model as seen from the experimental result [46].

$$\tau(\mathbf{q}, \dot{\mathbf{q}}, t) = \hat{\boldsymbol{H}}(\mathbf{q})\ddot{\mathbf{q}}^m(t) + \hat{\boldsymbol{F}}(\mathbf{q}, \dot{\mathbf{q}}) + \hat{\boldsymbol{E}}(\mathbf{q}, \dot{\mathbf{q}}) - \boldsymbol{K}_D\dot{\tilde{\mathbf{q}}}(t) - \boldsymbol{K}_P\tilde{\mathbf{q}}(t) \qquad (3.6)$$

where $\boldsymbol{K}_D$ and $\boldsymbol{K}_P$ are constant, positive, definite matrices, $\tilde{\mathbf{q}}(t)$ is the state error, $q^m(t)$ is the minimum jerk trajectory, and $q(t)$ are the current arm joint

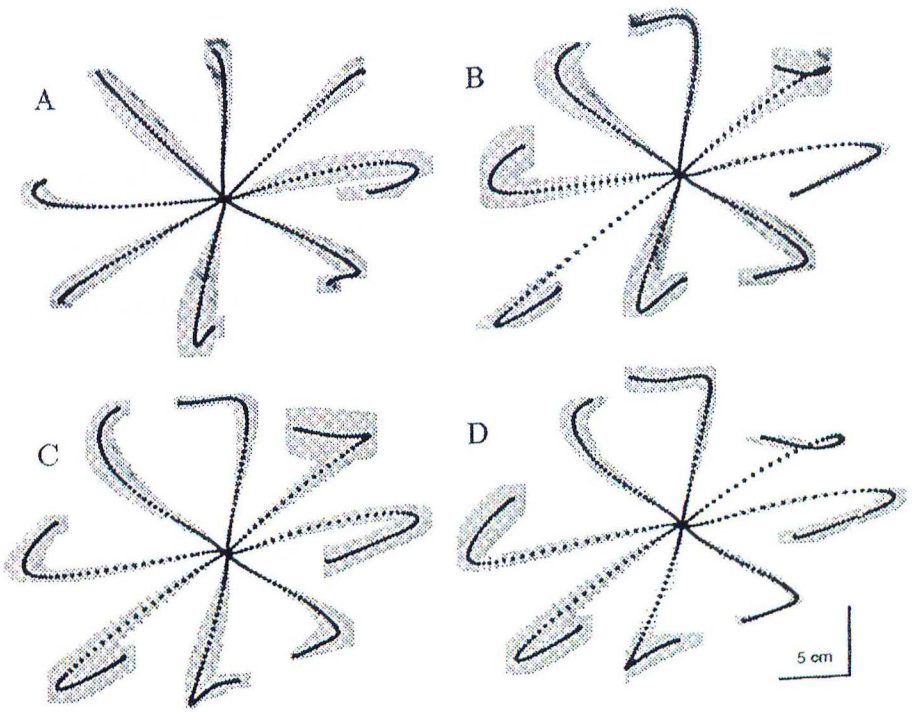

Figure 3.6: Experiment result: hand trajectories without disturbance force field during execution of motion. Path of the reaching motion after (A) 250 reaching motions (B) 500 reaching motions (C) 750 reaching motions (D) 1000 reaching motions.

angles, so that $\tilde{\mathbf{q}}(t) = q(t) - q^m(t)$. The terms $\hat{\boldsymbol{H}}$, $\hat{\boldsymbol{F}}$, and $\hat{\boldsymbol{E}}$ are the adaptive internal models of the arm inertia, centripetal and Coriolis forces, and external forces respectively. The fixed component of the equation assured that the closed loop arm motion approach the desired trajectory with stiffness and viscosity. The internal model of the environmental forces acting on the arm in the controller is $\hat{\boldsymbol{E}}$, and estimates of the true values are iteratively built as the task is practiced. Although in the formulation of computed torque, the construction of the internal model by the neural system is represented, Shadmehr and Mussa-Ivaldi did not address how the internal models are learned [46]. The development of a motor control model including the adaptation is discussed in detail in the next chapter.

Chapter 4

Mathematical Model of Adaptive Motor Control

This chapter details a control algorithm from adaptive robot control which allows simultaneous learning and control of multi-joint arm and contains components analogous to those identified in chapter 3. Although this control algorithm was developed independently in robot control, it nonetheless appears to capture some of the interesting characteristics of human motor control and the motor control adaptation which have been omitted in previous models.

4.1 Models of Arm Dynamics and Control Architecture

Consider the central nervous system as a controller which commands the execution of motion. The proprioception sensing can be considered as a position and velocity feedback path. The sensory system of the human is not addressed in this model and is assumed perfect for this study, neglecting lags and noise.

The study by Hogan [23], presented in the previous chapter, shows a convincing evidence that the desired trajectory is decoupled from the control algorithm.

And the desired trajectory attempts to minimize jerk [23]. This finding allows a use of a control algorithm from the field of robot control, which separates the desired trajectory calculation and the formulation of tracking control algorithm. In this study, the 'minimum jerk trajectory' is assumed as the desired trajectory.

In the analysis to follow, the mechanics of the human arm is simplified and modeled as a rigid two-link manipulator with revolute joints with torque about the joint center. The calculated output torque about the i-th joint is the sum of all of the torques applied by the muscles around that joint. Hence the redundancy of the joint actuation is not addressed. To create a mathematical model of actuation of multi-joint arm, a large number of solutions are available in the field of robotics where many years of research has been conducted in the area of dynamics and control. Equation of motion of a two joint robot arm can be expressed in the form of (4.1)

$$\boldsymbol{H}(\mathbf{q})\,\ddot{\mathbf{q}} + \boldsymbol{F}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{G}(\mathbf{q}) + \boldsymbol{E}(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\tau} \tag{4.1}$$

where $\boldsymbol{H}$ is the $n \times n$ inertia matrix, $\mathbf{q}$ is the $n-$vector describing the joint angles of the robot arm, $\boldsymbol{F}$ is the $n-$vector including the Coriolis and centripetal torques, $\boldsymbol{G}$ is the $n-$vector including the gravitational torques, (therefore is equal to zero when the motion is constrained to a horizontal plane), and $\boldsymbol{E}$ is the $n-$vector representing the external environmental force.

4.1.1 Feedback

A trajectory following motor control model incorporating the intrinsic properties of muscles can be accomplished by using PD feedback control, where proportional term represents the stiffness and the derivative term represents the viscous property of the muscles [14]. Simple PD control model is able to simulate a reaching

motion conducted in a familiar environment. But to model arm motion in a disturbance force field and the learning of the disturbance force, more complex model is necessary.

4.1.2 Passivity Controller

To address how the internal models adaptively change, robotics suggests a particular controller which is better suited for continuous adaptive operation. First, to discuss this controller, introduce a new error metric.

$$\mathbf{s}(t) = (\frac{d}{dt} + \mathbf{\Lambda})\,\tilde{\mathbf{q}} = \dot{\tilde{\mathbf{q}}}(t) + \mathbf{\Lambda}\tilde{\mathbf{q}}(t) \tag{4.2}$$

where Λ is a constant, positive definite matrix. Note that definition (4.2) states that if the controller is able to keep the condition of $s = 0$, the tracking error will converge exponentially to 0. Using this formulation of error, the passivity based controller can be expressed as (4.3).

$$\boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}}, t) = \hat{\boldsymbol{H}}(\mathbf{q})\ddot{\mathbf{q}}^r(t) + \hat{\boldsymbol{C}}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}}^r(t) + \hat{\boldsymbol{E}}(\mathbf{q}, \dot{\mathbf{q}}) - \boldsymbol{K}_D(t)\dot{\tilde{\mathbf{q}}}(t) - \boldsymbol{K}_D(t)\,\mathbf{\Lambda}\tilde{\mathbf{q}}(t) \tag{4.3}$$

where $\dot{\mathbf{q}}^r(t) = \dot{\mathbf{q}}^m(t) - \mathbf{\Lambda}\tilde{\mathbf{q}}(t)$, and both $\boldsymbol{K}_D$ and $\mathbf{\Lambda}$ are positive definite matrices [49].

As discussed earlier, the stiffness of the muscles vary. This controller can employ time-varying $\boldsymbol{K}_D$, but because there is currently no analytical formulation of how the muscle stiffness changes, $\boldsymbol{K}_D$ is kept as a constant matrix at this time of research.

Let $\boldsymbol{\tau}^o$ denote the control law obtained using the exact knowledge of the matrices, $\boldsymbol{H}$, $\boldsymbol{C}$, and $\boldsymbol{E}$, $\boldsymbol{\tau} = \boldsymbol{\tau}^o(\mathbf{q}, \dot{\mathbf{q}}, t)$. Then the resulting closed loop dynamics

can be expressed in the form

$$\boldsymbol{H}\dot{\mathbf{s}} = -\boldsymbol{K}_D\mathbf{s} - \boldsymbol{C}\mathbf{s} + \tilde{\boldsymbol{\tau}}(\mathbf{q}, \dot{\mathbf{q}}, t) \tag{4.4}$$

where error torque, $\tilde{\boldsymbol{\tau}} = \boldsymbol{\tau} - \boldsymbol{\tau}^o = 0$ is the discrepancy between the current control strategy and the optimal $\boldsymbol{\tau}^o$. Then, there exists a positive definite energy function $V(\mathbf{s}, t) = \mathbf{s}^T H(\mathbf{q}(t))\mathbf{s}/2$, whose derivative satisfies (4.5).

$$\dot{V}(\mathbf{s}, t) = -\mathbf{s}^T \boldsymbol{K}_D(t)\,\mathbf{s} + \mathbf{s}^T\tilde{\boldsymbol{\tau}} + \mathbf{s}^T(\dot{\boldsymbol{H}} - 2\boldsymbol{C})\mathbf{s}/2, \tag{4.5}$$

There is always an expression of $\boldsymbol{C}$ so that $\dot{\boldsymbol{H}} - 2\boldsymbol{C}$ is skew symmetric, hence the last term in (4.5) is zero, and thus, the energy function satisfies the dissipation inequality,

$$\dot{V}(\mathbf{s}, t) \leq -k_D\|\mathbf{s}\|^2 + \mathbf{s}^T\tilde{\boldsymbol{\tau}} \tag{4.6}$$

where $k_D \geq 0$, because it is the uniform lower bound on the eigenvalues of matrix K_D. Above equation, (4.6), states that the mapping from $\tilde{\boldsymbol{\tau}}$ to s is passive. Note that if $\boldsymbol{\tau} = \boldsymbol{\tau}^o$, so that $\tilde{\boldsymbol{\tau}} = 0$, (4.6) implies that $s(t)$ converges exponentially to 0, and hence $\tilde{q}$ converges exponentially to zero.

4.2 Adaptive Arm Control

Adaptive control strategies from the field of robotics offer possible insight into the successful adaptation of the internal models. In adaptive robot applications, if enough prior knowledge about the physical dynamics are available, the nonlinear components of the controller, $\boldsymbol{H}, \boldsymbol{C}$, and $\boldsymbol{E}$, can be factored into a matrix of functions, where the exact structure of the nonlinear terms are known a priori, and a vector of unknown constants, $\mathbf{a}$:

$$\begin{aligned} \boldsymbol{\tau}^{nl}(\mathbf{q}, \dot{\mathbf{q}}, t) &= \boldsymbol{H}(\mathbf{q})\ddot{\mathbf{q}}^r(t) + \boldsymbol{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}^r(t) + (\mathbf{q}, \dot{\mathbf{q}}) \\ &= \boldsymbol{Y}(\mathbf{q}, \dot{\mathbf{q}}, t)\mathbf{a}. \end{aligned} \tag{4.7}$$

An adaptive controller can then estimate the nonlinear terms, $\boldsymbol{H}, \boldsymbol{C}$, and $\boldsymbol{E}$, by changing its estimates of the constants $\mathbf{a}$. Substituting equation (4.7) for the nonlinear terms, the control law becomes (4.8).

$$\boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}}, t) = -\boldsymbol{K}_D(t)\mathbf{s}(t) + \boldsymbol{Y}(\mathbf{q}, \dot{\mathbf{q}}, t)\hat{\mathbf{a}}(t), \tag{4.8}$$

The control law uses the estimate of the nonlinear physical parameters in place of the true value. The controller stably adjusts the unknown physical parameters in real-time using the adaptation law (4.9) which expresses the mechanism of adaptation.

$$\dot{\hat{\mathbf{a}}}(t) = -\boldsymbol{\Gamma}\,\boldsymbol{Y}^T(\mathbf{q}, \dot{\mathbf{q}}, t)\,\mathbf{s}(t), \tag{4.9}$$

where $\boldsymbol{\Gamma}$ is a symmetric, positive definite matrix controlling the rate of learning. It has been shown that this strategy ensures that s approaches 0 asymptotically, therefore $\tilde{q}$ approaches 0 asymptotically [49]. This control law, coupled with the continuous, real-time adaptation strategy, results in globally stable operation, while achieving asymptotically perfect tracking of any sufficiently smooth desired trajectory.

Although this formulation is mathematically elegant, it is not likely that the human body is equipped with specific sets of nonlinear functions to utilize in building its internal models. This controller is also limited to physical system which is well expressed mathematically, but if there is a drastic change in the system, enough to change the structure of the nonlinearities in the matrix of functions, $\boldsymbol{Y}$, the controller no longer can guarantee stability or convergence.

4.3 Adaptive Arm Control and 'Neural' Networks

Consider a different factorization of the nonlinear terms which is more promising as a model of a biological system and is less sensitive to drastic changes in the structure of the physical system. The controller recently proposed by [42] instead parameterizes the nonlinear terms as multiplying a matrix of unknown functions by a known vector of signals:

$$\begin{aligned} \boldsymbol{\tau}^{nl}(\mathbf{q}, \dot{\mathbf{q}}, t) &= \boldsymbol{H}(\mathbf{q})\ddot{\mathbf{q}}^r(t) + \boldsymbol{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}^r(t) + \boldsymbol{E}(\mathbf{q}, \dot{\mathbf{q}}) \\ &= \boldsymbol{M}(\mathbf{q}, \dot{\mathbf{q}})\, \mathbf{v}(\mathbf{q}, \dot{\mathbf{q}}, t) \end{aligned}$$

or, in components,

$$\tau_i^{nl}(\mathbf{q}, \dot{\mathbf{q}}, t) = \sum_{j=1}^{2n+1} M_{i,j}(\mathbf{q}, \dot{\mathbf{q}})\, v_j(\mathbf{q}, \dot{\mathbf{q}}, t)$$

where $v_l = \ddot{q}_l^r$, $v_{l+n} = \dot{q}_l^r$, for $l = 1 \ldots n$, and $v_{2n+1} = 1$. Here control is determined by the estimates of unknown *functions* M_{ij} rather than estimates of constants.

The convergent force field hypothesis of the biological motor system presented in the previous chapter suggests a similar model of how the required motor control torques are constructed. Recall from chapter 3, that the experiments indicate that compensating torques generated are pieced together from low-level elementary structures collectively called motor computational elements. The biological evidence of linear superposition of these motor computational elements suggest that the adaptive nonlinear component is expressed as (4.10). The motor computational element approximate the necessary functions by piecing together the simple basis functions φ_k.

$$\hat{\tau}_i^{nl}(\mathbf{q}, \dot{\mathbf{q}}, t) = \sum_{k=1}^{N} \hat{\alpha}_{i,k}(t)\, \varphi_k(\mathbf{q}, \dot{\mathbf{q}}, t). \tag{4.10}$$

where $\hat{\tau}_i^{nl}(\mathbf{q}, \dot{\mathbf{q}}, t)$ is an adaptive estimate of the nonlinear torque required about the ith joint given the current state of the limb and the desired motion. Each φ_k represents a torque produced by a motor computational element, and the $\hat{\alpha}_{i,k}(t)$ represent the relative strength of each elementary torque at time t. To model biologically, the learning mechanism of these components uses a Hebbian rule of the form:

$$\dot{\hat{\alpha}}_{i,k}(t) = -\gamma\, \varphi_k(\mathbf{q}(t), \dot{\mathbf{q}}(t), t)\, s_i(t),$$

where the rate of learning is controlled by γ which is a constant. In this scheme, each weight $\hat{\alpha}_{i,k}$ adjusts in time according to the performance of elementary torque output, φ_k, to minimize the tracking error measure s_i to an appropriate level.

4.4 Motor computational elements

In this section, the mathematical description of the motor computational elements is discussed. For various computational models there is an expansion which satisfy the condition

$$\left| M_{i,j}(\mathbf{q}, \dot{\mathbf{q}}) - \sum_{k=1}^{N} c_{i,j,k}\, g_k(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\xi}_k) \right| \leq \epsilon_{i,j}$$

for any $(q, \dot{q})$ contained in a prespecified compact set $A \subset \mathcal{R}^{2n}$. An excellent approximation of the matrix $\boldsymbol{M}$ can be constructed by this expansion using a single hidden layer 'neural' network with $(q, \dot{q})$ as inputs to the network. The approximation accuracy is represented by $\epsilon_{i,j}$. g_k is the signal processing element at a node k, c_{ijk} is the output weights, and $\boldsymbol{\xi}_k$ is the input weights associated with node k. This expansion can approximate arbitrary functions over a compact set to any specified accuracy as long as the function being approximated is continuous in $(q, \dot{q})$.

where $\hat{\tau}_i^{nl}(\mathbf{q}, \dot{\mathbf{q}}, t)$ is an adaptive estimate of the nonlinear torque required about the ith joint given the current state of the limb and the desired motion. Each φ_k represents a torque produced by a motor computational element, and the $\hat{\alpha}_{i,k}(t)$ represent the relative strength of each elementary torque at time t. To model biologically, the learning mechanism of these components uses a Hebbian rule of the form:

$$\dot{\hat{\alpha}}_{i,k}(t) = -\gamma\, \varphi_k(\mathbf{q}(t), \dot{\mathbf{q}}(t), t)\, s_i(t),$$

where the rate of learning is controlled by γ which is a constant. In this scheme, each weight $\hat{\alpha}_{i,k}$ adjusts in time according to the performance of elementary torque output, φ_k, to minimize the tracking error measure s_i to an appropriate level.

4.4 Motor computational elements

In this section, the mathematical description of the motor computational elements is discussed. For various computational models there is an expansion which satisfy the condition

$$\left| M_{i,j}(\mathbf{q}, \dot{\mathbf{q}}) - \sum_{k=1}^{N} c_{i,j,k}\, g_k(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\xi}_k) \right| \leq \epsilon_{i,j}$$

for any $(q, \dot{q})$ contained in a prespecified compact set $A \subset \mathcal{R}^{2n}$. An excellent approximation of the matrix $\boldsymbol{M}$ can be constructed by this expansion using a single hidden layer 'neural' network with $(q, \dot{q})$ as inputs to the network. The approximation accuracy is represented by $\epsilon_{i,j}$. g_k is the signal processing element at a node k, c_{ijk} is the output weights, and $\boldsymbol{\xi}_k$ is the input weights associated with node k. This expansion can approximate arbitrary functions over a compact set to any specified accuracy as long as the function being approximated is continuous in $(q, \dot{q})$.

There is an algorithm developed by [43] which provides a precise specification of N and $\boldsymbol{\xi}_k$ based on the smoothness of the function being approximating. For example, take a radial basis function: $g_k(\mathbf{x}, \boldsymbol{\xi}_k) = g(\|h\mathbf{x} - \boldsymbol{\xi}_k\|)$ where h is a positive scaling parameter. The value of $\boldsymbol{\xi}_k$ is selected to encode uniform mesh over the compact set, A. From this analysis, to approximate the functions M_{ij}, only the values of $c_{i,j,k}$ need to be learned.

The network size increases with the number of independent variables in the functions to be approximated, M_{ij}. Thus, network size can be reduced by implementing prior knowledge of the matrix $\boldsymbol{M}$ as much as possible [42]. Consider for example, decomposing the term

$$\boldsymbol{C}(\mathbf{q}, \dot{\mathbf{q}})\, \dot{\mathbf{q}}^r = \boldsymbol{C}_1(\mathbf{q})\, [\dot{\mathbf{q}}\, \dot{\mathbf{q}}^r]$$

where $\boldsymbol{C}_1(\mathbf{q}) \in \mathcal{R}^{nxn^2}$, and $[\dot{\mathbf{q}}\, \dot{\mathbf{q}}^r] \in \mathcal{R}^{n^2}$ contains all possible combinations $\dot{q}_i\, \dot{q}_j^r$, for $i, j = 1, \ldots, n$. Similarly the component of the environmental force can be written as $\boldsymbol{E}(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{E}_1(\mathbf{q})\mathbf{p}(\dot{\mathbf{q}})$ where $\boldsymbol{E}_1(\mathbf{q}) \in \mathcal{R}^{nxn}$ and $\mathbf{p}(\dot{\mathbf{q}}) \in \mathcal{R}^n$ represent an assumed known $\dot{\mathbf{q}}$ dependence. Then, the nonlinear torque component can be decomposed in the form

$$\boldsymbol{\tau}^{nl}(\mathbf{q}, \dot{\mathbf{q}}, t) = \boldsymbol{N}(\mathbf{q})\, \mathbf{w}(\mathbf{q}, \dot{\mathbf{q}}, t)$$

where $\mathbf{w} \in \mathcal{R}^{n(n+2)}$ now contains the elements of $\ddot{\mathbf{q}}^r$, $[\dot{\mathbf{q}}\, \dot{\mathbf{q}}^r]$, and $\mathbf{p}(\dot{\mathbf{q}})$. Then, assume the functions required for each component of $\boldsymbol{\tau}^{nl}$ are sufficiently smooth. Then a neural network approximation of the form:

$$\tau_i^{\mathcal{N}}(\mathbf{q}, \dot{\mathbf{q}}, t) = \sum_{j=1}^{n(n+2)} \sum_{k=1}^{N} c_{i,j,k}\, g_k(\mathbf{q}, \boldsymbol{\xi}_k)\, w_j(\mathbf{q}, \dot{\mathbf{q}}, t) \tag{4.11}$$

can accurately approximate the required nonlinear control input for appropriate values of the network parameters N, $\boldsymbol{\xi}_k$, and $c_{i,j,k}$. In fact, by defining the difference between nonlinear torque and the network approximation of the nonlinear

torque, $\mathbf{d} = \boldsymbol{\tau}^{nl} - \boldsymbol{\tau}^{\mathcal{N}}$ one has

$$|d_i(\mathbf{q}, \dot{\mathbf{q}}, t)| \leq \sum_{j=1}^{n(n+2)} \epsilon_{i,j} |w_j(\mathbf{q}, \dot{\mathbf{q}}, t)|$$

for any inputs, joint angles and velocity of the joint angles in a bounded range, where each $\epsilon_{i,j}$ is now the worst case network approximation error to the components of $\boldsymbol{N}$. Since the smooth, minimum jerk cartesian paths produce correspondingly smooth desired joint trajectories, $\|\mathbf{d}\|$ is uniformly bounded provided $(\mathbf{q}, \dot{\mathbf{q}})$ can be guaranteed to remain bounded. Furthermore, this bound can be made arbitrarily small by increasing the size of the approximating network [42].

Combining the above insights, a control with plausible representation of each weighted motor computational element is created.

$$\alpha_{i,k}\varphi_k(\mathbf{q}, \dot{\mathbf{q}}, t) = g_k(\mathbf{q}, \boldsymbol{\xi}_k) \sum_{j=1}^{n(n+2)} c_{i,j,k}\, w_j(\mathbf{q}, \dot{\mathbf{q}}, t).$$

4.4.1 Adapting the Motor Computational Elements

The adaptive controller with neural network approximation of the nonlinear torque is expressed as (4.12)

$$\boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}}, t) = -\boldsymbol{K}_D(t)\mathbf{s}(t) + \hat{\boldsymbol{\tau}}^{\mathcal{N}}(\mathbf{q}, \dot{\mathbf{q}}, t) \tag{4.12}$$

where a network approximation is constructed from a weighted basis function:

$$\hat{\tau}_i^{\mathcal{N}}(\mathbf{q}, \dot{\mathbf{q}}, t) = \sum_{j=1}^{n(n+2)} \sum_{k=1}^{N} \hat{c}_{i,j,k}(t)\, g_k(\mathbf{q}, \boldsymbol{\xi}_k)\, w_j(\mathbf{q}, \dot{\mathbf{q}}, t) \tag{4.13}$$

which again uses estimates of the required parameters in place of the (assumed unknown) actual values. Use of (4.12) with dynamics (4.1) produces the closed-loop dynamics

$$\boldsymbol{H}\dot{\mathbf{s}} = -\boldsymbol{K}_D\mathbf{s} - \boldsymbol{C}\mathbf{s} + \boldsymbol{Y}^{\mathcal{N}}\tilde{\mathbf{c}} + \mathbf{d}$$

where the elements of $\tilde{\mathbf{c}}(t)$ are of the form $\hat{c}_{i,j,k}(t) - c_{i,j,k}$, and the elements of $\mathbf{Y}^{\mathcal{N}} \in \mathcal{R}^{nxN(n^2+2n)}$ are the corresponding combinations $g_k(\mathbf{q}, \boldsymbol{\xi}_k)\, w_j(\mathbf{q}, \dot{\mathbf{q}}, t)$.

Compare this closed-loop dynamics with the closed-loop dynamics of the adaptive robot algorithm. It is the same except for the disturbance term d, which describes the discrepancy between the actual value of the required nonlinear torque and the network approximation of the nonlinear torque. The controller of (4.13) can only provide the locally approximate parameterization of the nonlinear torque. Thus a careful treatment of the design of the adaptation is essential.

Provided that the joint angles and the joint velocities remain within their operating range, A, the disturbance term d can be accommodated for by a couple of *robust* adaptation methods. For one, a weight decay term can be added to the adaptive algorithm. The adaptive algorithm becomes

$$\dot{\hat{\mathbf{c}}}_{i,j,k}(t) = -\gamma \left(\omega(t)\, \hat{c}_{i,j,k}(t) + s_i(t)\, w_j(\mathbf{q}, \dot{\mathbf{q}}, t)\, g_k(\mathbf{q}(t), \boldsymbol{\xi}_k) \right), \tag{4.14}$$

where $\omega(t) = 0$ if $\|\hat{\mathbf{c}}(t)\| < c_0$, and $\omega(t) = \omega_0 > 0$ otherwise. This adaptation law sets an upper bound, c_0, on the total magnitude of the parameters required to accurately approximate $\boldsymbol{\tau}^{nl}$. The parameter γ is a positive constant, learning rate; it can be different for each weight.

Another form of stable adaptation law may be more biological. It allows each parameter value to saturate using a projection algorithm of the form:

$$\dot{\hat{\mathbf{c}}}_{i,j,k}(t) = \mathcal{P}\left(-\gamma s_i(t)\, w_j(\mathbf{q}, \dot{\mathbf{q}}, t)\, g_k(\mathbf{q}(t), \boldsymbol{\xi}_k), \hat{c}_{i,j,k}(t), c_{max} \right) \tag{4.15}$$

where $\mathcal{P}(x, y, z) = x$ if $-z < y < z$, or if $y \leq -z$ and $x > 0$, or if $y \geq z$ and $x < 0$; $\mathcal{P}(x, y, z) = 0$ otherwise. Since the neural networks is only able to approximate the nonlinear torque to a specified, bounded accuracy, the projection is required to stop the adaptation when the performance reaches the appropriate accuracy.

Human arm motion, due to its physical limitations, impose that the joint state variables remain within an easily computed nominal range, A. For more general case of nonlinear control, a special modification to the above control law is necessary, in case the state variable leave the compact set, A. The 'supervisory' controller which force the state variable back to the nominal range can be added to accommodate for such a situation. In fact, the physical limitations of the human arm can be viewed as a type of 'supervisory' control imposed on the arm motion. Additionally, the pain sensation caused by the hyperextension of the arm can be viewed as a command from the 'supervisory' controller.

This combination of 'neural' approximation, real-time adaptation, and 'supervisory' action (for some necessary case) can be proven to remain stable and to asymptotically converge to a small neighborhood of the desired trajectory [42], [43]. Explicitly, the convergence can be expressed as

$$\lim_{T\to\infty} \frac{1}{T} \int_0^T \|\tilde{\mathbf{q}}(t)\|^2 dt \leq \frac{\epsilon}{k_D^2 \lambda^2}. \tag{4.16}$$

where

$$\epsilon \triangleq \sup_t \sup_{\mathbf{x}\in A} \sum_{i=1}^{n} |d_i(\mathbf{q}, \dot{\mathbf{q}}, t)|^2.$$

and λ is the smallest eigenvalue of $\mathbf{\Lambda}$. This equation of the convergence capabilities of the algorithm captures how the two methods by which controller may deal with the disturbance force field. The controller can implement torques which effectively stiffen the muscle, which is represented by increasing the term $K_D\lambda$, or else the controller can improve its model based estimates by minimizing ϵ which is the representation of how good the approximation by the internal model is.

4.5 Simulation

4.5.1 Simulation Construction

A simulation of two-joint arm motion was constructed using the adaptive motor control model presented in the previous sections. The simulation was constructed to compare the mathematical prediction of the above model to the experimental results of Shadmehr and Mussa-Ivaldi study [46].

A model of two joint arm motion (actuation occurring at the elbow joint and the shoulder joint) was simulated. Point-to-point reaching motion of length 10 cm was conducted in a time duration of 0.65 sec. The simulation assumes that the human reaching motion is dictated by 'minimum jerk trajectory'. A pseudo-random target is presented within the workspace of 15 cm by 15 cm. The target is presented in one of eight possible directions, equally separated from each other by 45^o. The reaching task is updated every 1.3 seconds, 0.65 seconds for the execution of reaching motion and 0.65 seconds to hold. The hold and wait phase is there because the disturbance force field deviate the hand trajectory, and the motion will take longer to execute. Initially, the controller is given 'perfect' self knowledge and given no knowledge of the environmental force field, hence $\hat{\boldsymbol{E}} = 0$. The simulation assumes that the test subject is an adult who had learned the arm properties in his/her lifetime of practicing and learning motion. Although the controller can learn the other terms, the simulation assumes that the CNS only needs to learn the environmental force acting on the human motion. The control laws described in the previous section, (4.13) and (4.15) were used to model the neural computation of the necessary joint torques. The gain matrices from [46] was used in conjunction to the control laws:

$$\boldsymbol{K}_D = \begin{bmatrix} 2.3 & 0.9 \\ 0.9 & 2.4 \end{bmatrix} \quad \boldsymbol{\Lambda} = \begin{bmatrix} 6.5 & 0.064 \\ 0.064 & 6.67 \end{bmatrix}$$

4.5.2 Network Design

Radial basis function network with a fixed width nodes specified by $h > 0$ and a fixed range of translations, $\mathbf{k}$, was used to learn the nonlinear component of the dynamics equation [44] 4.17.

$$g_{\mathbf{k}}(\mathbf{q}) = g(h\mathbf{q} - \mathbf{k}) \tag{4.17}$$

The capacity of the network approximation of a smooth function is proportional to h^{-r} [41] where r is the rate of convergence of the approximation. The convergence rate varies according to the network size, smoothness of the function that is being approximated, and the type of basis function used. This simulation uses Gaussian basis function. The domain of the good approximation was selected to include the range of joint angles exercised during the experiment. The value of parameter h is set to 2, this makes the learning broad enough to generalize the learning to occur within the range of set A which is $A = [-.5, 2]$ x $[.5, 2.5]$. The larger value of h will give better approximation, but will require larger network size, and decreases the generalization capabilities. The optimization of h or the basis function choice is beyond the scope of this thesis. The network equation using radial basis function is given in equation (4.18).

$$\hat{\tau}_i^{\mathcal{N}}(\mathbf{q}, \dot{\mathbf{q}}, t) = \sum_{j=1}^{8} \sum_{\mathbf{k} \in \mathcal{K}} \hat{c}_{i,j,\mathbf{k}}(t) g(h\mathbf{q} - \mathbf{k})\, w_j(\mathbf{q}, \dot{\mathbf{q}}, t) \tag{4.18}$$

where

$$\mathbf{w}^T = [\ddot{q}_1^r,\ \ddot{q}_2^r,\ \dot{q}_1\dot{q}_1^r,\ \dot{q}_1\dot{q}_2^r,\ \dot{q}_2\dot{q}_1^r,\ \dot{q}_2\dot{q}_2^r,\ \dot{q}_1,\ \dot{q}_2].$$

The learning rate for each j was tuned to match the learning rate observed in the experiments with human subjects: after some experimentation, the values $\gamma_j = 0.01$ for $j = 1, \ldots, 6$ and $\gamma_j = 0.04$ for $j = 7, 8$ were used in the simulation. The upper bound on the network weights, $\hat{c}$ was set at $c_{max} = 75$.

The force field used in the simulation was

$$E(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{J}^T(\mathbf{q})\, \boldsymbol{B}\boldsymbol{J}(\mathbf{q})\, \dot{\mathbf{q}} \tag{4.19}$$

where

$$B = \begin{bmatrix} -10.1 & -11.2 \\ -11.2 & 11.1 \end{bmatrix}.$$

and $\boldsymbol{J}$ is the Jacobian of the two-link arm, which maps the joint angle velocity to the cartesian velocity. This force field is the same one used in the experiment by Shadmehr and Mussa-Ivaldi [46].

The reaching motion in 'null field' exhibit perfect tracking, as seen in Figure 4.1.

4.5.3 Controller Model Performance

When the force field described by the equation 4.19, the simulation result show the deviation from the reaching path in a 'hook' shape. Figure 4.2 shows the reaching path at the initial exposure to the force field.

These reaching performance agree very well with the human result reported in [46]. The reaching motion executed in a well known environment exhibit very straight motion to the target location in human data, while the reaching motion executed at the initial exposure to the force field is characterized by the hook pattern oriented in the same direction.

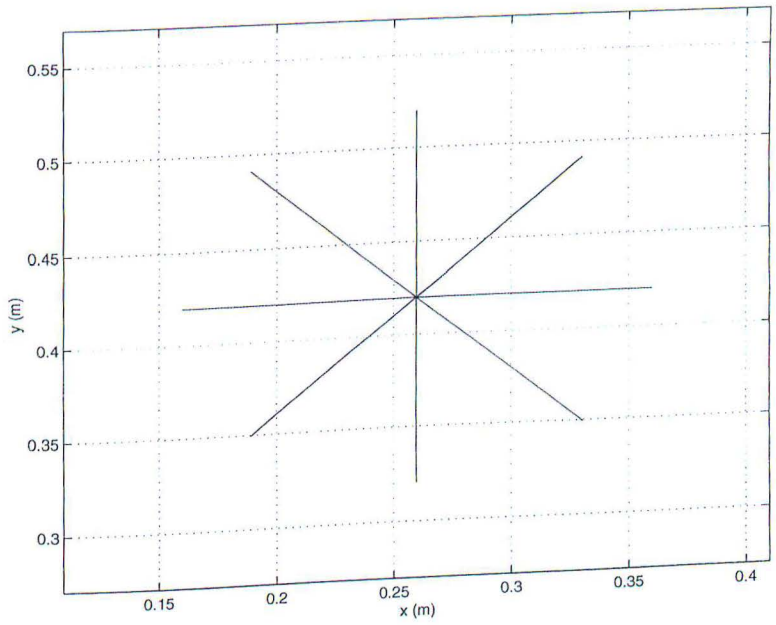

Figure 4.1: Evaluation of reaching motion under 'null field'

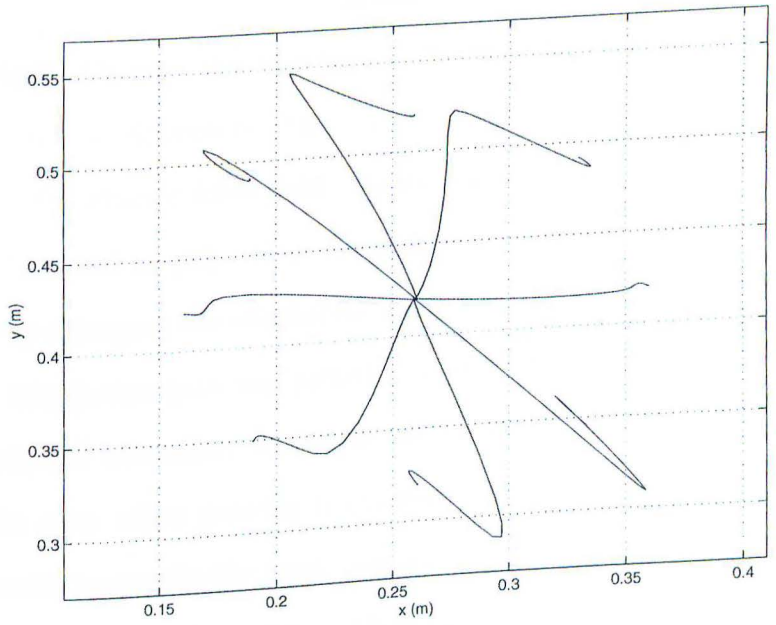

Figure 4.2: Evaluation of reaching performance at an initial exposure to the force field

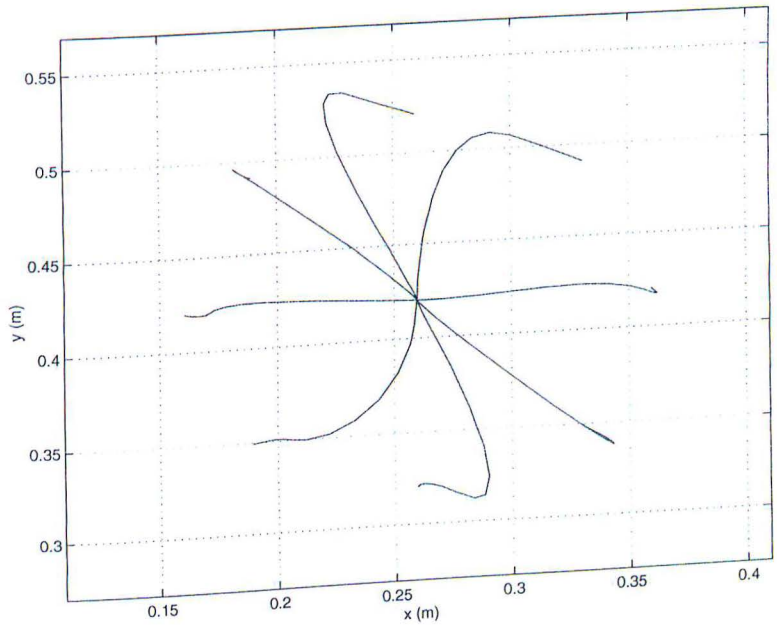

Figure 4.3: Reaching motion executed after 250 cycles

The Figures 4.3 to 4.6 show the progression of the performance under disturbance force field. Clearly, the learning of the force field is seen in these plots. The reaching motion approaches the baseline performance as the motion is executed in the disturbance force field. Again, these results correlate well to the human data reported in [46]. Note again that the control model can have any learning rate by changing the adaptation gain. For the purpose of the simulation, however, the adaptation gain was picked to match the human data.

The evidence of aftereffect is seen in the Figures 4.7 to 4.10. The time progression of the after effect growing is evident from these figures. The signature of internal model is seen as the hook pattern in the opposite direction from the initial performance in the force field. The appearance and the time progression of growing aftereffect is very similar to the human data reported in [46].

Thus, it appears this is mathematical model of the adaptive motor control

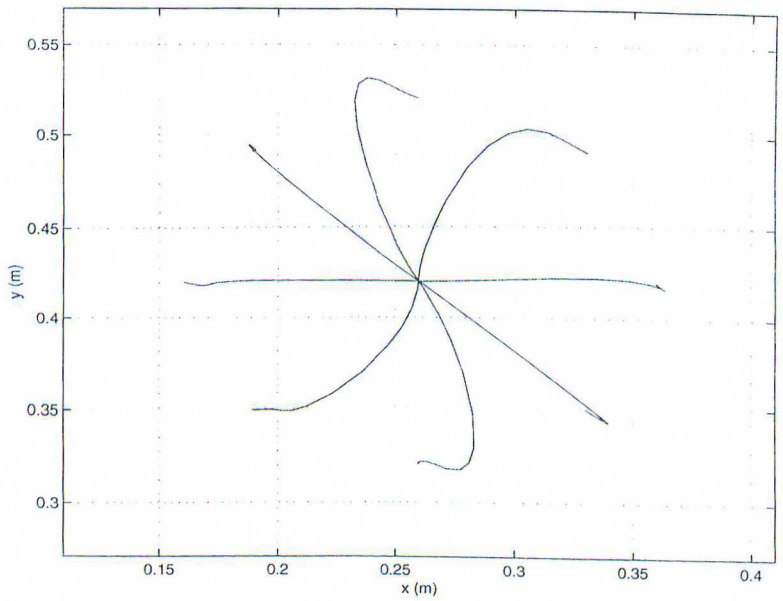

Figure 4.4: Reaching motion executed after 500 cycles

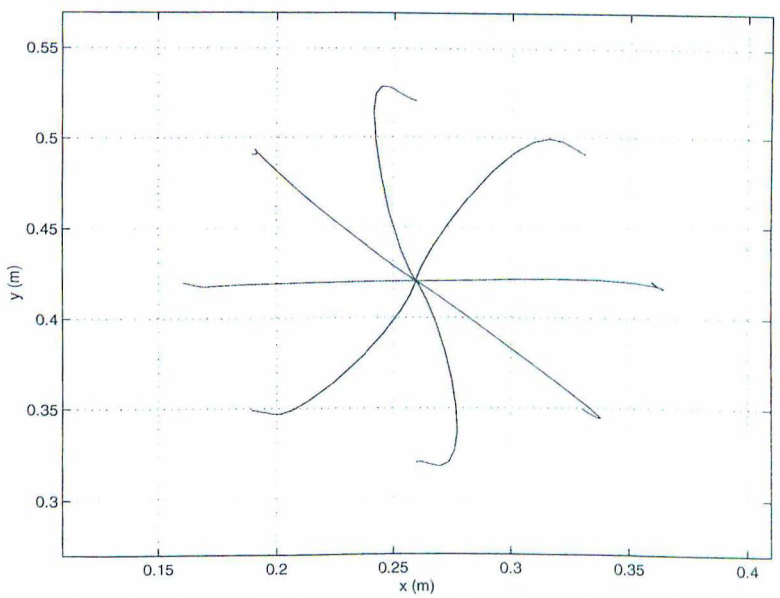

Figure 4.5: Reaching motion executed after 750 cycles

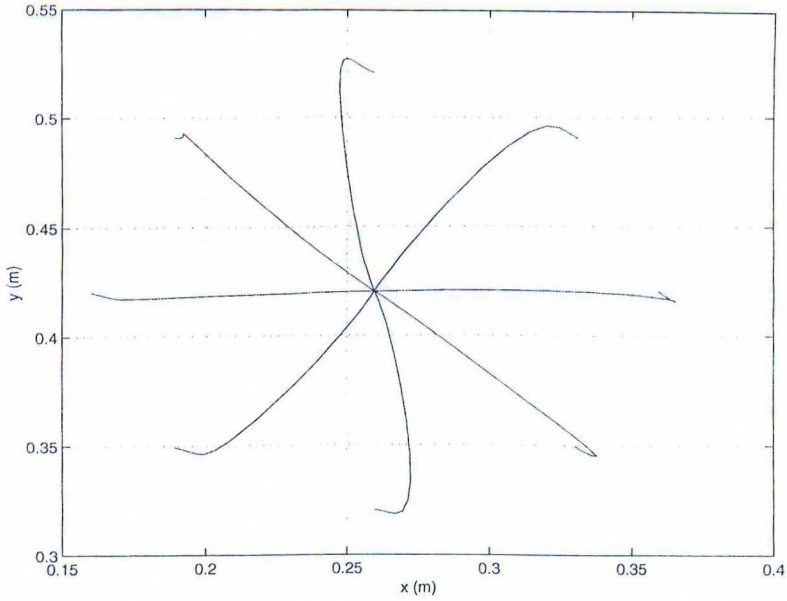

Figure 4.6: Reaching motion executed after 1000 cycles

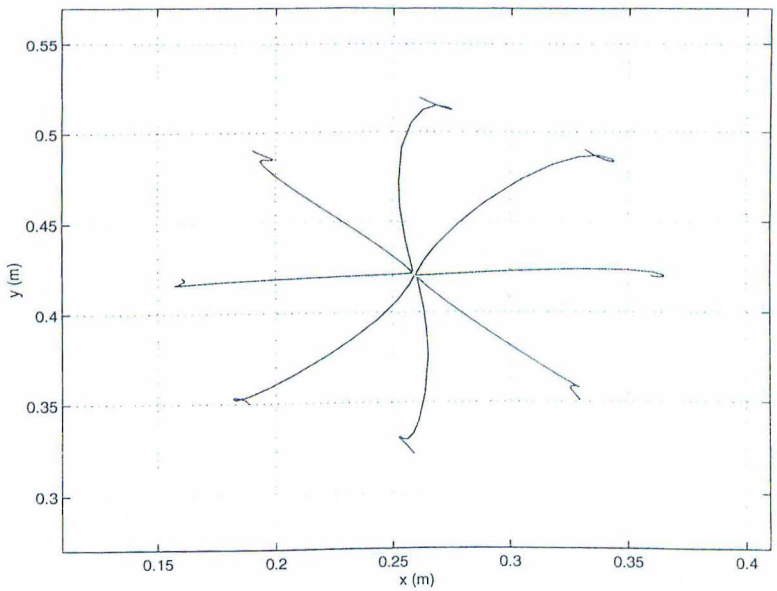

Figure 4.7: Evidence of aftereffect after 250 cycles

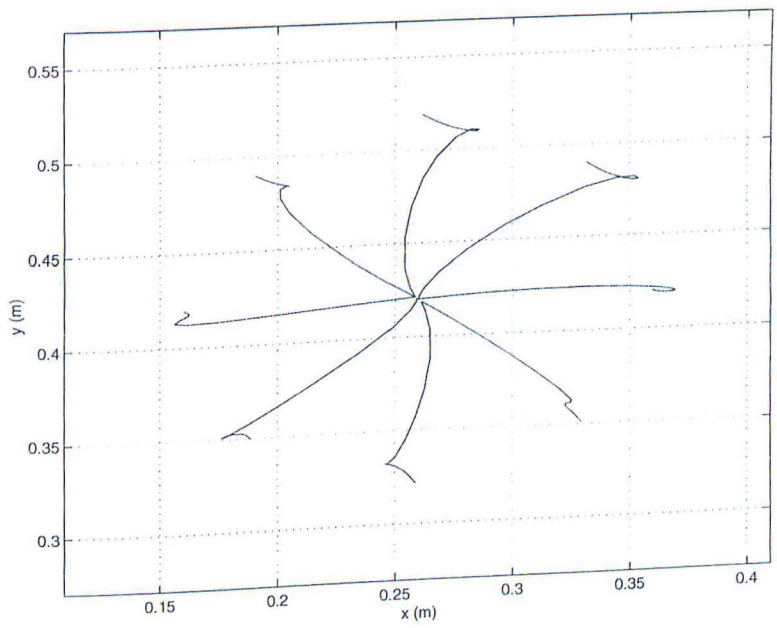

Figure 4.8: Evidence of aftereffect after 500 cycles

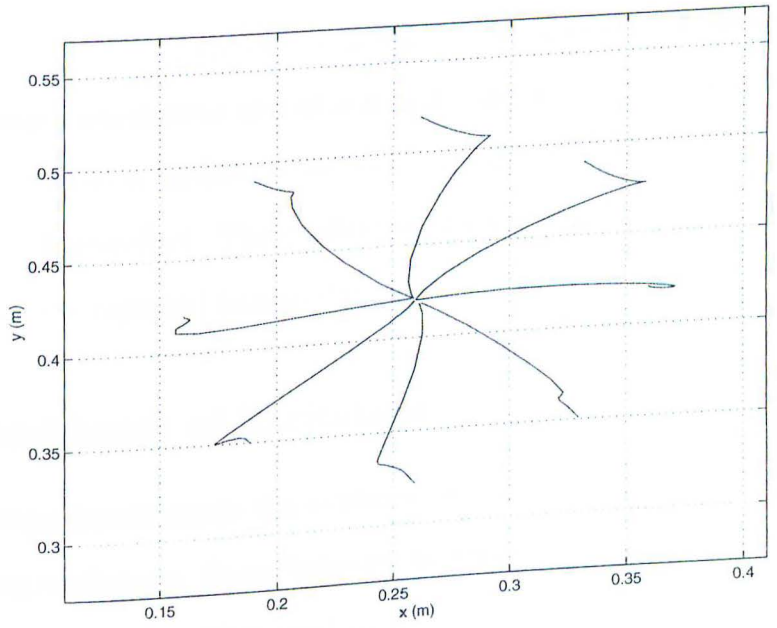

Figure 4.9: Evidence of aftereffect after 750 cycles

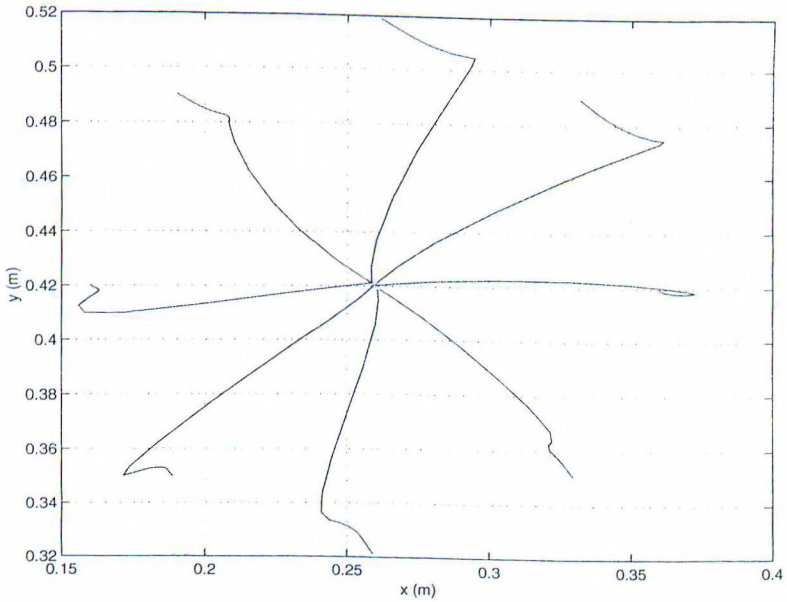

Figure 4.10: Evidence of aftereffect after 1000 cycles

is able to capture the performance seen during the execution of reaching motion in a well known environment and at a initial exposure to the disturbance force. Moreover, the model is capable of successfully capturing the time progression of the learning and aftereffect. The qualitative characteristics of the performance is very similar to the reported human data [46].

4.5.4 Persistency of Excitation

The human experiment shows the evidence of generalizable learning to outside of the learning workspace. Same behavior is difficult for this model of adaptive control. The control algorithm used in the model updates the internal model just enough to successfully track the desired trajectory. The construction of the exact model is not a necessity, unless appropriate desired trajectory is selected for learning. The desired trajectory necessary for the construction of the 'perfect'

model is known as the 'persistency of excitation'. For further presentation on this topic, refer to [49].

Chapter 5

A Haptic Device for Motor Control Studies

A haptic device capable of conducting future human motor control study was designed and built at the Space Systems Laboratory (SSL). The haptic system, named TSUNAMI (Test Stand for Upper-body Neuromuscular Adaptive Movement Investigation), was built to study the adaptive motor control of upper limb motions same as the simulation described in chapter 4. This chapter presents design considerations and the system description of TSUNAMI. To design a test apparatus capable of conducting an experiment in interesting environments such as the neutral buoyancy and the KC-135 investigating the same scenario as the simulation described previously, the following considerations must be included in the system design. First, TSUNAMI must be capable of applying variable force to the test subject as he/she reaches for a target position. The virtual force field applied to the test subject must be well characterized in order to study the response of the human arm dynamics to the applied environmental forces, and the motor control adaptation to the external forces. Secondly, TSUNAMI must be able to perform the experimental task while resisting the small vertical force applied by the test subject during testing. Thirdly, TSUNAMI must produce forces large enough to be sensed by the human subjects but small enough so that

they will not harm the test subject. It is advantageous for the system designer to design a robot arm whose force application capacity is not harmful to the subject as an additional safety precaution yet moves fast enough to accommodate the speed of the human arm motion. Fourth, the system must be designed to be portable and functional under water. For future research, it is important to be able to test human motor control in the astronaut training environments, such as in neutral buoyancy and on board KC-135 research aircraft.

5.1 Design Criteria

5.1.1 Workspace Sizing and Force Capacity

The operational requirements for this test stand include realistic workspace and force reflecting environments. The system workspace, which must exercise the normal working range of human arm joint angles (shoulder and elbow), is designed to fit the target workspace size of 13.7cm by 13.7cm (5.4 inches by 5.4 inches). The size of the target workspace will allow normal reaching motion without approaching the extremes of the test subject's joint angles. The workspace of TSUNAMI is further described in the later section.

The force capability of the system, which must be large enough to induce effective deviation in operator motions, yet not endanger the operator, has been set at a maximum value of five pounds of force throughout the work area. These constraints, determined from data obtained via computer simulation, agree with the requirements of JPL's Universal Force Reflecting Hand Controller, which operates in a twelve inch cube and provides three pounds of translational force and 5.5 inch-pounds of rotational capability [4].

The actuator has to be able to apply the specified force field to the subjects hand during reaching motions without allowing any other system dynamics to be sensed. For example, the test subject must not feel the mass and the friction of the actuators during any part of the experiment. For a manipulator that is not instrumented with a force-torque sensor, the actuator design must minimize cogging and backlash, but maximize the motors backdrive. In addition, the sensation of cogging and backlash would add uncharacterized and complicated disturbances to the force field during testing, producing unwanted disturbances during a null force field phase. The motors are there only to compensate for the system friction and to apply an external environmental force field. Additionally, the test subject must be able to backdrive the motor with very little effort. Thus, presence of virtually no manipulator dynamics is preferred.

5.1.2 Data Measurement

The manipulator joint angles and the joint velocity must be known in order to specify the appropriate tip force while compensating for the friction. To study the reaching motion data, the joint position and velocity of the test subject must be available for post-test analysis. To sufficiently characterize the human reaching motion which takes place in approximately 0.65 seconds for this experiment, the kinematic data is stored at a rate of 80 Hz. This data collection rate is in agreement with similar studies conducted by Shadmehr and Mussa-Ivaldi [46].

5.2 Adaptability and Future Considerations

Although some experiments are conducted in a typical laboratory environment, the SSL conducts most studies of human operations in the neutral buoyancy tank and few studies on board KC-135 research aircraft to simulate microgravity environment. Therefore, TSUNAMI is designed to be adaptable to conduct research under water. Hence, all electronics and motors are encased in waterproof boxes for use in the neutral buoyancy environment. The Intel 80386-based data collection system, which includes a PCMCIA data storage card, is also waterproofed and located in the vicinity of the arm to minimize noise contamination from the analog signals. The computers and the motors can be powered by a set of DC batteries for the testing in neutral buoyancy tank.

5.3 System Description

The test apparatus being described here is a two-dimensional, cable driven, parallel manipulator. The parallel manipulator configuration of TSUNAMI allows the two motors, which are instrumented with optical encoders, to be collocated and mounted off of the manipulator links. This parallel configuration enables motor placement outside of the workspace, thus reducing the inertia of the manipulator significantly. Figure 5.1 shows a picture of TSUNAMI test setup. The actuators need to be easily back drivable so that the test subject can easily move the handle to the target position. The cable driven system allows the manipulator to have a very small gear ratio without backlash or cogging. This is very important because, during the test, the unknown system dynamics need to be minimized as much as possible. In summary, this configuration minimizes the manipulator

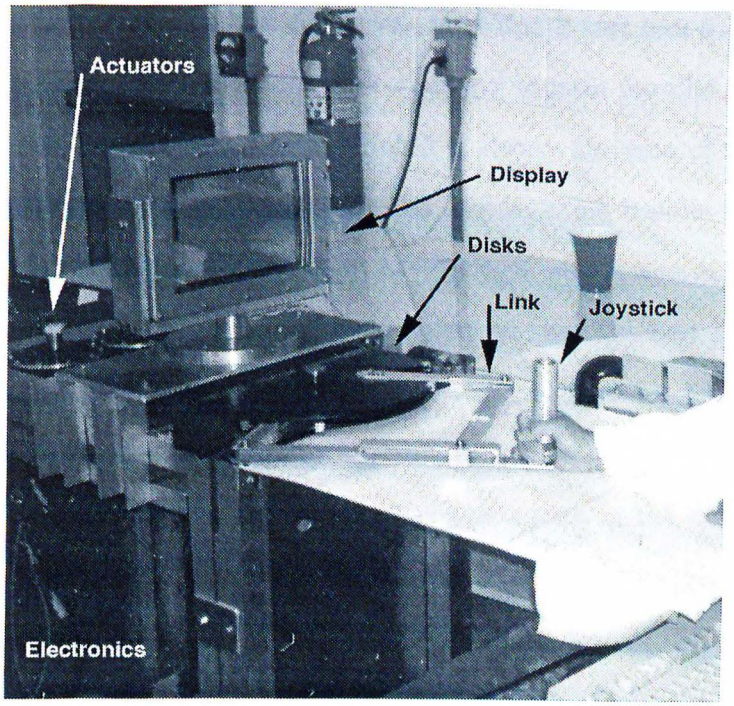

Figure 5.1: TSUNAMI test setup

inertia and dynamics so that they are negligible relative to the dynamics of the system being simulated. Hence, the system is capable of testing without being instrumented with a force-torque sensor, reducing the cost and development time.

5.4 Hardware Design: Mechanical Description

5.4.1 Parallel Manipulator

The TSUNAMI manipulator is a parallel linkage, two-dimensional system that is driven with a set of cables attached at the link base. The cable spans between the link base and the motor shaft. A sketch of the manipulator configuration is shown in Figure 5.2. Link 1 and link 3 are collocated at the disk center. And link

3 and link 2 are connected by link 4 in a way that link 2 and link 3 always remain parallel to each other, and link 1 and link 4 always remain parallel to each other. This configuration assures that angle of link 3 is always the sum of angle of link 1 and the angle between link 1 and link 2. The free rotating handle is constructed using two aluminum cylinders with a set of bearings, and it sits at the tip of the last link. The handle is designed to freely rotate so that the test subject's wrist orientation can remain constant regardless of the manipulator tip position. To meet the workspace size criteria, the manipulator link lengths were chosen by iterating for the desired workspace for various gear ratios in the desirable output torque range.

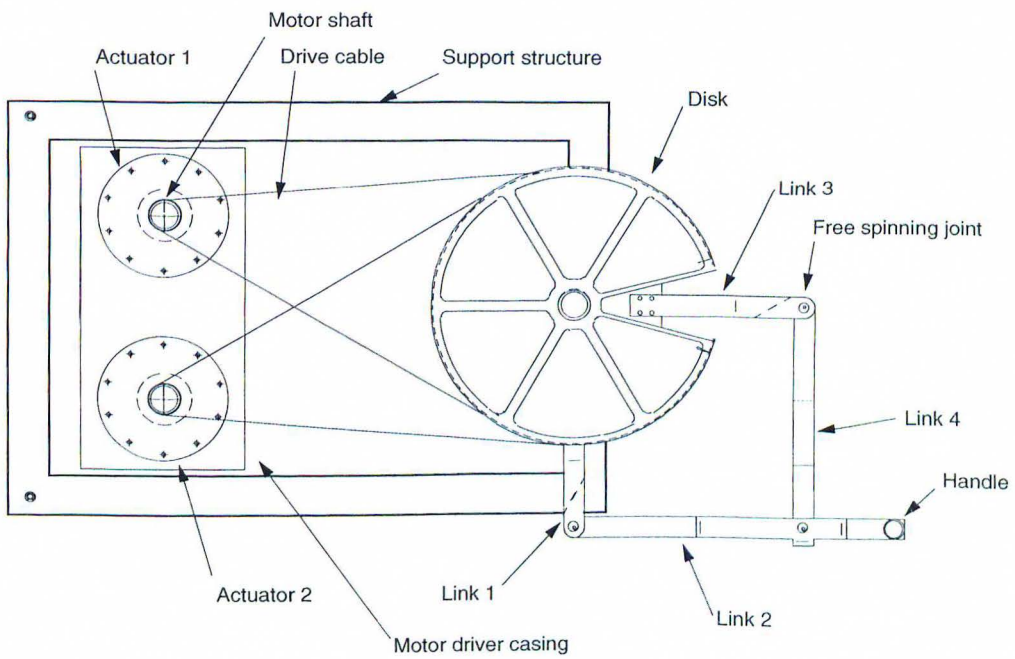

Figure 5.2: TSUNAMI top view

5.4.2 Workspace

The manipulator workspace is the area in which the links are able to reach. The targets are constrained by the target workspace, and placed within the manipulator workspace. During the experiment, the test subject may overshoot the target and reach beyond the target workspace, therefore the target workspace is constrained in a smaller space, within the area reachable by the manipulator. Furthermore, the experiment should be conducted away from the edges of the manipulator workspace, because it is more difficult to move the manipulator near the edges of the workspace. The top edges of the manipulator workspace are the singularity points for the TSUNAMI arm, and are best avoided. The lower edges do not quite reach the singularity point but are very close to the base, and therefore do not have as much moment arm to move the arm. Consequently, the target workspace should be located as far away from the top edge as possible without quite extending to the lower edge. Figure 5.3 is a plot of both areas.

5.4.3 Forward Kinematics

Forward kinematics is the calculation of the Cartesian tip position from the joint angles. The forward kinematics of the parallel, two-link manipulator is very similar to that of the serial two-link manipulator. See Figure 5.4 for the schematic of the linkages. The difference between the two manipulator configurations is the specific angle data available from the sensors mounted on actuators. The serial manipulator has the angle data of link 1 and link 2, while parallel manipulator has the angle data of link 1 and link 3. But, as can be seen in Figure 5.4, a_1 is equal to q_1 and a_2 is equal to $q_1 + q_2$. Equations (5.1) show the similarity of the

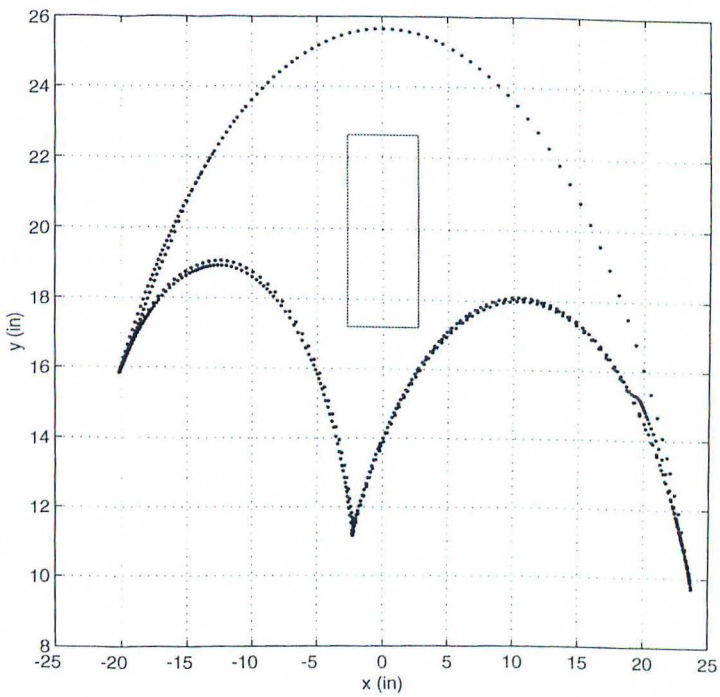

Figure 5.3: TSUNAMI's workspace

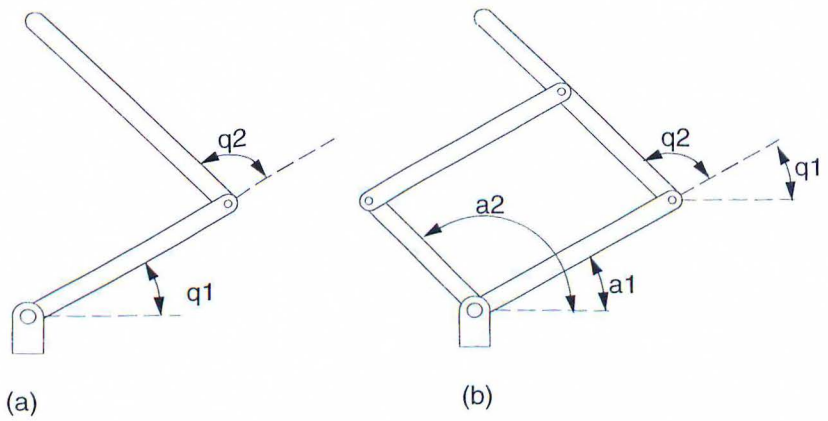

Figure 5.4: Serial Manipulator versus Parallel Manipulator

two equations.

$$
\begin{aligned}
&\textit{Serial} \\
&\qquad x = l_1 \cos(q_1) + l_2 \cos(q_1 + q_2) \\
&\qquad y = l_1 \sin(q_1) + l_2 \sin(q_1 + q_2) \\
&\textit{Parallel} \\
&\qquad x = l_1 \cos(a_1) + l_2 \cos(a_2) \\
&\qquad y = l_1 \sin(a_1) + l_2 \sin(a_2)
\end{aligned}
\tag{5.1}
$$

For further details on the discussions of forward kinematics, torque calculation, and Jacobian, see [11].

5.4.4 Jacobian

The Jacobian is a multidimensional form of the derivative. It is used to transform velocities and forces between joint space and Cartesian space. Equation (5.2) represents the Cartesian transformation of static forces from joint torques. The

velocity and force calculations are shown in (5.3) and (5.4), while (5.5) shows the Jacobian of a parallel manipulator.

$$\tau = \boldsymbol{J}^T \cdot f \tag{5.2}$$

$$\begin{pmatrix} \tau_1 \\ \tau_2 \end{pmatrix} = \begin{pmatrix} -l_1 \sin(a_1) & l_1 \cos(a_1) \\ -l_2 \sin(a_2) & l_2 \cos(a_2) \end{pmatrix} \begin{pmatrix} f_x \\ f_y \end{pmatrix} \tag{5.3}$$

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} -l_1 \sin(a_1) & -l_1 \cos(a_1) \\ l_1 \cos(a_1) & l_2 \cos(a_2) \end{pmatrix} \begin{pmatrix} \dot{a}_1 \\ \dot{a}_2 \end{pmatrix} \tag{5.4}$$

$$\boldsymbol{J} = \begin{pmatrix} -l_1 \sin(a_1) & -l_1 \cos(a_1) \\ l_1 \cos(a_1) & l_2 \cos(a_2) \end{pmatrix} \tag{5.5}$$

5.4.5 Motor Sizing

The motor was sized to not have any greater mechanical advantage than a human arm (to prevent any harm), and moves fast enough to keep up with the human reaching motion. Thus the motor was sized to accommodate a low gear ratio and still meet the force application criteria discussed above. The TSUNAMI gear ratio provided by the cable system is 15 to 1.

Inland Motor's, direct-drive DC torque motor[1] was chosen to actuate the linkages on TSUNAMI. The gear ratio and link lengths were designed specifically for these experiments. The Table 5.1 lists the specifications of the motor used for the TSUNAMI actuators. These frameless motors are encased in motor housings were fabricated which hold the stator assembly, brush ring, rotor assembly, and stainless steel drive shaft together and provide mounting points. The motor

[1]Part Number 2955A from Inland motor

Size constants	Value	Units
Peak torque rating	0.85	lb-ft
Motor constant	0.097	lb-ft/watt
Motor weight	1.5	lb
Voltage, stalled at	12.2	volts
Peak current	6.8	amperes

Table 5.1: Inland motor data sheet.

assembly of TSUANMI was covered with a delrin plate to protect mechanically sensitive parts, such as bearings and encoders, from motor brush dust.

Motor drivers, which require much more power than the above mentioned electronics, are encased in a custom water proof aluminum box connected to the motor casing, which is also separately water proofed. From each of the two motor casings, shafts to drive the cables protrude through holes with a shaft seals[2] installed to prevent water from entering the motor casing. The shaft seals are commercially available seals that allow a rotating shaft to pass through a hole while staying water-tight. The shaft seals add a small amount of additional friction to the system. Figure 5.5 shows the assembly.

The force sensing threshold of the average person is 0.25 ounces to 0.36 ounces [39]. The test stand is capable of producing over fourteen times that amount at any point within the workspace. The performance capability is listed on the Table 5.2.

[2]Part Number: 114MB124-G from Bal Seal Engineering Company

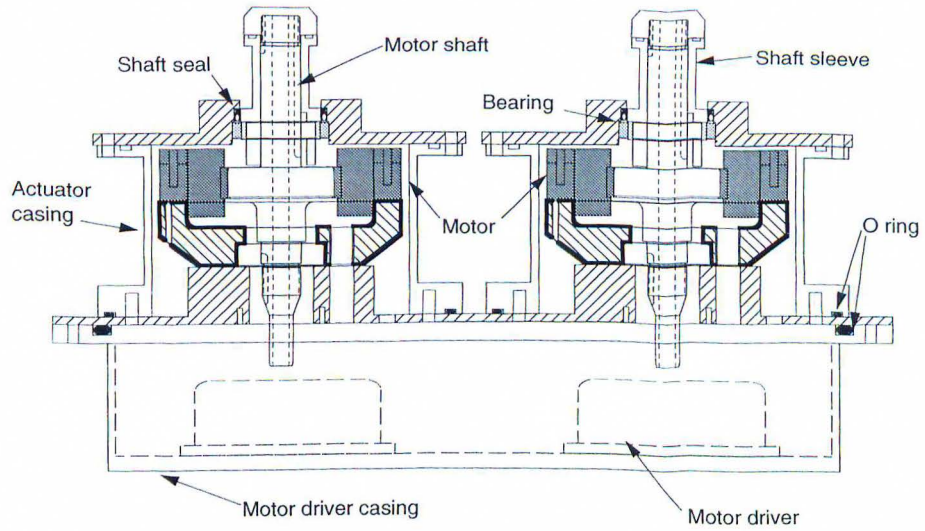

Figure 5.5: TSUNAMI actuators: side view

Performance characteristic	Value	Units
Force application frequency	125	Hz
Maximum Force at the tip	9.0	lb

Table 5.2: Performance capability of TSUNAMI

5.4.6 Support Structure

The actuator housing, electronics, video monitor, and manipulator assembly are held together by a rectangular support structure constructed from aluminum C channels. This support structure also holds a flat plate in front of the manipulator to support the test subject's elbow. The table mounted on TSUNAMI allows the subject to keep his/her testing arm at shoulder level without installing an arm harness, which could interfere with the intended motion. The arm harness is often used for tests which constrain the limb to a plane and simulate reduced gravity environments. However, these harness often constrict the active muscles,

interfering with the motion. The support structure is 34 in in height, 18 in in width, and 28 in in depth. The front and the back portions of the support frame have aluminum panels installed for additional strength, but the two sides are left open for easy access to the electronic cables and hardware installation.

The flat panel video display is also enclosed in a separate custom waterproof box. The casing for the it consists of a hollow aluminum box, an aluminum back plate, and a plexiglass front cover. The video power is supplied from a DC power supply into the casing, and the video signal runs through a cable from the video driver card in the electronics box. The video display is mounted on top of the manipulator assembly. It is situated about 36 in from the test subject.

All of the electronics necessary for driving the DC brush motor and data collection are encased in a water proof, commercially available camera housing[3]. Just as the video casing, the DC power is available remotely, and is supplied by cable into the electronics housing.

5.5 Electronics Design

5.5.1 Computer

The electronic system is based on the Ampro 80386 SX mother board[4] with math coprocessor and with 8 MB of RAM. The CPU, encoder reader card, PCMCIA interface card, video driver card, and digital to analog converter card are all connected to each other through a PC104 bus with memory locations allocated to data from each card. The 386 SX Little Board is a 25 MHz machine with dual

[3]Ikelite box, manufacturer info

[4]Ampro Computers Inc.

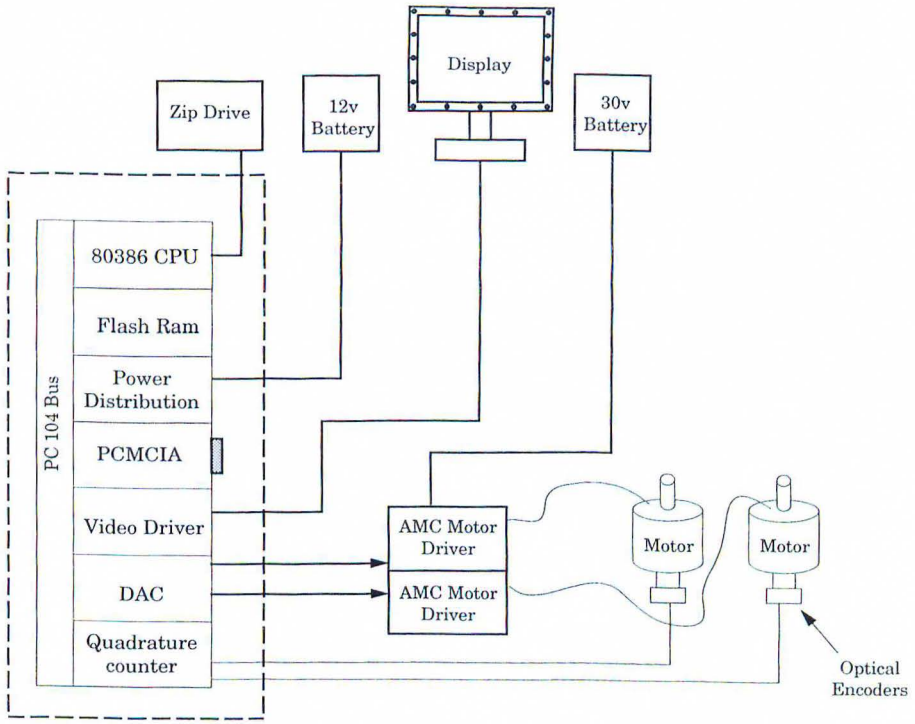

Figure 5.6: TSUNAMI computer system block diagram

serial and parallel port, floppy drive controller, IDE hard disk drive interface, SCSI interface, and stackable PC/104 expansion. The software for data collection and for running TSUNAMI is stored on the PCMCIA card. AMPRO supplies the PCMCIA interface card which is compatible to their 386 SX Little Board. The data of the force output and the manipulator tip position and velocity are stored in the PCMCIA card during testing, while ZIP drive is used for long term data storage. This PCMCIA card is not a bootable disk. To start up the computer, a 20MB IDE Flash Drive[5] is used. A sketch of the electronics is provided in Figure 5.6.

The power distribution board takes the unregulated voltage from the remotely

[5]Part Number: SDIB-20-101 from California Peripherals

located twelve volt, DC power, (either twelve volt, DC battery[6] or a power supply) and regulates to five volt computer power using a Datel DC-DC converter[7]. A DC-DC converter is used instead of a voltage regulator because it is much more efficient. Its efficiency reduces power consumption and heat generation. The DC-DC converter output has greater noise than voltage regulators, because it cycles to provide the appropriate voltage, but the electronics powered by this board are not sensitive to small ripple. The board supplies power to the electronics in the electronics housing (CPU, PCMCIA card, video driver card, DAC board, encoder reader card, and solid state disk). The power distribution board also has a five amp fuse before the DC-DC converter for current protection. The electronics powered by this board require a maximum of three amps total. Due to the limit of current output by the power distribution board, a separate power supply is used for the video display, a floppy drive, and a ZIP drive.

The encoder reader card[8] with PC 104 bus is used to convert the output from the optical encoders on the motors into digital values readable by the computer. Additionally, this encoder reader card provides power to the encoders. This card has four encoder reader channels, configured in such a way that one of the channels can be jumpered to provide a clock counter which runs at 500 kHz. The clock signal from the encoder card is used to provide a time signal for the experiment. The power requirement for the encoder card is a five volts computer power at approximately 400 milliamps.

A video driver card compatible with the PC104 bus and the PLANAR display

[6] 20 amp-hour, Power Sonic re-chargeable battery

[7] Model number: UWR-5/4000-D12

[8] Part Number: 4I30 from MESA Electronics

is available through AMPRO[9]. The video driver card is supplied with video driver software. This card also requires five volts about 400 milliamps.

To control the torque output using a computer, the computer signal must be converted to an analog input and sent to the motor driver card. The analog voltage level is then converted by the drivers to a phase width modulation (PWM) signal used to control the motors. First, the digital to analog converter card (DAC board)[10] converts the digital signal from the computer, then it supplies the appropriate analog signal to the motor driver board[11]. The DAC board is able to output either monopolar 5 volt signal or bipolar 5 volt signal. TSUNAMI is configured to output a bipolar 5 volt signal which commands the motor to spin in one direction with positive voltage and command the motor to spin in the opposite direction with negative voltage. The DAC board requires a maximum of 700 milliamps. When the operating mode of the motor driver board (also called PWM servo amplifiers) is set to a current (torque) mode, the input voltage is converted to a current output, and the driver board commands the motors by current control. This way, the output torque is directly commanded. The motor driver board has its own dedicated unregulated DC power source for the reference voltage, or the main power. By connecting a completely separate power source, the computers and the actuator system can be kept electronically isolated from each other.

[9] Ampro MiniModule/SuperVGA Display Controller

[10] Part Number AX10415 from AXIOM

[11] Part Number: 10A8 from Advanced Motion Controls

5.5.2 Sensors

The position and the velocity of the manipulator joints are calculated from the motor shaft angle, which is measured using optical encoders installed on the motor shaft[12]. The velocity of the shaft motion is calculated by taking a derivative of the shaft position information. The data is filtered to reduce the noise introduced from the process of taking the derivative. The third-order filter is designed with a Chebyshev type II digital filter design. The position and the velocity of the manipulator tip are calculated using forward kinematics calculations. The position and velocity information of the arm joints of the human subjects are also calculated from the position of the manipulator tip (handle position). The endpoint of the human arm is collocated with the manipulator tip, because the test subject grips the handle installed at the manipulator tip. Since the arm motion is confined in two-dimensional space, the joint position can be calculated from the tip position using an inverse-kinematics calculation. But this method assumes that the planar arm motion is perfectly confined in two-dimensional space. Also assumed is perfect knowledge of limb segments, that each arm link is rigid, and that the subject arm motion is caused only by two joints. Prior to the experiment, the limb segments are measured from joint center to joint center using a tape measure. The same person conducts all measurements for consistency.

5.5.3 Velocity Filter Design

The tip velocity of the manipulator is calculated from the position information measured from the encoder counts. When the velocity information is not mea-

[12]Part Number: RM15D-1024-3/8-G6-5-CA18-LD-1-C2 from RENCO

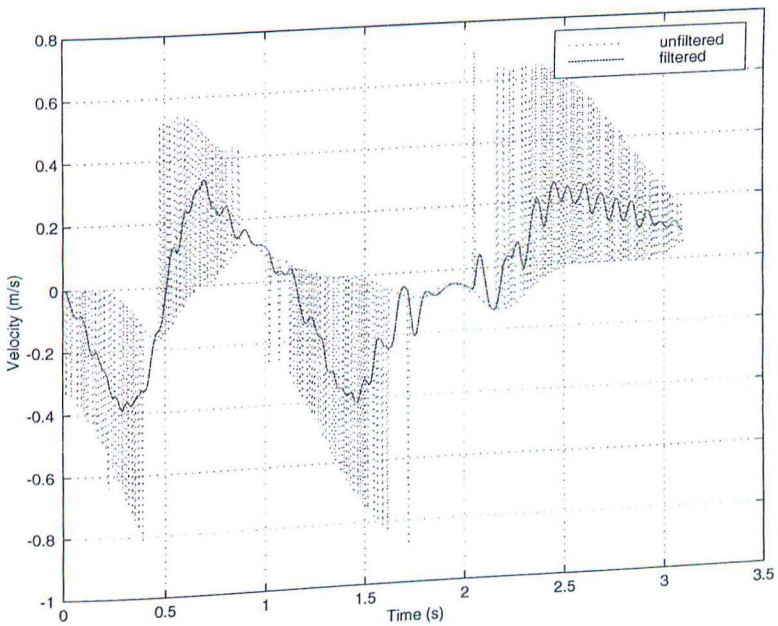

Figure 5.7: Filtered velocity data for link 1

sured with a sensor, but is calculated from position information, some noise is introduced. To extract the significant velocity value from the noisy, calculated velocity data, the noise is filtered from the velocity data. The filter is designed using MATLAB's third order, 'chebyshev filter'. The plots in 5.7 and 5.8 show the raw velocity data and the post-filter data. The filter used for TSUNAMI is shown in (5.6). The filter used is a third order low pass filter with cut off frequency of 50 Hz. The filter is designed with the stopband ripple of -40 dB.

$$y(3) = b_3x(3) + b_2x(2) + b_1x(1) + b_0x(0) - a_2y(2) - a_1y(1) - a_0y(0) \qquad (5.6)$$

where n signifies the number of order $y(3)$ is the current filtered velocity while $x(3)$ is the current raw velocity data. $y(2)$ and $x(2)$ are the filtered velocity and the raw velocity from the one sample data previous to the current time. Because of order 3, the raw and the filtered data from 3 sample periods previous to the

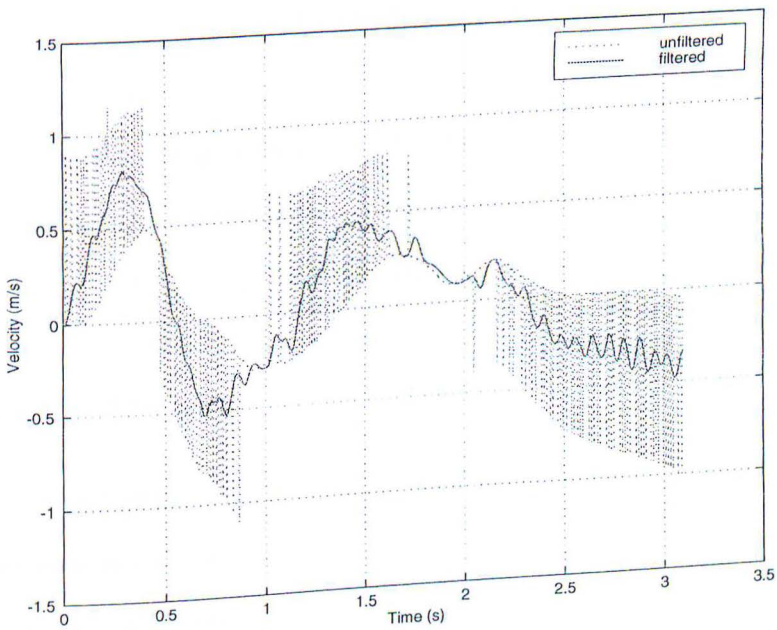

Figure 5.8: Filtered velocity data for link 2

Constants	Value	Constants	Value
a_3	1.000000	b_3	0.01829742761014
a_2	-2.18883806645027	b_2	0.00612189363910
a_1	1.67715473732202	b_1	0.00612189363910
a_0	-0.43947802837328	b_0	0.01829742761014

Table 5.3: Filter constants

current data are stored to make the filter calculations.

5.5.4 Memory and Data Management

The data is stored in RAM real-time. At the end of the experiment, the data is downloaded from PCMCIA card. The data storage is performed after the real-time data collection to reduce the computational time. The maximum data

collection time is partly dictated by the fact that running TSUNAMI and storing the data are conducted in one code, and the motor is dithered at about 500 Hz for this experiment. The data is then transferred over to Macintosh using zip drive for later analysis. The experimental data analysis is conducted using MATLAB [code in appendix]. The data collection runs at 80 Hz as previously mentioned.

5.5.5 Video Display

The display is a flat panel, plasma display manufactured by Planar[13]. This display was used because it can be easily enclosed in a custom under water casing, and it requires low enough power to be used under water for future study. Also, the display has to be large enough so that the test subjects can see the information easily and clearly. LED displays were rejected at the outset of the study because of the high cost of active matrix LED and because anticipated arm motion rates were sufficiently high to cause non-active matrix LED displays to exhibit ghosting effects.

5.6 System Modeling

To improve the experimentation, various tests were conducted on TSUNAMI. The target workspace had to be optimally placed within the manipulator workspace. The torque output was calibrated to account for the discrepancy between the output torque and the commanded torque due to motor inefficiency and the nonlinearity of the DAC output. Also, tests were conducted to construct a friction model of the two actuators. In this section of the chapter, the various tests

[13]Model Number: EL7768MS from Planar Systems Inc.

conducted on TSUNAMI are presented.

5.6.1 Torque Calibration

Theoretically, the torque output can be calculated from the equation $\tau_{out} = K_t \times i \times \tau_{command} \times (gearratio)$. K_t is the torque sensitivity. For the motor used for TSUANMI, K_t is 0.125 $lb \cdot ft/amp$ and i is the current command from the PWM servo amplifier. Realistically, the DAC board does not output a perfectly linear voltage. Nonlinearity may be introduced at the motor driver board; and there is an inefficiency in the motor which could alter the actual output torque from the theoretical value. To make sure that we have a precise knowledge of the tip force, the torque output of the motor needs to be calibrated. The force output, in a stall condition at the first set of links, is measured using a force sensor. The force sensor is fixed on the table and attached to the test link with a stainless steel cable. A voltage is commanded while dithering the motor. The resultant force output is measured. This test is repeated for both links at numerous levels of the commanded voltage. Figure 5.9 illustrates the calibration setup for the stall torque test. Link two and link four are disconnected during this test.

The output force is converted to an equivalent output torque at the base disk. The data is reduced to calculate the linear relationship between the commanded torque and the output torque. The calibration curve is a set of four linear curve fits. As a result, the equations describing the linear relationship are used in the software to command the desired torque output. Plots 5.10 and 5.11 show the calibration data along with the curves fits.

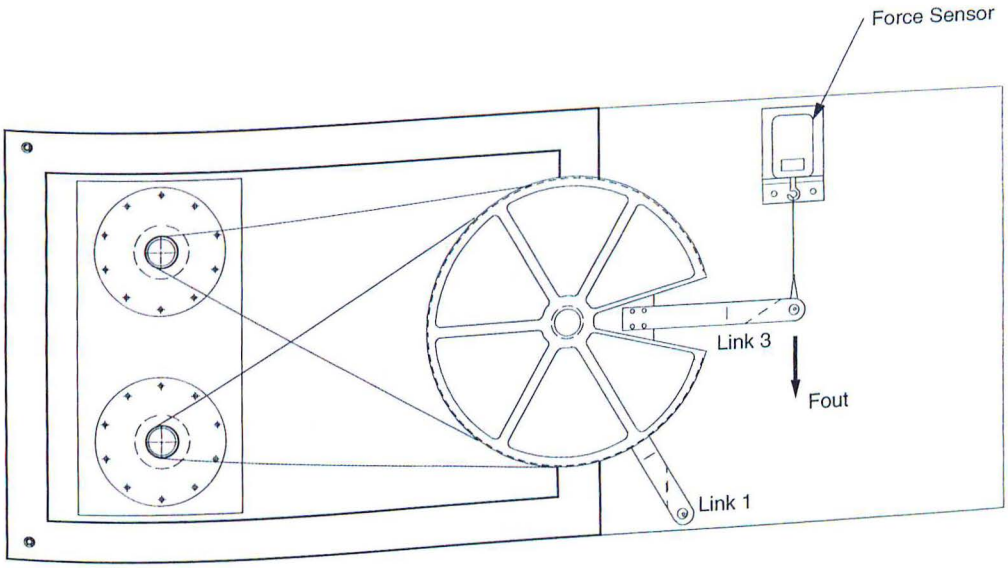

Figure 5.9: Torque calibration test setup

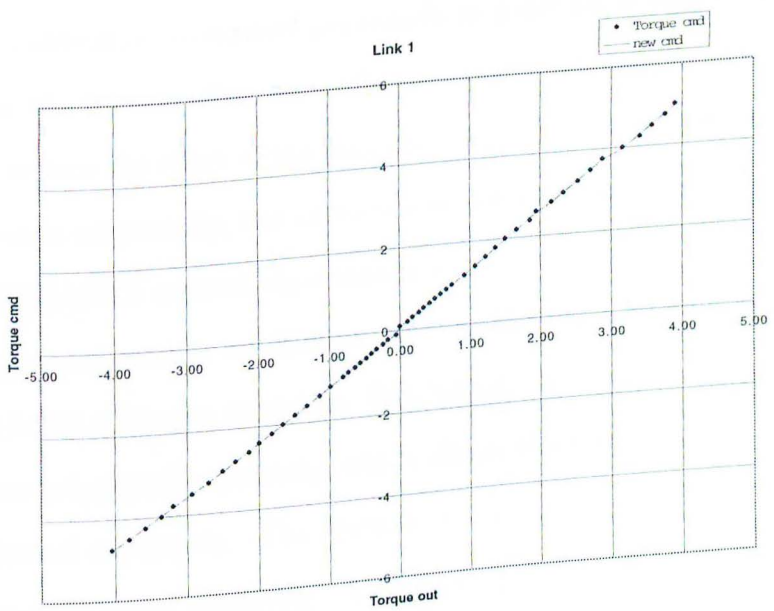

Figure 5.10: Torque calibration data for link 1

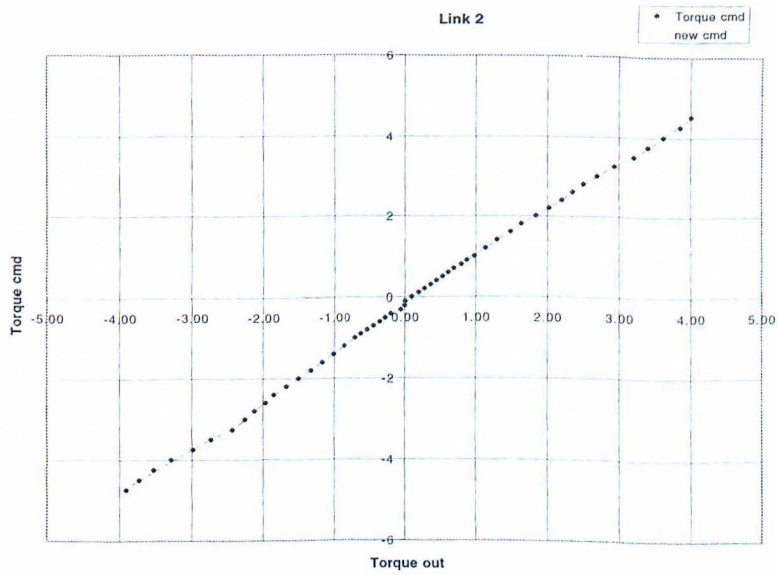

Figure 5.11: Torque calibration data for link 2

5.6.2 Friction Compensation

The torque calibration described previously is good for a stall condition. But with motion, the friction effect also influences the torque output. The motors are dithered to reduce the effect of the friction. The dither amplitude and frequency are both chosen empirically, but there is still not enough friction compensation. Because TSUNAMI's control algorithm does not include the force feedback, it is essential to compensate for friction by constructing a friction model for the system. To build a friction model for TSUNAMI, the friction model for the two actuators are calculated separately, which allows the measurement for each link to be conducted separately. The friction at the bearings installed in the link joints is assumed to be negligible. This assumption is viable because most of the friction comes from the actuator assembly. To conduct the friction test, a known voltage with dither is commanded to the test link. The link is free to

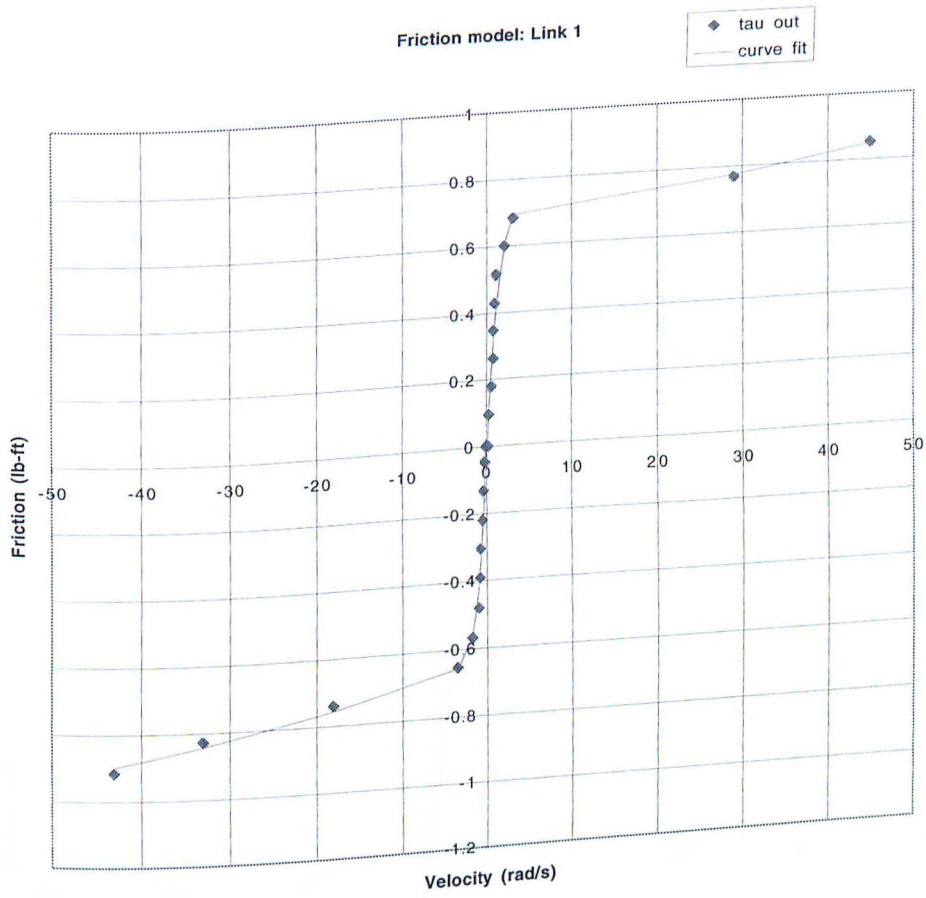

Figure 5.12: Friction model for link 1

move while the position and the velocity data of the link are recorded for later analysis. Friction is modeled as a function dependent solely on velocity, hence the commanded torque is plotted against measured steady state velocity. The commanded torque, once the steady state velocity is reached, balances only the friction at that particular velocity since acceleration is zero, hence the inertial effect is zero.

From the friction test, the data is plotted on the Figures 5.12 and 5.13. To include the friction compensation model in the control of TSUNAMI, the equations which describe the relationship between the friction and velocity must be

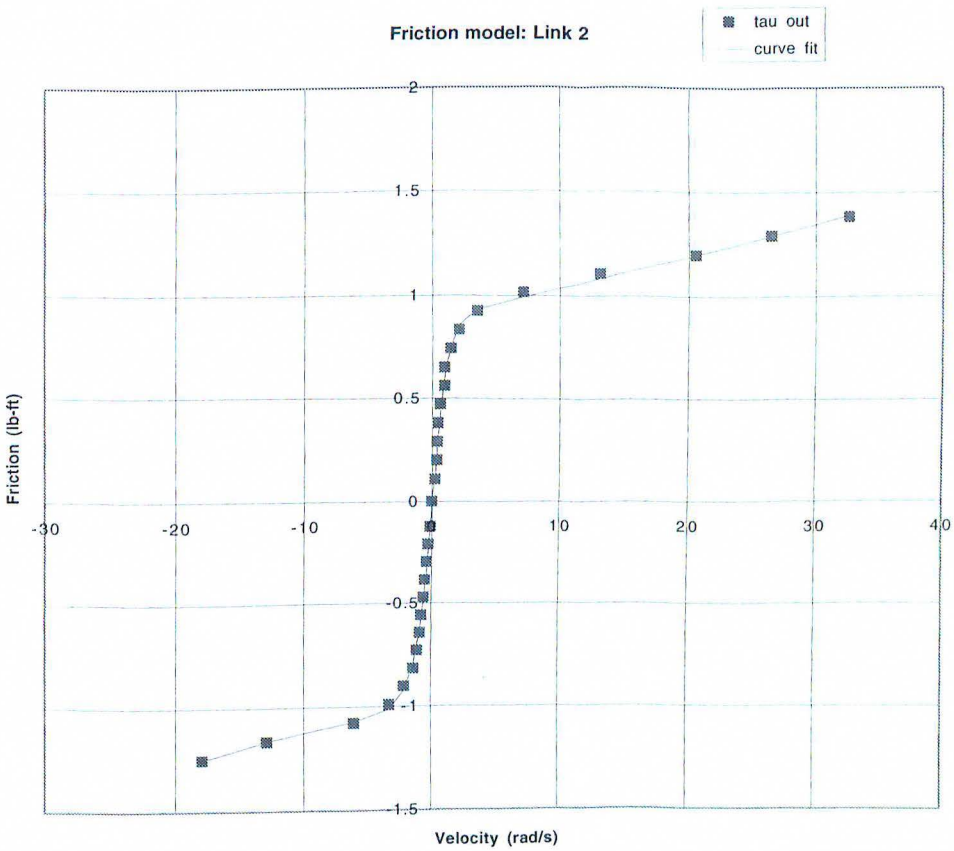

Figure 5.13: Friction model for link 2

estimated. To perform the curve fit, the form of equation must first be chosen. The simplest form of curve fit is the linear curve fit. But, obviously, the linear equation would be grossly incorrect. Another level of complexity would be a linear curve fit with a stiction torque at $\dot{q} = 0$. Note that the curve and the offset at zero is not necessarily the same for $\dot{q} > 0$ and $\dot{q} < 0$. This is still a crude estimate for the lower range of velocity. It is previously reported by [3], [5] that the friction has a very nonlinear function near $\dot{q} = 0$. This effect is know as "Stribeck effect" or as slip-stick friction. For TSUNAMI, the function described by the equation (5.7) [9] which includes viscous, stiction, and Stribeck effects, is used for the curve fit.

$$\tau_{fric}(\dot{q}) = [c_0 + c_1 e^{(c_2|\dot{q}|)} + c_3 \left(1 - e^{(c_4|\dot{q}|)}\right)] sgn(\dot{q}) \tag{5.7}$$

Table 5.4 lists the values used for the coefficients for the friction model equation, (5.7) to create the curve fit approximations for friction values.

Link #	c_0	c_1	c_2	c_3	c_4
Link 1 (positive)	-0.074952	9.79E-05	0.14962	0.841276	-0.77746
Link 1 (negative)	3.757823	-3.79467	-0.0016	0.68793	-1.10855
Link 2 (positive)	-4.637	4.4585	0.00323	1.0691	-1.3546
Link 2 (negative)	-0.1405	0.12703	0.0585	1.0331	-1.0507

Table 5.4: Friction model monstants

5.7 Test Description

A typical experiment using TSUNAMI will be a upper-limb, motor control study where a subject will execute reaching motions while holding onto the manipulan-

dum end-effector. The pseudo-random target locations are displayed on the video display mounted on top of the manipulator base. Test subjects are instructed to move a cursor, which represents the location of the end-effector, to the square target by moving the manipulandum. During testing, variable force can be applied by the manipulandum. The position and velocity data of the manipulandum will be recorded for later data analysis. The position and velocity of the test subject's joint angles are calculated from the manipulandum data using inverse kinematics equations.

5.8 Software Description

The operating system on the TSUNAMI computer is DOS 6.2.2 and the compiler is gcc for DOS called djgpp which is available on the Internet at no cost. The experiment apparatus was programmed using djgpp on a 386 computer. TSUNAMI code requires a large block of RAM available for data storage (amount beyond DOS's 640K flat memory limit), and has interrupt driven functions; gnu based c compiler was used since it allows the entire memory space to be accessed. The TSUNAMI software performs manipulator control, graphical output, and data collection. It is included in the appendix A and B.

5.8.1 Control Scheme

Control of the manipulator is fairly simple. The proposed experiments require the manipulator to apply a force which is a function of current tip position and velocity, or a function of current joint angles and velocity. This test requires only torque control, which is achieved by simply sending out command current that is

proportional to the desired motor torque. The resulting torque is not fed back, but a calibration test was performed to assure that the output torque is equal to the commanded torque.

5.8.2 Execution Time

The two slowest portions of the code are displaying graphics to the video output and writing data to a data file. To run the code faster, the data is stored in RAM during testing, and at the end of a test session, the data from RAM is written to a data file. Furthermore, the target is not updated every main loop cycle, but is limited to 33 Hz. The target remains on the screen until the display function is called. The display function erases the target from the screen and calculates the new target position. It then updates the new target position if the full motion duration is reached. Otherwise, the same target is drawn on the screen. The motor is dithered at 66 Hz. Therefore, to attain an equal amount of dither voltage in both directions, the torques are commanded at much faster frequency. The dither calculation and DAC output occurs in the main loop which executes the fastest.

5.8.3 Interrupt Driven Code

To run the system and record data at a fixed rate, the actuation function, display function, and the data collection function are all tied to the operating system interrupt. As shown on Figure 5.15, the main loop continuously runs. An interrupt handler is tied to a function which sets the flags for RunTSUNAMI, Display_learn, Display_star, and TakeDATA. If the flag for a function is set to an "on" state, that function is executed. RunTSUNAMI executes every time the

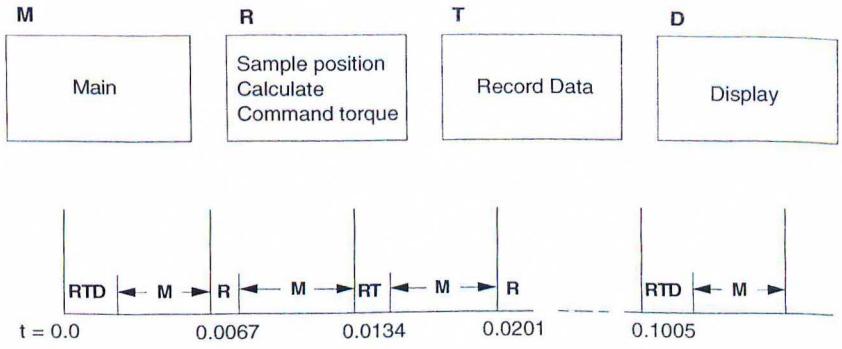

Figure 5.14: Software timing diagram

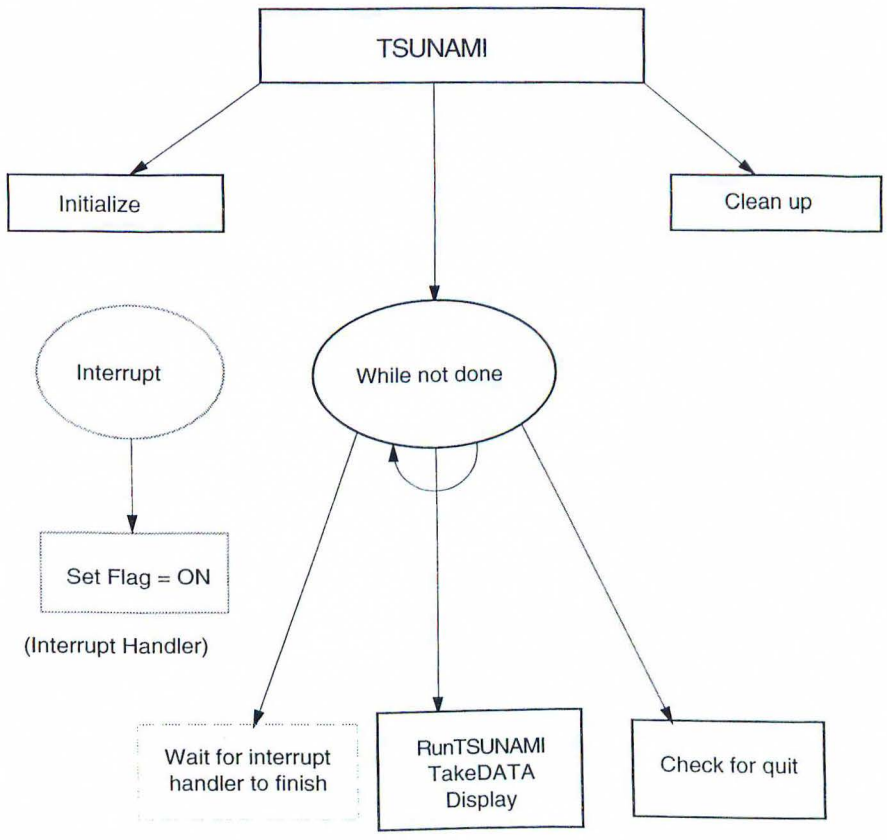

Figure 5.15: Algorithm diagram

interrupt fires, while data collection function execute every other interrupt, and display function only execute every fifth interrupt. Figure 5.14 shows the timing sequence of the function execution.

5.8.4 Target Generation

The target is generated by calculating a fixed length distance in a pseudo-random direction from the current position. The target is constrained to remain within the target work space. The directions are pseudo-random because it is picked randomly using the C pseudo-random number generator from eight possible directions equally spaced at 45^o. The algorithm to display the target during the experiment has two components. First, during both the learning phase and the evaluation phase, a pseudo-random target has to be calculated. Secondly, during the evaluation phase, the target from all eight directions must be executed twice. The algorithm must pick the next target while trying to minimize the number of sequences required to execute the eight directions. The algorithm has to keep track of which target has been already generated and which has not. All of the target directions must be taken twice; once with, and once without the applied force field on.

A new target is presented every 2.0 seconds, but each reaching motion takes only about 650 milliseconds. The longer duration is allocated to accommodate for the test subject's reaction time which is about another 500 milliseconds and a possible longer reaching time due to the deviation in the reaching trajectory induced by a disturbance force field.

During each target presentation, the display function is called about 65 times. There are two timing variables calculated in display function. One is the indexing,

which is counted every time the display function is called and reset to 0 when a new reaching sequence is initiated. The second variable keeps track of whether the time has increased beyond 2.0 seconds. If the time is equal to or exceeds 2.0 seconds the variable n is set to 1, otherwise it is 0. The purpose of this is to know when the next reaching sequence should be started.

The computer generates a random integer between 0 through 7. This integer is converted to an angle from 0^o to 360^o at intervals of 45^o. When the loop counter index is equal to 60, the next target location is calculated from the angle. It is previously determined that maximum index for a reaching sequence is between 63 and 67. Therefore, by the time the index reaches 60, the test subject should have reached the previous target position and is waiting for the next target position to be displayed. The target location is calculated to be 0.29528 ft (9cm) from the current tip position at the random angle previously mentioned. This target position is checked to see if it remains in the workspace. If it does not, another target position which will stay within the target workspace will be calculated. The current tip location is compared to the workspace grid shown in the Figure 5.16.

The grids in Figure 5.3 are created to provide previously calculated target directions which remain within the confines of the workspace. The position is first checked to see if it is within the center grid. If so, a random direction is picked out of the direction set of 1, 3, 5, or 7. For each workspace grid, calculations are conducted to find which directions the next targets can go and still remain in the target workspace. If the tip position is not currently in the center grid, then it is placed in one of the four distinct, workspace quadrants. Each quadrant is further divided into four sections. The optional directions for those areas are

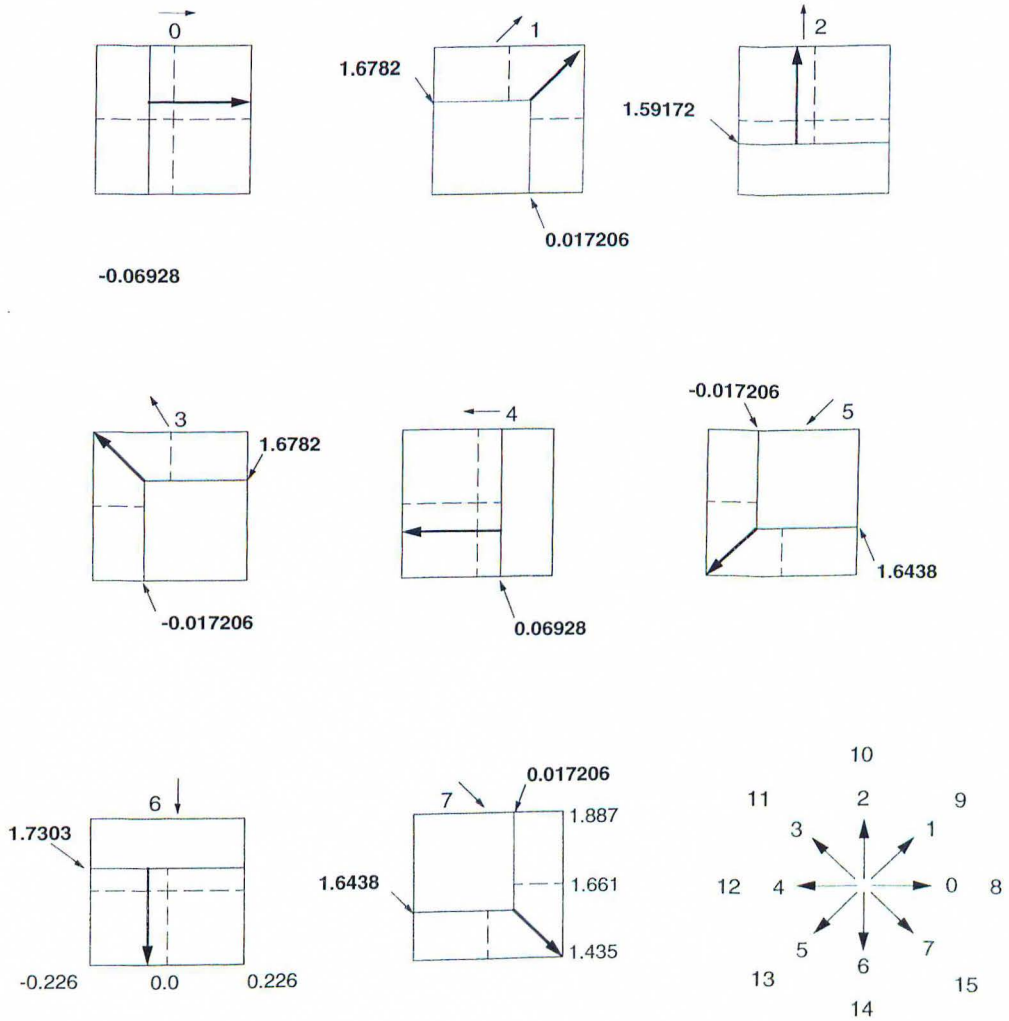

Figure 5.16: Calculation of reachable space for each directions

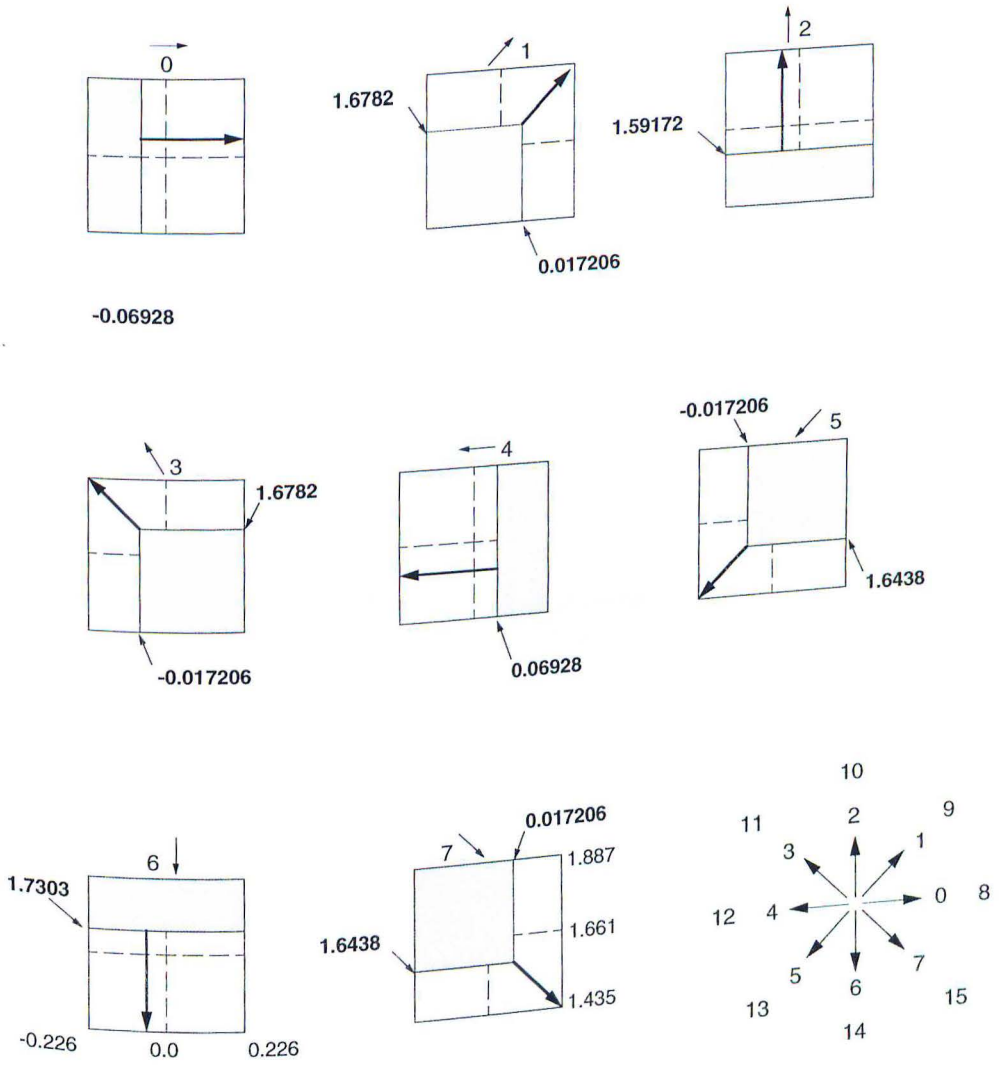

Figure 5.16: Calculation of reachable space for each directions

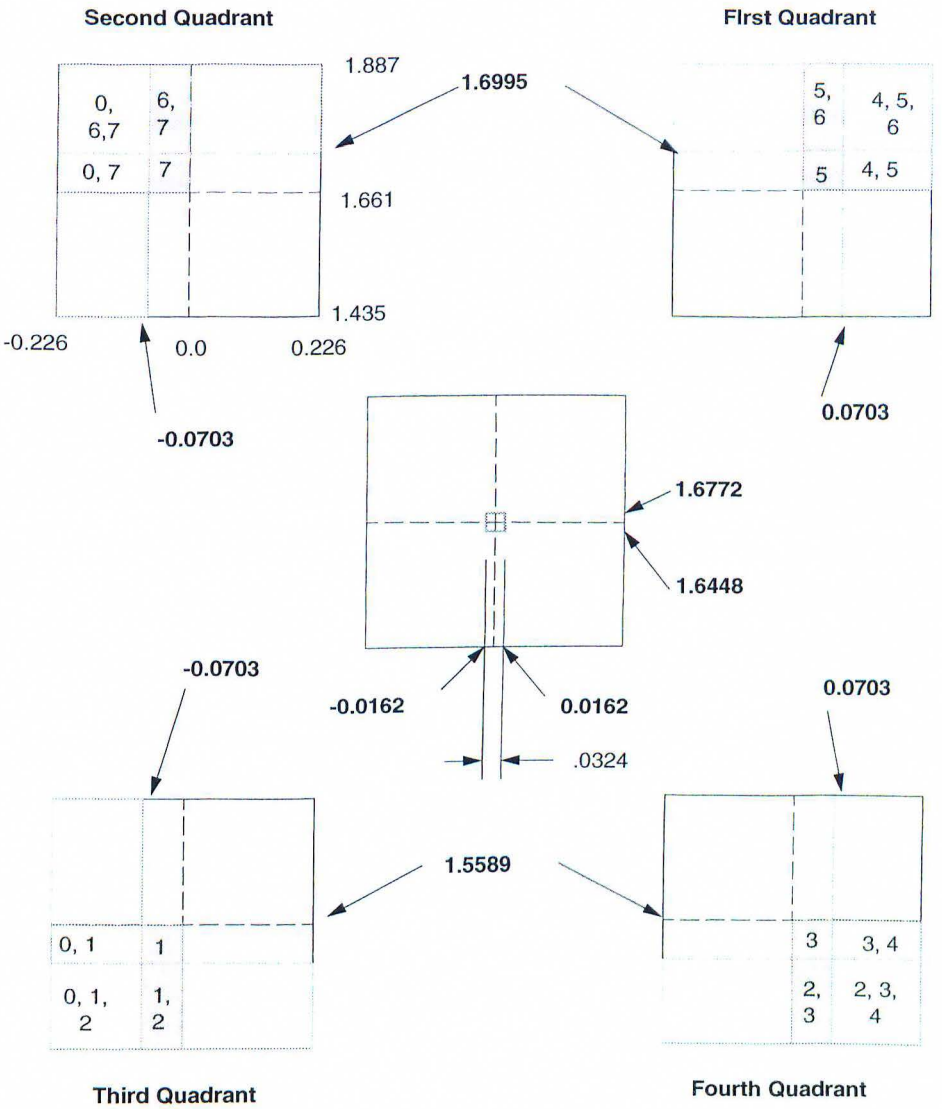

Figure 5.17: Workspace grid: workspace is divided into 4 distinct quadrants and an additional grid at the origin.

indicated in the diagram. One of those directions is picked to calculate the next target position. At the end of the reaching duration, the new target location is displayed and the loop counter index, m is reset to 0.

The second part of the display function makes sure that all eight directions are displayed twice; once with force field "off" and once with force field "on". Figure 5.18 illustrates the algorithm to keep track of which directions have been used for the target generation. For the evaluation phase, a status flag is set to "evaluation" and a direction is selected from the target array which holds numbers between zero to fifteen in a random order. The numbers from zero through seven will represent directions from zero through seven with the force flag set to "off", while the numbers from eight to fifteen represent directions from

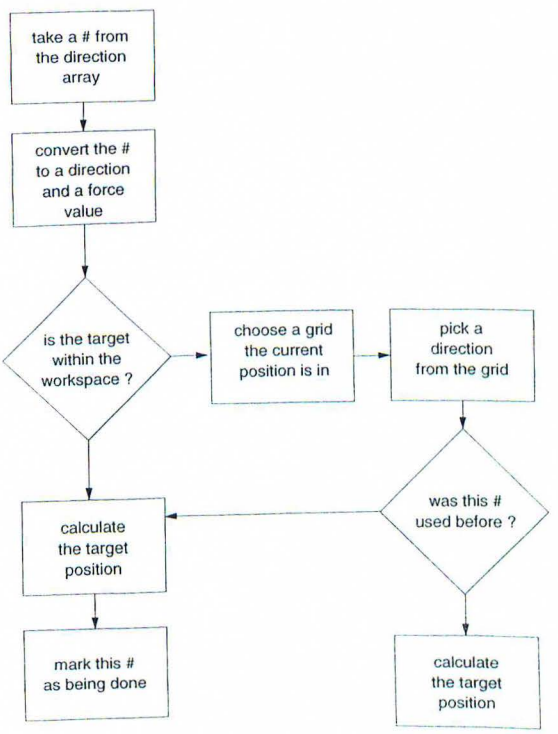

Figure 5.18: Target generation algorithm

zero through seven with force flag set to "on". From the numbers provided, a candidate target location is calculated. If the target doesn't remain within the workspace, a new target position is calculated in the same manner. In the cases where the target position is calculated from the first selection of a number from the target array, that value in the array address is marked as having been executed, and the target array is indexed until the next element in the array is not labeled as having been executed. If the first selection from the target array does not produce a viable target position, another number is selected as described earlier. Then, that direction number has to be checked to see if it has been executed previously with force "on" and/or "off". To accomplish this, two variables are created, one equal to the direction number and the other equal to the direction number plus eight. Then the first variable is checked to see if the direction number still exists in the target array. If it does exist, the target location is calculated from that value, the force flag is set to "off", and the number is labeled as having been executed. The target calculation is now complete for this reaching sequence. The array index is moved until the next element is not labeled as having been executed. If the number is not found, that direction was executed with the force field "off", and was marked as completed by setting the value to -1. Second variable, direction number plus eight, is checked to see if it still exists in the target array. If it is found in the array, the target location is calculated, the force flag is set to "on", and marked as having been executed. If the direction value plus eight also doesn't exist, the target value is calculated and the force flag is set to"on", but there is nothing to be done with the target array. At this point, the array index is moved until the next element is not labeled as having been executed. This is repeated until the array index reaches sixteen, at

which time the status flag is set to "complete" and the evaluation phase ends. The target generation algorithm is illustrated in Figure 5.18.

5.9 Preliminary Experimentation

A preliminary experiment of human reaching motion was conducted with TSUNAMI. To investigate the capabilities of this test apparatus to study upper limb motion, a simple experiment was conducted. This section describes the experiment method and the result.

In order to conduct this experiment, TSUNAMI was built to measure the human reaching motion during null and specified force field applied to the human subject at his/her hand. This specified force field, disturbance to the reaching motion, is chosen to be proportional to the arm motion velocity because the most force field experienced in nature is proportional to velocity, for example, drag.

The velocity and the tip position of a test subject is measured and analyzed to study the changes in reaching motion during disturbance force field. The disturbance force field used in this experiment is the same force field used in the simulation described in chapter 4 and in the experiment conducted by Shadmehr and Mussa-Ivaldi [46]. The goal of this preliminary study is to evaluate TSUNAMI's capability to study adaptive motor control of humans. Hence, the experiment is performed to have a test subject execute reaching motions in the same force field as used by Shadmehr and Mussa-Ivaldi [46] and as used by the simulation described in chapter 4. The reaching motion in a well known environment referred to as the 'null field' and in disturbance force field is studied to see if the result will correlate well with the previously reported data from chapter 3 and chapter 4.

5.9.1 Experiment Method

First, the test subject practices the reaching motion for 250 reaching cycles in no disturbance force field called 'null field'. Test subject is presented with a target described in the target generation section. Test subject is instructed to move the manipulator handle to the target position and rest until a new target is presented. New target is displayed every 2.0 seconds. The current tip position is available as a circular cursor moving on the screen. During the practice session, the tip position and tip velocity data is recorded to be analyzed later. Following the practice session, a similar test is conducted to study the reaching motion executed in the disturbance force field. The tip position and tip velocity data is again recorded for later analysis.

5.9.2 Test Setup

During the experiment, the test subject is outfitted with a wrist guard to prevent motion at the wrist. The subject is seated in a car seat fixed in front of the test stand with the table being at the shoulder height. The body position is fixed with respect to the testing table to assure all motion is executed through the shoulder and the elbow joint. The table, which holds up the test subject's elbow to constrain the arm motion in a plan, is covered with a teflon sheet to reduce the friction effect on the test subject's arm. At the start of the experiment, once the subject is seated, the initial angles of the elbow and the shoulder joints are measured.

5.9.3 Experiment Result

Plots in, Figure 5.19 to 5.21, show the experimental results of the reaching test described in the previous section. The reaching motion executed in the 'null field' exhibit straight path to the target position, as reported in studies by [14], [51], [16] [46]. Plot shown in the Figure 5.20 is one of the reaching motion executed in the 'null field'. This reaching motion is executed in the positive x, negative y direction, 135°. The reaching path is straight to the target while velocity curve is bell shaped as expected. The reaching motion occurs about 490 milliseconds after the target was presented. The reaction time, the time from the presentation of the target until the movement is initiated, ranged from about 480 milliseconds to 520 milliseconds. The movement duration seen in this experiment was about 750 milliseconds instead of the reported 650 milliseconds [46].

Figure 5.21 shows the reaching path generated during execution of reaching motion at the initial exposure to the disturbance force field. The reaching motion executed in the disturbance force shows the same hook pattern as seen in the simulation and in the experimental result of Shadmehr and Mussa-Ivaldi study [46]. The reaching motion in the force field takes longer trajectory due to the influence of the arm being pushed away from the straight path, and the corrective motion made toward the target.

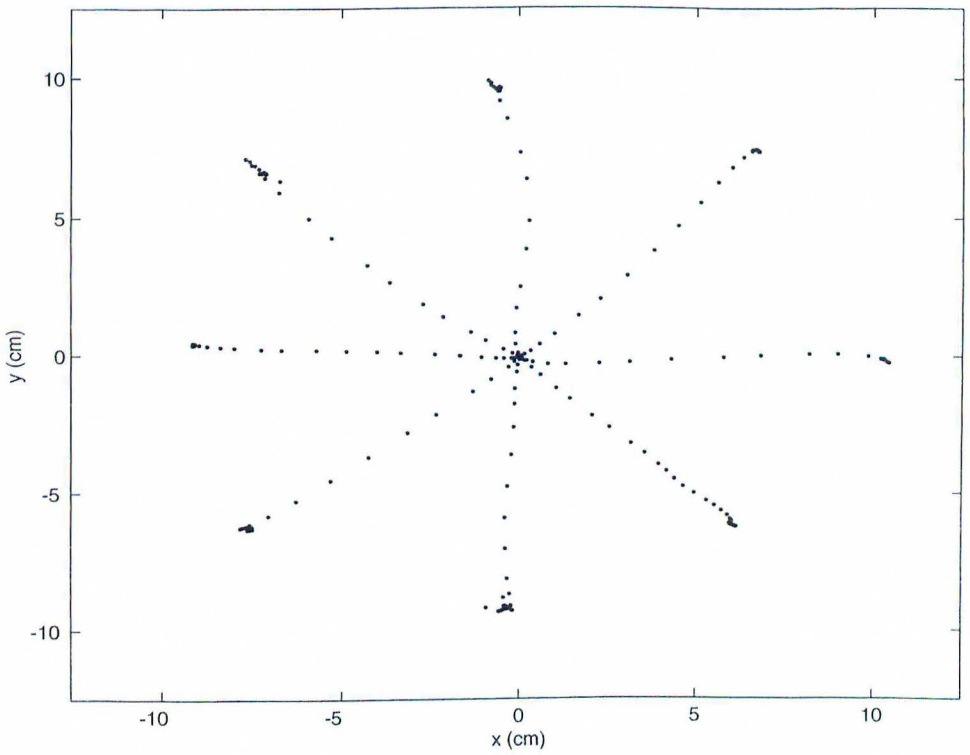

Figure 5.19: Reaching motion in null force field

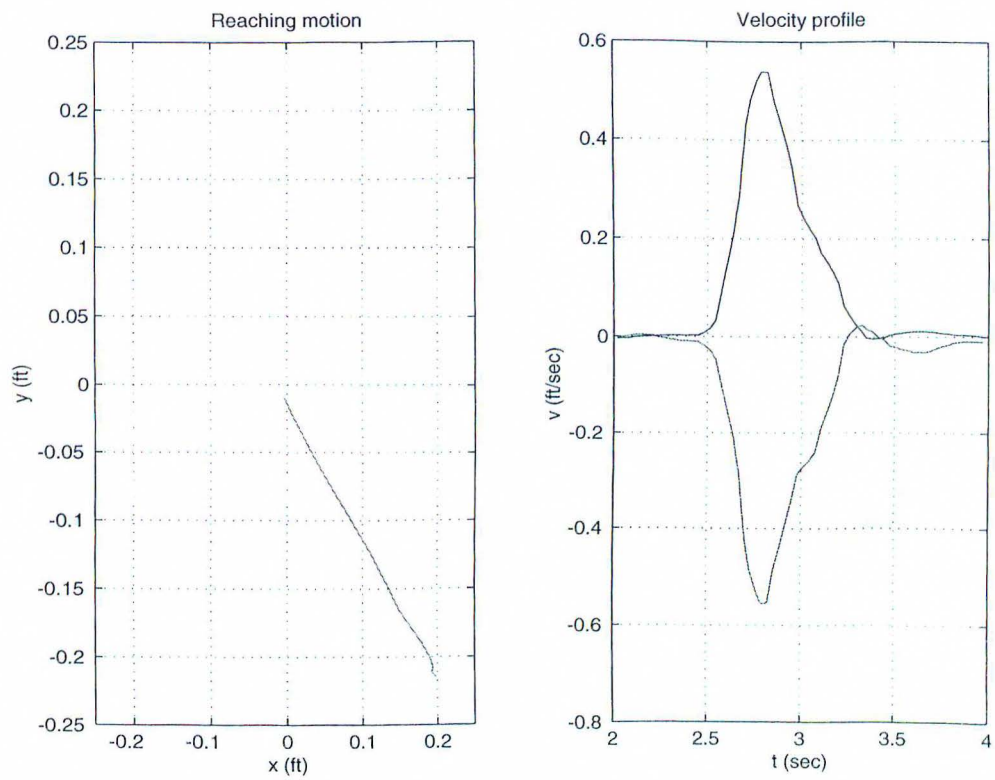

Figure 5.20: The reaching motion and its velocity profile

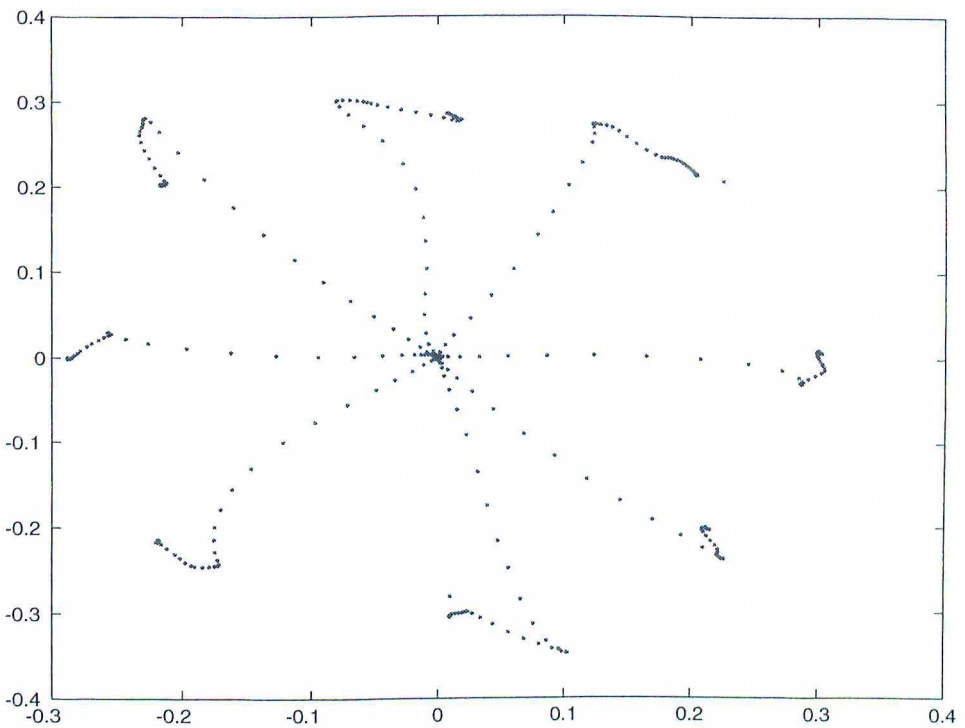

Figure 5.21: Reaching motion conducted during an initial exposure to the disturbance force.

Chapter 6

Conclusion and Further Study

6.1 Discussion

The adaptive motor control model proposed in this thesis appears to successfully capture the characteristics seen during the learned point-to-point reaching motions of upper limb in a horizontal plane. The model also describes a plausible algorithm describing the adaptive construction of an internal model by the CNS. The adaptive computational element used in the model proposed in this thesis uses the Gaussians, but the model is not sensitive to the particular basis function chosen for the function approximation. IF the physiological data supporting a certain type of computational element become available, this model can incorporate the new motor computational element.

The internal model of the environmental force is constructed in a similar manner to the convergent force fields observed in Bizzi's study [7]. Both the convergent force field, which is produced in the spinal cord, and the low-level elementary computational units are computed in intrinsic coordinate frames.

6.2 The Upgrades to TSUNAMI

The obvious first step to the upgrade to TSUNAMI is to complete waterproofing the TSUNAMI system. Additional work needed to make TSUNAMI function under water includes installing waterproof electrical connectors to the motor driver casing, the video display casing, and the electronics box. The video display casing, motor driver casing, and the electronics box should be pressurized as well. The knot which fixes the shaft sleeve on the motor shaft needs to be redesigned so that there is no water leakage through the connection between the knot and the shaft sleeve. Also the power cable needs to be built to provide power to the electronics from the surface.

Another improvement recommended for TSUNAMI is to modify the actuator casing to make sure that the motor shaft is aligned better. I suspect the motor shaft is ever so slightly off angle which is causing an additional friction that is position dependent. The cable connection must be aligned just right also.

6.3 Recommendations for Future Research

The mathematical model proposed in this thesis is an early attempt to more ambitious research bridging the human physiology to engineering. Numerous future studies can come out directly from this study.

The first step is to strengthen this study by conducting multiple subject experiments, perform statistical analysis on the experimental results to quantify the differences between the subjects, and compare to the difference between the simulation result to the average experimental result. This study would allow quantified comparisons between the model and the human motion.

Second, extend the study to encompass more disturbance environments that are more meaningful especially for NASA and the SSL. The model can be tested under the influence of disturbance modeling the drag force in the neutral buoyancy environment. The comparison of the result to the experimental result obtained from the human experiment obtained from under water will evaluate the simulation's predictive ability. Also the data from neutral buoyancy simulation will be a step closer to improving the astronaut training program, by potentially providing a powerful analytic methodologies to training astronauts.

On more scientific note, instrument the target muscles, such as biceps and triceps pair, with EMG sensors during testing will provide further human physiological data to improve the model which is not currently available. By studying the change in the muscle recruitment during learning, issue of stiffness change during learning can be investigated, providing another data which can be incorporated into the model.

This same model can be extended to a full 6 DOF model of adaptive neuromuscular control by modeling the human posture control during execution of arm motion in multiple environments. This study will also extend a research conducted previously at the SSL to quantify the simulation differences between the neutral buoyancy, KC-135, and the microgravity environment by observing the differences in the postural maintenance in those environments [53]. The disturbance force to the postural control will be present due to the execution of arm motion. Conduce an experiment of postural maintenance during reaching motion in 1-g lab environment and the neutral buoyancy environment in order to quantify the changes in sensory-motor coordination across two environments. In time the same experiment can be conducted on-board KC-135 and on-board

space shuttle to provide quantitative differences of the two microgravity simulation environments to the actual microgravity environment.

The control model proposed in this thesis with the developments mentioned above can be applied to designing a better control of the prosthesis. The electrical signals of the motor control commands from the human body can be measured and can become the input to conducting simple motions of the prosthesis. This would give more capabilities and closer to typical human functionality to a state of the arm prosthesis.

Appendix A

Header File for the Experiment Software

This header file is linked with RUN.C to run TSUNAMI and collect data during experiments. The headerfiles include defined variables, initialization of functions, and the defined shortcuts.

```
/*
 ------------------------------------------------------------------------------------------------------------------------
*       Filename:       Arm2.h
*       Current location:       e:\gprog
*       Included in file:       run.c
*       Date last changed:      June 99
 ------------------------------------------------------------------------------------------------------------------------
*/

typedef unsigned char byte;

/**********         Constants          **********/
#define READ_CHAN0      0x84
#define READ_CHAN1      0x85
#define READ_CHAN3      0x87
#define CLEAR_CHAN0     0x88
#define CLEAR_CHAN1     0x89
#define CLEAR_CHAN3     0x8B
#define DCH1_LOW        0x0220      /* DAC board base address          */
#define DCH1_HIGH       0x0221      /* DAC : base + 1                  */
#define DCH2_LOW        0x0222      /* DAC : base + 2                  */
#define DCH2_HIGH       0x0223      /* DAC : base + 3                  */
#define DOUT_CTL        0X0225      /* DAC : base + 5                  */
#define EBASE_0         0x0210      /* Counter board base address */
#define EBASE_1         0x0211      /* Counter: base + 1               */
#define EBASE_2         0x0212      /* Counter: base + 2               */
#define EBASE_3         0x0213      /* Counter: base + 3               */
#define E_CMMD          0x0216      /* Counter: base + 6 cmmd register */
#define REV_CNT         4096    /* (Pulse cnt)/rev: 1024*4 for quadrature */
#define FREQ            500000.0    /* Freq. of encoder counter        */
#define HALF_CTS_PER    2147483647
#define FULL_CTS        4294967295  /* Max. number of encoder counter  */
#define ENC_CNT         4096        /* 1024 puse/rev *4 for quadrature */
#define GEAR_RATIO      15.254237288  /* Gear Ratio                    */
#define L1              0.89808     /* Link 1 length  */
#define L2              1.2498      /* Link 2 length  */
#define KT              0.125     /* Torque sensitivity                */
#define BASE8254        0x0040
#define TIMER           8           /*  timer interrupt 8              */
/*   Filter coefficients   */
#define B_3 0.01829742761014
#define B_2 0.00612189363910
#define B_1 0.00612189363910
#define B_0 0.01829742761014
#define A_2 -2.18883806645027
#define A_1 1.67715473732202
#define A_0 -0.43947802837328
```

```
                        /* filter order + 1 */
#define Nf  4
#define MaxNf   5

/***             Function Macro                          ***/
#define LOCK_FUNCTION(x)    _go32_dpmi_lock_code(x, (long)sizeof(x))
#define LOCK_VARIABLE(x)    _go32_dpmi_lock_data((void *)&x, (long)sizeof(x))


/*  clear_reg(command)
 *
 *  This function will clear the register which is specified
 *  by 'command'.  For encoder counter board.
 */
#define clear_reg(command)      outportb (E_CMMD, command)

/* dacout(void)
 *
 * This function will enable DAC output.
 * set highest bit to 1 to enable output
 */
#define dacout()            outportb(DOUT_CTL, 0x80)

/*
*  Sign(voltage)
*
*  Take in the voltage value calculates the sign of
*  the voltage value
*  Returns:
*  float   -1.0:
*  float   +1.0:
*/
#define Sign(voltage)   ((voltage<0.0)?-1.0:(voltage>0.0)?1.0:0.0)
                                                    ***/

/***        Declare Structures
struct var{
float a;
float b;
float c;
float d;
float e;
};                                                  ***/
/***             Function Prototyping

void Const_volt(float, int);
void dac1(float);
void dac2(float);
void dacShutdown(void);
void dataFile(void);
void Display_learn(void);
void Display_star(void);
void Flag_Maker(void);
void initial(void);
void RunTSUNAMI(void);
void TakeDATA(void);
float Dith1(float);
float Dith2(float);

float volt_cmd(int, float);
float Ang_calc(int, float);
float Filter(float, float);
float Rot_calc(int, float);
float StatFrict(float);
float time(void);
float V_calc (float, float);
float Vel_calc(float, float, float);
float RollOver(float);
//    float RollOver(float, float);
float enc(byte);
//    double enc(byte);
struct var Input_key (void);

/*   Temporary for xy.c     */
//    void clear_reg(byte);
```

Appendix B

Code to Execute Experiment

This program runs TSUNAMI for the experiment and collects data for later analysis.

```
/*
-------------------------------------------------------------------------------
FILE:   RUN.C
DATE:   June 29,1999
CODE:   interrupt driven TSUNAMI code
FOLDER: GPROG
-------------------------------------------------------------------------------
*/

#include <stdio.h>
#include <stdlib.h>
#include <pc.h>
#include <dpmi.h>
#include <go32.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include "grx20.h"
#include "drawing.h"
#include "grfontdv.h"
#include "arm2.h"
#ifdef _GNUC_
extern int getch(void);
extern int kbhit(void);
#endif

#define Dither_t    0.0075  /* check this part out  timeing issue */
#define Dith2_t     0.015
#define Dv  0.75
#define ON  1
#define OFF 0
#define XMAX    0.225       /* 2.70 inches */
#define XMIN    -0.225      /* -2.70 inches */
#define YMAX    1.885       /* 22.62 inches */
#define YMIN    1.435       /* 17.22 inches */
#define Rch_time 2.0
#define Eval    5
#define COMPLETE    3

int tx[2], ty[2];
int disx, disy;
int dirtest[17] = {0, 10, 5, 7, 3, 9, 14, 11, 6, 1, 12, 15, 2, 4 , 8, 13, 0};
int basedir[17] = {0, 10, 5, 7, 3, 9, 14, 11, 6, 1, 12, 15, 2, 4 , 8, 13, 0};
int drcase0[4] = {1, 3, 5, 7};
int drcase1[3] = {4, 5, 6};
int drcase2[3] = {0, 6, 7};
int drcase3[3] = {0, 1, 2};
int drcase4[4] = {2, 3, 4};
int drcase11[2] = {4, 5};
int drcase21[2] = {0, 7};
int drcase31[2] = {1, 0};
int drcase41[2] = {3, 4};

int drcase12[2] = {5, 6};
int drcase22[2] = {7, 6};
```

```
int drcase32[2] = {1, 2};
int drcase42[2] = {2, 3};

/**************/
float v1 = 0.0;
float v2 = 0.0;
float tt;                              /* dummy time variable */
float var_t[10000], var_1[10000], var_2[10000];  /* stored data */
float var_3[10000], var_4[10000], var_5[10000], var_6[10000];
int var_7[10000];
float data_x, data_y;                          /* global x, y */
float disp_x, disp_y;
/*-------------------------*/
int cycle = 0;
int direct = 0;
int Td1 = 0;
float Fr1 = 0.0;
float Fr2 = 0.0;
/*-------------------------*/
float nvolt1 = 0.0;
float nvolt2 = 0.0;
float dumt = 0.0;
float subt = 0.0;
float xnext, ynext;
int modeFlag = 0;
int finish = 0;
int Force = OFF;
int Warning = 0;
int cnt_d = 0;
int cnt_t = 0;
int cnt_r = 0; /* function enabling flags for 3 primary functions */
volatile int counter = 0;

void Flag_Maker(void)
{
if (cnt_d == ON || cnt_t == ON || cnt_r == ON)
{
if (cnt_d == ON)
Warning = 1;    /* Display() is running */
if (cnt_t == ON)
Warning = 2;    /* TakeDATA() is running */
if (cnt_r == ON)
Warning = 3;    /* RunTSUNAMI() is running */
}
else if (counter%15 == 0)
{
cnt_d = ON;
cnt_t = ON;     /* should be ON */
cnt_r = ON;
}
else if (counter%5 == 0)
{
cnt_d = ON;
cnt_t = OFF;     /* should be OFF */
cnt_r = ON;
}
else
{
cnt_d = OFF;
cnt_t = OFF;     /* should be OFF */
cnt_r = ON;
}
counter++;
}

int main(void)
{
_go32_dpmi_seginfo  old_handler, new_handler;
float t = 0;
float dt = 0.0;
float t_old = 0.0;
int k = 0;
char record[] = "rec.dat";
FILE *fp;

/*  Open Data File  */
fp = fopen(record, "w");
dacout();
dac1(0.0);
dac2(0.0);
/*--------   change the timer rate   --------*/
outp( BASE8254+0x03, 0x04);
outp( BASE8254+0x00, 0xE1);
outp( BASE8254+0x00, 0x1C);
/*--------    set up for graphics mode  --------*/
GrSetMode(GR_default_graphics);
```

```
GrClearScreen(GrAllocColor(0, 0, 0));
GrSetMode(GR_width_height_color_graphics, 640, 480, 16);

getkey();

/* load the address of the old timer ISR into the OldISR structure     */
_go32_dpmi_get_protected_mode_interrupt_vector(TIMER, &old_handler);
/* point NewISR to the proper selector:offset for handler function     */
new_handler.pm_offset = (int)Flag_Maker;              /* integer value is the function pointer value */
new_handler.pm_selector = _go32_my_cs();
/* chain the new ISR onto the old one so that first the old timer ISR  */
/*   will be called, then the new timer ISR                            */
_go32_dpmi_chain_protected_mode_interrupt_vector(TIMER, &new_handler);

initial();
tt = time();
disx = 2;
disy = 2;
xnext = -0.041667;
ynext = 1.625;
tx[0] = 520;
ty[0] = 90;
tx[1] = 320;
ty[1] = 240;
modeFlag = Eval;
Force = OFF;
/*---------   First Evaluation   -------------*/
while ((modeFlag == Eval) && !kbhit() )
{
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
/*--------------------------------------------*/
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*--------------------------------------------*/
if (cnt_d == ON && Warning == OFF)
Display_star();
/*--------------------------------------------*/
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*--------------------------------------------*/
if (cnt_t == ON && Warning == OFF)
TakeDATA();
t = time();
tt = t;
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
if (Warning == 1)
GrCircle(0, 8, 8, GrAllocColor(255,255,255));
if (Warning == 2)
GrCircle(200, 8, 8, GrAllocColor(255,255,255));
if (Warning == 3)
GrCircle(450, 8, 8, GrAllocColor(255,255,255));
}
GrBox(tx[0]-10, ty[0]-10, tx[0]+10, ty[0]+10,  GrAllocColor(0, 0, 0) );
/*---------------*/
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*---------------*/
GrFilledCircle(disx, disy, 3, GrAllocColor(0, 0, 0));
Force = ON;
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
subt = tt;
dumt = t - subt;
/*----------------*/
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
if (cnt_t == ON && Warning == OFF)
{
```

```
cnt_t = 0;
Warning = OFF;
}
if (cnt_d == ON && Warning == OFF)
{
cnt_d = 0;
Warning = OFF;
}
/*   First Learning Session */
while ((dumt<215.0) && !kbhit() )
{
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
/*-----------------------------------------------*/
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*-----------------------------------------------*/
if (cnt_d == ON && Warning == OFF)
Display_learn();
/*-----------------------------------------------*/
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*-----------------------------------------------*/
if(cnt_t == ON && Warning == OFF)
{
cnt_t = 0;
Warning = OFF;
}
t = time();
tt = t;
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
dumt = t - subt;
if (Warning == 1)
GrCircle(0, 8, 8, GrAllocColor(255,255,255));
if (Warning == 2)
GrCircle(200, 8, 8, GrAllocColor(255,255,255));
if (Warning == 3)
GrCircle(450, 8, 8, GrAllocColor(255,255,255));
}
modeFlag = Eval;
Force = OFF;
/*---------------*/
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*---------------*/
GrBox(tx[0]-10, ty[0]-10,tx[0]+10,ty[0]+10,  GrAllocColor(0, 0, 0) );
GrFilledCircle(disx, disy, 3, GrAllocColor(0, 0, 0));
cycle = 0;
dirtest[0] = 0; /* re-initialize the direction array */
dirtest[1] = 10;
dirtest[2] = 5;
dirtest[3] = 7;
dirtest[4] = 3;
dirtest[5] = 9;
dirtest[6] = 14;
dirtest[7] = 11;
dirtest[8] = 6;
dirtest[9] = 1;
dirtest[10] = 12;
dirtest[11] = 15;
dirtest[12] = 2;
dirtest[13] = 4;
dirtest[14] = 8;
dirtest[15] = 13;
dirtest[16] = 0;
/*-----------------------------------------------*/
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
if (cnt_t == ON && Warning == OFF)
TakeDATA();
if (cnt_d == ON && Warning == OFF)
Display_star();
```

```
/*------------Second Evaluation----------------*/
while ((modeFlag == Eval) && !kbhit() )
{
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
/*---------------------------------------------*/
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*---------------------------------------------*/
if (cnt_d == ON && Warning == OFF)
Display_star();
/*---------------------------------------------*/
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*---------------------------------------------*/
if (cnt_t == ON && Warning == OFF)
TakeDATA();
t = time();
tt = t;
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
if (Warning == 1)
GrCircle(0, 8, 8, GrAllocColor(255,255,255));
if (Warning == 2)
GrCircle(200, 8, 8, GrAllocColor(255,255,255));
if (Warning == 3)
GrCircle(450, 8, 8, GrAllocColor(255,255,255));
}
GrBox(tx[0]-10, ty[0]-10, tx[0]+10, ty[0]+10,  GrAllocColor(0, 0, 0) );
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*---------------------------------------------*/
GrFilledCircle(disx, disy, 3, GrAllocColor(0, 0, 0));
Force = ON;
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
subt = tt;
dumt = t - subt;
/*---------------------------------------------*/
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
if (cnt_t == ON && Warning == OFF)
{
cnt_t = 0;
Warning = OFF;
}
if (cnt_d == ON && Warning == OFF)
{
cnt_d = 0;
Warning = OFF;
}
/*---------Second Learning Session-------------*/
while ((dumt<215.0) && !kbhit() )
{
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
/*---------------------------------------------*/
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*---------------------------------------------*/
if (cnt_d == ON && Warning == OFF)
Display_learn();
/*---------------------------------------------*/
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
```

```
t_old = tt;
/*-----------------------------------------------*/
if(cnt_t == ON && Warning == OFF)
{
cnt_t = 0;
Warning = OFF;
}
t = time();
tt = t;
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
dumt = t - subt;
if (Warning == 1)
GrCircle(0, 8, 8, GrAllocColor(255,255,255));
if (Warning == 2)
GrCircle(200, 8, 8, GrAllocColor(255,255,255));
if (Warning == 3)
GrCircle(450, 8, 8, GrAllocColor(255,255,255));
}
modeFlag = Eval;
Force = OFF;
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*-----------------------------------------------*/
GrBox(tx[0]-10, ty[0]-10,tx[0]+10,ty[0]+10,  GrAllocColor(0, 0, 0) );
GrFilledCircle(disx, disy, 3, GrAllocColor(0, 0, 0));
cycle = 0;
dirtest[0] = 0;
dirtest[1] = 10;
dirtest[2] = 5;
dirtest[3] = 7;
dirtest[4] = 3;
dirtest[5] = 9;
dirtest[6] = 14;
dirtest[7] = 11;
dirtest[8] = 6;
dirtest[9] = 1;
dirtest[10] = 12;
dirtest[11] = 15;
dirtest[12] = 2;
dirtest[13] = 4;
dirtest[14] = 8;
dirtest[15] = 13;
dirtest[16] = 0;
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
if (cnt_t == ON && Warning == OFF)
TakeDATA();
if (cnt_d == ON && Warning == OFF)
Display_star();
/*-------------Third Evaluation----------------*/
while ((modeFlag == Eval) && !kbhit() )
{
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
/*-----------------------------------------------*/
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*-----------------------------------------------*/
if (cnt_d == ON && Warning == OFF)
Display_star();
/*-----------------------------------------------*/
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*-----------------------------------------------*/
if (cnt_t == ON && Warning == OFF)
TakeDATA();
t = time();
tt = t;
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
```

```
if (Warning == 1)
GrCircle(0, 8, 8, GrAllocColor(255,255,255));
if (Warning == 2)
GrCircle(200, 8, 8, GrAllocColor(255,255,255));
if (Warning == 3)
GrCircle(450, 8, 8, GrAllocColor(255,255,255));
}
GrBox(tx[0]-10, ty[0]-10, tx[0]+10, ty[0]+10,  GrAllocColor(0, 0, 0) );
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
GrFilledCircle(disx, disy, 3, GrAllocColor(0, 0, 0));
Force = ON;
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
subt = tt;
dumt = t - subt;
/*---------------------------------------------*/
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
if (cnt_t == ON && Warning == OFF)
{
cnt_t = 0;
Warning = OFF;
}
if (cnt_d == ON && Warning == OFF)
{
cnt_d = 0;
Warning = OFF;
}
/*------------Third Learning Session-----------*/
subt = t;
dumt = tt - subt;
while ((dumt<215.0) && !kbhit() )
{
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
/*---------------------------------------------*/
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*---------------------------------------------*/
if (cnt_d == ON && Warning == OFF)
Display_learn();
/*---------------------------------------------*/
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*---------------------------------------------*/
if(cnt_t == ON && Warning == OFF)
{
cnt_t = 0;
Warning = OFF;
}
t = time();
tt = t;
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
dumt = t - subt;
if (Warning == 1)
GrCircle(0, 8, 8, GrAllocColor(255,255,255));
if (Warning == 2)
GrCircle(200, 8, 8, GrAllocColor(255,255,255));
if (Warning == 3)
GrCircle(450, 8, 8, GrAllocColor(255,255,255));
}
modeFlag = Eval;
Force = OFF;
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
tt = time();
dt = tt - t_old;
Fri = Dith1(dt);
```

```
Fr2 = Dith2(dt);
t_old = tt;
/*-------------------------------------------*/
GrBox(tx[0]-10, ty[0]-10,tx[0]+10,ty[0]+10,  GrAllocColor(0, 0, 0) );
GrFilledCircle(disx, disy, 3, GrAllocColor(0, 0, 0));
cycle = 0;
dirtest[0] = 0;
dirtest[1] = 10;
dirtest[2] = 5;
dirtest[3] = 7;
dirtest[4] = 3;
dirtest[5] = 9;
dirtest[6] = 14;
dirtest[7] = 11;
dirtest[8] = 6;
dirtest[9] = 1;
dirtest[10] = 12;
dirtest[11] = 15;
dirtest[12] = 2;
dirtest[13] = 4;
dirtest[14] = 8;
dirtest[15] = 13;
dirtest[16] = 0;
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
if (cnt_t == ON && Warning == OFF)
TakeDATA();
if (cnt_d == ON && Warning == OFF)
Display_star();
/*------------Fourth Evaluation----------------*/
while ((modeFlag == Eval) && !kbhit() )
{
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
/*-------------------------------------------*/
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*-------------------------------------------*/
if (cnt_d == ON && Warning == OFF)
Display_star();
/*-------------------------------------------*/
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*-------------------------------------------*/
if (cnt_t == ON && Warning == OFF)
TakeDATA();
t = time();
tt = t;
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
if (Warning == 1)
GrCircle(0, 8, 8, GrAllocColor(255,255,255));
if (Warning == 2)
GrCircle(200, 8, 8, GrAllocColor(255,255,255));
if (Warning == 3)
GrCircle(450, 8, 8, GrAllocColor(255,255,255));
}
GrBox(tx[0]-10, ty[0]-10, tx[0]+10, ty[0]+10,  GrAllocColor(0, 0, 0) );
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
GrFilledCircle(disx, disy, 3, GrAllocColor(0, 0, 0));
Force = ON;
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
subt = tt;
dumt = t - subt;
/*-------------------------------------------*/
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
if (cnt_t == ON && Warning == OFF)
```

```
{
cnt_t = 0;
Warning = OFF;
}
if (cnt_d == ON && Warning == OFF)
{
cnt_d = 0;
Warning = OFF;
}
/*------------ Forth Learning Session----------*/
while ((dumt<215.0) && !kbhit() )
{
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
/*-----------------------------------------------*/
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*-----------------------------------------------*/
if (cnt_d == ON && Warning == OFF)
Display_learn();
/*-----------------------------------------------*/
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*-----------------------------------------------*/
if(cnt_t == ON && Warning == OFF)
{
cnt_t = 0;
Warning = OFF;
}
t = time();
tt = t;
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
dumt = t - subt;
if (Warning == 1)
GrCircle(0, 8, 8, GrAllocColor(255,255,255));
if (Warning == 2)
GrCircle(200, 8, 8, GrAllocColor(255,255,255));
if (Warning == 3)
GrCircle(450, 8, 8, GrAllocColor(255,255,255));
}
modeFlag = Eval;
Force = OFF;
/*-----------------------------------------------*/
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
GrBox(tx[0]-10, ty[0]-10,tx[0]+10,ty[0]+10,  GrAllocColor(0, 0, 0) );
GrFilledCircle(disx, disy, 3, GrAllocColor(0, 0, 0));
cycle = 0;
dirtest[0] = 0;
dirtest[1] = 10;
dirtest[2] = 5;
dirtest[3] = 7;
dirtest[4] = 3;
dirtest[5] = 9;
dirtest[6] = 14;
dirtest[7] = 11;
dirtest[8] = 6;
dirtest[9] = 1;
dirtest[10] = 12;
dirtest[11] = 15;
dirtest[12] = 2;
dirtest[13] = 4;
dirtest[14] = 8;
dirtest[15] = 13;
dirtest[16] = 0;
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
if (cnt_t == ON && Warning == OFF)
TakeDATA();
if (cnt_d == ON && Warning == OFF)
Display_star();
/*----------Last Evaluation-------------------*/
```

```
while ((modeFlag == Eval) && !kbhit() )
{
if (cnt_r == ON && Warning == OFF)
RunTSUNAMI();
/*-----------------------------------------------*/
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*-----------------------------------------------*/
if (cnt_d == ON && Warning == OFF)
Display_star();
/*-----------------------------------------------*/
tt = time();
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
/*-----------------------------------------------*/
if (cnt_t == ON && Warning == OFF)
TakeDATA();
t = time();
tt = t;
dt = tt - t_old;
Fr1 = Dith1(dt);
Fr2 = Dith2(dt);
t_old = tt;
if (Warning == 1)
GrCircle(0, 8, 8, GrAllocColor(255,255,255));
if (Warning == 2)
GrCircle(200, 8, 8, GrAllocColor(255,255,255));
if (Warning == 3)
GrCircle(450, 8, 8, GrAllocColor(255,255,255));
}
GrBox(tx[0]-10, ty[0]-10,tx[0]+10,ty[0]+10,  GrAllocColor(0, 0, 0) );
GrFilledCircle(disx, disy, 3, GrAllocColor(0, 0, 0));
dac1(0.0);
dac2(0.0);
dacShutdown();                  /* disable DAC output  */
/* load the old timer ISR back without the new ISR chained on */
_go32_dpmi_set_protected_mode_interrupt_vector (TIMER, &old_handler);

/*-------------  Store data into the file  -----------------*/
 for (k=0; k<3100; k++)
fprintf(fp,"%f\t %f\t %f\t %f\t %f\t %f\t %f\t %d\n", var_t[k], var_1[k], var_2[k], var_3[k], var_4[k], var_5[k], var_6[k],
var_7[k]);
 fclose(fp);

return 0;
}

//  FUNCTIONS ..............................................................

/*  initial()
*
*  This function initializes encoder counter channels.
*  Input: Void
*  Returns: Void
*
*/
void initial(void)
{
int n = 0;
byte stat;

clear_reg(CLEAR_CHAN0);                    /* initialize          */
stat = 0x80;
while (stat==0x80)
{
stat = inportb(E_CMMD) & 0x80;
n++;
}
clear_reg(CLEAR_CHAN1);
stat = 0x80;
while (stat==0x80)
{
stat = inportb(E_CMMD) & 0x80;
n++;
}
clear_reg(CLEAR_CHAN3);
}
/*  time()
*
*  This function reads from the encoder counter channel 3 and converts
*      the counter value to absolute time.
```

```
*  Input: Void
*  Returns: time, t
*
*/
float time(void)
{
float fcnt, t;
fcnt = enc(READ_CHAN3);
t = fcnt/((float) FREQ);
if (t == 0.) t = 0.0001;
return (t);
}
/*  enc(command)
*
*  This function will read the counter channel specified by 'command'.
*      The data is read as 4 separate bytes and converted to a float
*      variable fcnt(float counter value).
*  Input: command
*  Returns: cnt
*
*/
float enc(byte command)
{
int n = 0;
float fcnt;
long int cnt;
long int cnt_0, cnt_1, cnt_2, cnt_3;
byte stat;
outportb(E_CMMD, command);      /* read counter */
stat = 0x80;
while (stat==0x80)
{
stat = inportb(E_CMMD) & 0x80;
n++;
}
cnt_0 = inportb(EBASE_0);
cnt_1 = inportb(EBASE_1);
cnt_2 = inportb(EBASE_2);
cnt_3 = inportb(EBASE_3);
cnt = cnt_0 + (cnt_1<<8) + (cnt_2<<16) + (cnt_3<<24);
fcnt = (float) cnt;
return (fcnt);
}
/*
*   Rot_calc(actuator id, fenco)
*
*   code to calculate the encoder value from encoder 1 or 2.
*
*   Argument:actuator id #, previous encoder value
*   Returns: current encoder value
*/
float Rot_calc(int act_id, float cnt_old)
{
float cnt = 0.0;
float denc;

if (act_id == 1) cnt = enc(READ_CHAN0);
else if(act_id == 2) cnt = enc(READ_CHAN1);
denc = cnt - cnt_old;
cnt = cnt + RollOver(denc)* (float) FULL_CTS;
return (cnt);
}
/*
*   RollOver(denc)
*
*   Take in the new and old encoder values and calculates
*   if the it has gone through the roll over point
*   float HALF_CTS_PER: Number of counts between rollovers
*
*   Argument:   enc is the current encoder count
*               enc_old is the previous encoder count
*
*   Returns:
*   float   -1.0 :  if it passes from neg to pos through rollover pt.
*   float   +1.0:   if it passes from pos to neg through rollover pt.
*   float    0.0:   if it does not pass through rollover pt.
*/
float RollOver(float denc)
{
if (denc < - ((float) HALF_CTS_PER) )
return (1.0);       /*   rollover from - to + : value is increasing   */
else if (denc > ((float) HALF_CTS_PER))
return (-1.0);      /*   rollover from + to - : value is decreasing   */
else
return (0.0);
}
```

```
/*  Ang_calc(int, float)
*
*  This function reads the encoder count and converts to the angles\
*       alpha1 and alpha1 where alpha1 = theta1 and alpha2 = theta1 + theta2
*  Input: link #, fenc1 or fenc2
*  Returns: alpha1 or alpha2
*
*/
float Ang_calc(int act_id, float fenc)
{
float ang, alpha;
/* bottom disk is connected to motor 2, link 1 */
/* top disk is connected to motor 1, link 2 */

ang = fenc;
alpha = ( (ang/ENC_CNT) *2*PI)/GEAR_RATIO;
if (act_id == 1)  /* calculation of alpha2 = theta1+theta2 */
alpha = alpha + PI*(120/180.0);   /* angle of link2 = 90 initially         */
else if(act_id == 2)  /* calculation of alpha1 */
alpha = -1.0*(alpha-PI*(30/180.0));
return (alpha);
}

/*
*   Dith1(svolt, dt)
*
*   Add dither
*   Argument:   volt, dt
*   Returns:
*   float: voltage value which is voltage + dynamic friction voltage
*/
float Dith1(float dt)
{
float dvolt;
static float dtd;

dtd = dtd + dt;

if (dtd < Dither_t)
dvolt = nvolt1 + 0.75;
else if (dtd >= Dither_t && dtd < Dith2_t)
dvolt = nvolt1 - 0.75;
else
{
dtd = 0.;
dvolt = nvolt1 + 0.75;
}
dvolt = -1.0*dvolt;  /* motor 2 is connected backwards */
dac2(dvolt);        /* motor 2 is connected to link1 */
return (dvolt);
}
/*
*   Dith2(svolt, dt)
*
*   Add dither
*   Argument:   volt, dt
*   Returns:
*   float: voltage value which is voltage + dynamic friction voltage
*/
float Dith2(float dt)
{
float dvolt;
static float dtd;

dtd = dtd + dt;

if (dtd < Dither_t)
dvolt = nvolt2 + 0.75;
else if (dtd >= Dither_t && dtd < Dith2_t)
dvolt = nvolt2 - 0.75;
else
{
dtd = 0.;
dvolt = nvolt2 + 0.75;
}
dac1(dvolt);    /* motor 1 is connected to link2 */
return(dvolt);
}

/*  dac1(voltage)
*
*  DAC output from channel 1:
*  This function will take the voltage value and output analogue
*       signal at the specified voltage.
*  Bipolar output set to + and - 10V range
*  Input: Voltage
```

```
*  Returns: Void
*
*/
void dac1(float voltage)
{
byte high_byte, low_byte;
float num, lownum, highnum;
long temp;
lownum = highnum = 0.;

/***  voltage limit   ***/
if (voltage > 6.8)
voltage = 6.8;
else if (voltage < -6.8)
voltage = -6.8;

/* convert voltage in two bytes in decimal  */
num = voltage*(2048.0/10.0)+2048.0;
if (num>=256.0)
{
temp = (long)num;
lownum = temp & 0xff;
low_byte = (unsigned char) lownum;
temp = (long)num;
highnum = temp>>8;
high_byte = (unsigned char) highnum;
}
else
{
low_byte = num;
high_byte = 0.0;
}
/* command voltage out  */
outportb (DCH1_LOW, low_byte);
outportb (DCH1_HIGH, high_byte);
}

/*  dac2(voltage)
*
*  DAC output from channel 2:
*  This function will take the voltage value and output analogue
*       signal at the specified voltage.
*  Bipolar output set to + and - 10V range
*  Input: Voltage
*  Returns: Void
*
*/
void dac2(float voltage)
{

byte high_byte, low_byte;
float num, lownum, highnum;
long temp;
lownum = highnum = 0.;

/***  voltage limit   ***/
if (voltage > 6.8)
voltage = 6.8;
else if (voltage < -6.8)
voltage = -6.8;

/* convert voltage in two bytes in decimal  */
num = voltage*(2048.0/10.0)+2048.0;
if (num>=256.0)
{
temp = (long)num;
lownum = temp & 0xff;
low_byte = (unsigned char) lownum;
temp = (long)num;
highnum = temp>>8;
high_byte = (unsigned char) highnum;
}
else
{
low_byte = num;
high_byte = 0.0;
}
/* command voltage out  */
outportb (DCH2_LOW, low_byte);
outportb (DCH2_HIGH, high_byte);
}

/*  dacShutdown(void)
*
*  This function will disable DAC output.
*  Input: Void
```

```
*  Returns: Void
*
*/
void dacShutdown(void)
{
outportb(DOUT_CTL, 0x00);     /* set highest bit to 0 to inhibit output */
}
/*
*   volt_cmd (int, float)
*
*   command voltage out for the specified torque value
*
*   Argument: actuator id #, specifed torque value
*
*   Returns: voltage value
*/
float volt_cmd(int act_id, float tau)
{
float volt;

volt = tau/(KT*GEAR_RATIO);
if (volt > 5.0)
volt = 5.0;
else if (volt < -5.0)
volt = -5.0;

return(volt);
}
/*--------------------------------------------------------------------------*/
/*  Display_learn()
*
*  This function displays the target location and tip location
*  information on the monitor.
*       MORE CHANGES NEEDED: TIMING and STAR PATTERN
*  Input: void
*  Returns: void
*
*/
void Display_learn(void)
{
static int n = 0;
static int m = 0;
static int oldx, oldy, no;
int dir, j;
float t, temp;
float theta;

GrBox(tx[0]-10, ty[0]-10, tx[0]+10, ty[0]+10,  GrAllocColor(0, 0, 0) );
GrFilledCircle(oldx, oldy, 3, GrAllocColor(0, 0, 0));

t = tt;

temp = t/Rch_time;
n = (int)(temp);
if (m == 60)   /* calculate next target pos. <- current act. pos. */
{
dir = rand()%8;
theta =  dir*PI/4.0;
xnext = data_x + 0.29528*cos(theta);
ynext = data_y + 0.29528*sin(theta);

if (xnext<XMAX && xnext>XMIN && ynext<YMAX && ynext>YMIN)
{
tx[1] = (int)((480*(0.0-xnext)/0.6) + 320);
ty[1] = (int)(480*(ynext - 1.66)/0.6)+240;
}
else
{
if((data_x<0.0162) && (data_x>-0.0162) && (data_y<1.6762) && (data_y>1.6438))
{
j = rand()%4;
dir = drcase0[j];
}
else
if(data_x > 0.0)
if(data_y > 1.66)
if ( (data_x<=0.0703) && (data_y<=1.7303) )
dir = 5;
else if ((data_x>0.0703) && (data_y<=1.7303))
{
j = rand()%2;
dir = drcase11[j];
}

else if ((data_x <= 0.0703) && (data_y>1.7303))
{
```

```
j = rand()%2;
dir = drcase12[j];
}
else
{
j = rand()%3;
dir = drcase1[j];
}
else
if ((data_x<=0.0703) && (data_y>=1.5897))
dir = 3;
else if ( (data_x>0.0703) && (data_y >=1.5897))
{
j = rand()%2;
dir = drcase41[j];
}
else if ((data_x <= 0.0703) && (data_y<1.5897))
{
j = rand()%2;
dir = drcase42[j];
}
else
{
j = rand()%3;
dir = drcase4[j];
}
else
if((data_y > 1.66))
if ((data_x>=-0.0703) && (data_y<=1.7303))
dir = 7;
else if ( (data_x<-0.0703) && (data_y<=1.7303) )
{
j = rand()%2;
dir = drcase21[j];
}
else if ((data_x >= -0.0703) && (data_y>1.7303))
{
j = rand()%2;
dir = drcase22[j];
}
else
{
j = rand()%3;
dir = drcase2[j];
}

else
if ((data_x>=-0.0703) && (data_y>=1.5897))
dir = 1;
else if ((data_x<-0.0703) && (data_y>=1.5897))
{
j = rand()%2;
dir = drcase31[j];
}
else if ((data_x>=-0.0703) && (data_y<1.5897))
{
j = rand()%2;
dir = drcase32[j];
}
else
{
j = rand()%3;
dir = drcase3[j];
}
theta =  dir*PI/4.0;
xnext = data_x + 0.29528*cos(theta);
ynext = data_y + 0.29528*sin(theta);

tx[1] = (int)((480*(0.0-xnext)/0.6) + 320);
ty[1] = (int)(480*(ynext - 1.66)/0.6)+240;
}
/*-----------------------------------------------*/
}
if (n > no)
{
tx[0] = tx[1];
ty[0] = ty[1];
m = 0;
}
disp_x = xnext;
disp_y = ynext;
GrBox(tx[0]-10, ty[0]-10, tx[0]+10, ty[0]+10, GrAllocColor(255, 255, 255) );
GrFilledCircle(disx, disy, 3, GrAllocColor(255, 255, 255));
oldx = disx;
oldy = disy;
m++;
```

```
no = n;
cnt_d = 0;
Warning = OFF;
}
/*  Display_star()
*
*  This function displays the target location and tip location
*  information on the monitor.
*      MORE CHANGES NEEDED: TIMING and STAR PATTERN
*  Input: void
*  Returns: void
*
*/

void Display_star(void)
{
static int n = 0;
static int m = 0;
static int oldx, oldy, no;
static int fstat;
int dir, j, fdir;
float t, temp;
float theta;
static int kk = 0;

t = time();
GrBox(tx[0]-10, ty[0]-10, tx[0]+10, ty[0]+10,  GrAllocColor(0, 0, 0) );
GrFilledCircle(oldx, oldy, 3, GrAllocColor(0, 0, 0));
temp = t/Rch_time;
n = (int)(temp);
if (m == 60)           /* calculate next target pos. <- current act. pos. */
{
dir = dirtest[cycle]%8;
fstat = (int)(dirtest[cycle]/8.0);
theta =  dir*PI/4.0;
xnext = data_x + 0.29528*cos(theta);
ynext = data_y + 0.29528*sin(theta);
if (xnext<XMAX && xnext>XMIN && ynext<YMAX && ynext>YMIN)
{
tx[1] = (int)((480*(0.0-xnext)/0.6) + 320);
ty[1] = (int)(480*(ynext - 1.66)/0.6)+240;
}
else
{
if((data_x<0.0162) && (data_x>-0.0162) && (data_y<1.6762) && (data_y>1.6438))
{
j = rand()%4;
dir = drcase0[j];
}
else
if(data_x > 0.0)
if(data_y > 1.66)
if ( (data_x<=0.0703) && (data_y<=1.7303) )
dir = 5;
else if ((data_x>0.0703) && (data_y<=1.7303))
{
j = rand()%2;
dir = drcase11[j];
}
else if ((data_x <= 0.0703) && (data_y>1.7303))
{
j = rand()%2;
dir = drcase12[j];
}
else
{
j = rand()%3;
dir = drcase1[j];
}
else
if ((data_x<=0.0703) && (data_y>=1.5897))
dir = 3;
else if ( (data_x>0.0703) && (data_y >=1.5897))
{
j = rand()%2;
dir = drcase41[j];
}
else if ((data_x <= 0.0703) && (data_y<1.5897))
{
j = rand()%2;
dir = drcase42[j];
}
else
{
j = rand()%3;
dir = drcase4[j];
```

```
}
else
if((data_y > 1.66))
if ((data_x>=-0.0703) && (data_y<=1.7303))
dir = 7;
else if ( (data_x<-0.0703) && (data_y<=1.7303) )
{
j = rand()%2;
dir = drcase21[j];
}
else if ((data_x >= -0.0703) && (data_y>1.7303))
{
j = rand()%2;
dir = drcase22[j];
}
else
{
j = rand()%3;
dir = drcase2[j];
}
else
if ((data_x>=-0.0703) && (data_y>=1.5897))
dir = 1;
else if ((data_x<-0.0703) && (data_y>=1.5897))
{
j = rand()%2;
dir = drcase31[j];
}
else if ((data_x>=-0.0703) && (data_y<1.5897))
{
j = rand()%2;
dir = drcase32[j];
}
else
{
j = rand()%3;
dir = drcase3[j];
}
}
theta =  dir*PI/4.0;
fstat = ON;
xnext = data_x + 0.29528*cos(theta);
ynext = data_y + 0.29528*sin(theta);
tx[1] = (int)((480*(0.0-xnext)/0.6) + 320);
ty[1] = (int)(480*(ynext - 1.66)/0.6)+240;
direct = dir;
fdir = dir + 8;
kk = 0;
while (kk<16)
{
if (dirtest[kk] == dir)
{
dirtest[kk] = -1;
fstat = OFF;
kk = 16;
}
else if((dirtest[kk] == fdir) && (kk<16) )
{
dirtest[kk] = -1;
fstat = ON;
kk = 16;
}
kk++;
}
while (dirtest[cycle] == -1)
  cycle++;
if(cycle >= 16)
  modeFlag = COMPLETE;
}
if (n > no)
{
tx[0] = tx[1];
ty[0] = ty[1];
m = 0;
}
disp_x = xnext;
disp_y = ynext;
Force = fstat;
GrBox(tx[0]-10, ty[0]-10, tx[0]+10, ty[0]+10, GrAllocColor(255, 255, 255) );
GrFilledCircle(disx, disy, 3, GrAllocColor(255, 255, 255));
oldx = disx;
oldy = disy;
m++;
no = n;
cnt_d = 0;
Warning = OFF;
```

```
}
/*  TakeDATA();
*
*  This function records the test data onto a file.
*  Input: void
*  Returns: void
*
*/
void TakeDATA(void)
{
static int j = 0;
var_t[j] = tt;
var_1[j] = data_x;
var_2[j] = data_y;
var_3[j] = v1;
var_4[j] = v2;
var_5[j] = disp_x;
var_6[j] = disp_y;
var_7[j] = Force;
if (j <= 4500)
j++;
cnt_t = 0;
Warning = OFF;
}
/*  RunTSUNAMI();
*
*  This function runs TSUANMI.
*  Input: void
*  Returns: void
*
*/
void RunTSUNAMI(void)
{
float fenc1, fenc2;
float disk1_angle, link2_angle, disk2_angle;
float tau1_out, tau1_cmd, tau2_out, tau2_cmd;
float t, x, y;
static float xo = 0.0;
static float yo = 0.0;
static float t_old = 0.0;
static float s1_adot = 0.0;
static float s2_adot = 0.0;
static float fenco1 = 0.0;
static float fenco2 = 0.0;
static float dth1[7] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
static float dth2[7] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
static float f_dth1[7] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
static float f_dth2[7] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
static float vx[7] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
static float vy[7] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
static float f_vx[7] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
static float f_vy[7] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};

float a1dot, a2dot, shaft1_angle, shaft2_angle;
float fric_x, fric_y;
int i = 0;
static int j = 0;
int l = 0;
float c1, c12, s1, s12, c2, s2, dt;
float denum;
float fx = 0.0;
float fy = 0.0;

t = time();
fenc1 = Rot_calc(1, fenco1);    /* Calculate encoder count */
fenc2 = Rot_calc(2, fenco2);
tt = t;
dt = t - t_old;
if (dt ==0.0)
dt = 0.00001;
disk1_angle = Ang_calc(2, fenc2);    /* Calculate angle */
disk2_angle = Ang_calc(1, fenc1);
link2_angle = disk2_angle - disk1_angle;
c1 = cos(disk1_angle);
c2 = cos(link2_angle);
c12 = cos(disk2_angle);
s1 = sin(disk1_angle);
s2 = sin(link2_angle);
s12 = sin(disk2_angle);

x = L1*c1 + L2*c12;           /* Calculate tip location from angles */
y = L1*s1 + L2*s12;
data_x = x;
data_y = y;

disx = (int)((480*(0.0-x)/0.6) + 320);
```

```
disy = (int)(480*(y - 1.66)/0.6)+240;

 /*-----------------------------------------------*/
shaft1_angle = disk1_angle*GEAR_RATIO;
shaft2_angle = disk2_angle*GEAR_RATIO;
a1dot = (shaft1_angle - s1_adot)/dt;
a2dot = (shaft2_angle - s2_adot)/dt;
denum = 1/(L1*L2*s12);

 /* Filter velocity data */
vx[j] = vx[j+1];
vy[j] = vy[j+1];
dth1[j] = dth1[j+1];
dth2[j] = dth2[j+1];
if (j == 2)                  /* Calculate tip velocity: raw velocity data */
{
vx[4] = (x - xo)/dt;
vy[4] = (y - yo)/dt;
dth1[4] = a1dot;
dth2[4] = a2dot;
}
if (j == 3)                  /* Calculate filtered velocity */
{
f_vx[4] = (float)(B_3*(double)vx[3] + B_2*(double)vx[2] + B_1*(double)vx[1] + B_0*(double)vx[0] - A_2*(double)f_vx[2] -
A_1*(double)f_vx[1] -A_0*(double)f_vx[0]);
f_vy[4] = (float)(B_3*(double)vy[3] + B_2*(double)vy[2] + B_1*(double)vy[1] + B_0*(double)vy[0] - A_2*(double)f_vy[2] -
  A_1*(double)f_vy[1] -A_0*(double)f_vy[0]);
f_dth1[4] = (float)(B_3*(double)dth1[3] + B_2*(double)dth1[2] + B_1*(double)dth1[1] + B_0*(double)dth1[0] -
A_2*(double)f_dth1[2] - A_1*(double)f_dth1[1] - A_0*(double)f_dth1[0]);
f_dth2[4] = (float)(B_3*(double)dth2[3] + B_2*(double)dth2[2] + B_1*(double)dth2[1] + B_0*(double)dth2[0] -
A_2*(double)f_dth2[2] - A_1*(double)f_dth2[1] - A_0*(double)f_dth2[0]);
}
f_vx[j] = f_vx[j+1];
f_vy[j] = f_vy[j+1];
f_dth1[j] = f_dth1[j+1];
f_dth2[j] = f_dth2[j+1];
j++;
if(j == 4)
{
j = 0;
v1 = f_vx[3];
v2 = f_vy[3];
l++;}
/*-------------------------------------------------------------------------*/
/*      Friction Compensation calculation: linear Friction Model filtered   */
if(f_dth1[3] >0.0)
{
if(f_dth1[3]> 100.0)
f_dth1[3] = 100.0;
fric_x = -0.074952+0.0000979*exp(0.14962*fabs(f_dth1[3]))+0.841276*
(1-exp(-0.77746*fabs(f_dth1[3])));
fric_x = 0.80*fric_x;
}
else if(f_dth1[3] < 0.0)
{
if(f_dth1[3]<-100.0)
f_dth1[3] = -100.0;
fric_x = -1*(3.757823-3.79467*exp(-0.0016*fabs(f_dth1[3]))+0.68793*(1-exp(-1.10855*fabs(f_dth1[3]))) );
fric_x = 0.93*fric_x;
}
else
fric_x = 0.0;
if (fric_x > 2.0)
fric_x = 2.0;
else if(fric_x < -2.0)
fric_x = -2.0;
if ( f_dth2[3] > 0.0 )
{
if(f_dth2[3] > 100.0)
f_dth2[3] = 100.0;
fric_y = -4.637+4.4585*exp(0.00323*fabs(f_dth2[3]))+1.0691*(1-exp(-1.3546*fabs(f_dth2[3])));
fric_y = 0.67*fric_y;
}
else if (f_dth2[3] < 0.0)
{
if(f_dth2[3] < -100.0)
f_dth2[3] = -100.0;
fric_y = -1*(-0.1405+0.12703*exp(0.0585*fabs(f_dth2[3]))+1.0331*(1-exp(-1.0507*fabs(f_dth2[3]))) );
fric_y = 0.67*fric_y;
}
else
fric_y = 0.0;
if (fric_y > 2.0)
fric_y = 2.0;
else if(fric_y < -2.0)
fric_y = -2.0;
```

```
if (Force == OFF)
{
fx = 0.0;
fy = 0.0;
}
else if (Force == ON)
{
fx = -0.69205*v1-0.76742*v2;
fy = -0.76742*v1 + 0.76057*v2;
}
tau1_out = L1*c1*fy - L1*s1*fx; /* Calculate motor torque to create the disturbance */
if (tau1_out == 0.0)
tau1_cmd = 0.01;
else if ((tau1_out>0.0) && (tau1_out<0.75))
tau1_cmd = 1.2335*tau1_out + 0.0803 + fric_x;
else if ((tau1_out>=0.75) && (tau1_out < 1.86))
tau1_cmd = 1.2944*tau1_out + 0.024 + fric_x;
else if ((tau1_out>=1.86) && (tau1_out < 2.89))
tau1_cmd = 1.2601*tau1_out + 0.0864 + fric_x;
else if ((tau1_out>= 2.89))
tau1_cmd = 1.2223*tau1_out + 0.1615 + fric_x;
else if ((tau1_out<0.0) && (tau1_out>-0.33))
tau1_cmd = 1.0924*tau1_out -0.0317 + fric_x;
else if ((tau1_out<=-0.33) && (tau1_out>-1.8))
tau1_cmd = 1.1982*tau1_out -0.0206 + fric_x;
else if ((tau1_out<=-1.8) && (tau1_out>-2.9))
tau1_cmd = 1.1743*tau1_out -0.065 + fric_x;
else if ((tau1_out<=-2.9))
tau1_cmd = 1.1375*tau1_out -0.1603 + fric_x;
tau2_out = L2*c12*fy - L2*s12*fx;
if(tau2_out == 0.0)
tau2_cmd = -0.165;
else if ((tau2_out>0.0) && (tau2_out<1.13))
tau2_cmd = 1.1579*tau2_out - 0.11556 + fric_y;
else if ((tau2_out>=1.13) && (tau2_out < 1.84))
tau2_cmd = 1.16248*tau2_out - 0.12025 + fric_y;
else if ((tau2_out>=1.84) && (tau2_out < 2.93))
tau2_cmd = 1.14169*tau2_out - 0.08959 + fric_y;
else if ((tau2_out>= 2.93))
tau2_cmd = 1.12814*tau2_out - 0.07322 + fric_y;
else if ((tau2_out<0.0) && (tau2_out>-0.05))
tau2_cmd = 1.85546*tau2_out -0.2 + fric_y;
else if ((tau2_out<=-0.05) && (tau2_out>-1.6))
tau2_cmd = 1.21237*tau2_out -0.17357 + fric_y;
else if ((tau2_out<=-1.6) && (tau2_out>-2.4))
tau2_cmd = 1.35332*tau2_out + 0.06027 + fric_y;
else if ((tau2_out<=-2.4))
tau2_cmd = 1.00724*tau2_out -0.74568 + fric_y;
 /*-----------------------------------------------*/
nvolt1 = volt_cmd(1, tau1_cmd);
nvolt2 = volt_cmd(2, tau2_cmd);
fenco1 = fenc1;                    /* Set the old values */
fenco2 = fenc2;
t_old = t;
xo = x;
yo = y;
s1_adot = shaft1_angle;
s2_adot = shaft2_angle;
if (i<5000)
i++;
cnt_r = OFF;                       /* Reset the old values */
Warning = OFF;
}
```

Appendix C

Simulation Code

The simulation of the adaptive motor control model during reaching motion.

```
/*
-------------------------------------------------------------------------------------------------------------------------------
 Latest version 3/12/95
  Star Trajectory :  Working Copy:  March 12, 1995
-------------------------------------------------------------------------------------------------------------------------------
*/
#include <stdio.h>
#include "mathfuns.h"
#define NRANSI
#include "nrutil.h"
#define NUMNODE 182
#define Q1MIN -5
#define Q1MAX 8
#define Q2MIN -3
#define Q2MAX 9
#define ETIME 26.4
#define LTIME 412.5
#define DT 0.0001
#define FIELD 1
#define EVAL 2
#define LEARN 4
#define Q10 0.2618
#define Q20 1.4835
#define SYSDIM  4
#define NETOUT  16
#define ADAPTIVE
#define NSTATES SYSDIM+NETOUT*NUMNODE

float q1d, q2d, q1dd, q2dd, q1ddd, q2ddd, tau[2], s1, s2;
float tor1, tor2, ka1, ka2, ka3, ka4, is1, is2;
float actlev[NUMNODE], *actref = actlev, *act;
float *oldx, *newx;
float   ka1, kb, beta;
float lambda[2][2], kd[2][2], kp[2][2], ehat[2][2];
float Hinv[2][2], C[2][2], hhat[2][2], chat[2][4], H[2][2];
short modeFlag;
double nrand(double);
/* Maki's global variables */
short leg;
short star;
static float dptx[260];
static float dpty[260];

void odeint(float ystart[], int nvar, float x1, float x2,
float eps, float h1, float hmin, int *nok, int *nbad,
void (*derivs)(float, float [], float []),
void (*rkqs)(float [], float [], int, float *, float, float,
float [], float *, float *, void (*)(float, float [], float [])),
float, void (*savvars)(float,float[]));
void rkck(float y[], float dydx[], int n, float x, float h,
float yout[], float yerr[], void (*derivs)(float, float [], float []));
void rkqs(float y[], float dydx[], int n, float *x,
float htry, float eps, float yscal[], float *hdid, float *hnext,
void (*derivs)(float, float [], float []));
void rhs(float, float[], float[]);
void mySave(float, float[]);
void desTraj(int, float, float*, float*, float*);
float drand48(void);
```

```
float ffloor(float);

float x0[NSTATES];
FILE *eFile, *eFile1, *eFile2, *eFile3, *eFile4;
FILE *nFile, *nFile1, *nFile2, *nFile3, *nFile4;

main()
{
  short ii,que;
  float eps = 1.0e-05, dt=0.0001;
  float piglet, theta, xx, xxx, yy, yyy;
  int nok, count, nbad;
  double pooh;
  int tiger, kk;
  int i=0;
  int dirwalk;
  oldx = (float*) malloc( NSTATES * sizeof( float));
  newx = (float*) malloc( NSTATES * sizeof( float));

  dptx[0] = 0.259725;
  dpty[0] = 0.420247;
  dptx[1] = 0.33043568;
  dpty[1] = 0.490958;
  xx = dptx[0] - 0.12;
  xxx = dptx[0] + 0.12;
  yy = dpty[0] - 0.12;
  yyy = dpty[0] + 0.12;

printf(" %f\t %f\n %f\t %f\n\n ", xx, yy, xxx, yyy);
  que = 0;
  while (que< 255)   /*** # of counts ***/
{
piglet = 8.*drand48();
dirwalk = (int) ffloor(piglet);
theta = PI*dirwalk/4;
dptx[que+1] = dptx[que] + 0.1*cos(theta);
dpty[que+1] = dpty[que] + 0.1*sin(theta);
if ((dptx[que+1]<xxx) && (dptx[que+1]>xx) && (dpty[que+1]<yyy) && (dpty[que+1]>yy))
que++;
}

for (que=0; que<255; que++)
{
printf("%f\t%f\n",dptx[que], dpty[que]);
}

printf("ok!\n");fflush(stdout);

  for (ii = 0; ii < NSTATES; ii++) x0[ii] = 0.;

  lambda[0][0] = 6.4968;
  lambda[0][1] = 0.0637;
  lambda[1][0] = 0.0637;
  lambda[1][1] = 6.6667;
  kd[0][0]= 2.3;
  kd[0][1] = 0.9;
  kd[1][0] = 0.9;
  kd[1][1] = 2.4;
  kp[0][0] = 15.0;
  kp[0][1] = 6.0;
  kp[1][0] = 6.0;
  kp[1][1] = 16.0;
  ka1 = 0.01;
  ka2 = 0.01;
  ka3 = 0.04;
  ka4 = 0.04;

  printf("ok!\n");fflush(stdout);
  nFile = fopen("null0.dat","w");
  nFile1 = fopen("null1.dat","w");
  nFile2 = fopen("null2.dat","w");
  nFile3 = fopen("null3.dat","w");
  nFile4 = fopen("null4.dat","w");
  eFile = fopen("robot0.dat","w");
  eFile1 = fopen("robot1.dat","w");
  eFile2 = fopen("robot2.dat","w");
  eFile3 = fopen("robot3.dat","w");
  eFile4 = fopen("robot4.dat","w");

  /*       Show initial performance in Field    */
  x0[0] = Q10;
  x0[1] = Q20;
  x0[2] = 0.;
  x0[3] = 0.;
  star = 0;
  modeFlag = EVAL+FIELD;
```

```
  odeint(x0, NSTATES, 0., ETIME, eps, dt, 1.0e-05, &nok, &nbad, rhs, rkqs, .05, mySave);

  x0[0] = Q10;
  x0[1] = Q20;
  x0[2] = 0.;
  x0[3] = 0.;
  star = 1;
  modeFlag = EVAL;
  odeint(x0, NSTATES, 0., ETIME, eps, dt, 1.0e-05, &nok, &nbad, rhs, rkqs, .05, mySave);

   x0[0] = Q10;
   x0[1] = Q20;
   x0[2] = 0.;
   x0[3] = 0.;
  for (kk = 0; kk < 4; kk++) {

  modeFlag = LEARN+FIELD;
    odeint(x0, NSTATES, 0., LTIME, eps, dt, 1.0e-05, &nok, &nbad, rhs, rkqs, .05, mySave);

  x0[0] = Q10;
  x0[1] = Q20;
x0[2] = 0.;
x0[3] = 0.;
  star++;
  modeFlag = EVAL+FIELD;
    odeint(x0, NSTATES, 0., ETIME, eps, dt, 1.0e-05, &nok, &nbad, rhs, rkqs, .05, mySave);

  x0[0] = Q10;
  x0[1] = Q20;
  x0[2] = 0.;
 x0[3] = 0.;
  star++;
  modeFlag = EVAL;
    odeint(x0, NSTATES, 0., ETIME, eps, dt, 1.0e-05, &nok, &nbad, rhs, rkqs, .05, mySave);

  x0[0] = Q10;
  x0[1] = Q20;
 x0[2] = 0.;
 x0[3] = 0.;
  }
}

void mySave(float t, float x[])
{
  static float qa,qad,qadd,qb,qbd,qbdd;
  int i,que;
  float tt;
  if (!(modeFlag&EVAL)) return;
  tt = fmod(t, 3.3);
  if (tt>1.65) return;
  if (star == 0)
  {
  fprintf(oFile,"%lf\t",t);
  for (i=0;i<4;i++) fprintf(oFile,"%lf\t",x[i]);
  desTraj(1,t,&qa,&qad,&qadd);
  desTraj(2,t,&qb,&qbd,&qbdd);
  fprintf(oFile,"%lf\t",qa);
  fprintf(oFile,"%lf\t",qb);
  fprintf(oFile,"%lf\t",qad);
  fprintf(oFile,"%lf\t",qbd);
  fprintf(oFile,"%d\t",leg);
  fprintf(oFile,"\n");
  fflush(oFile);
  }
  else if (star == 2)
  {
  fprintf(oFile1,"%lf\t",t);
  for (i=0;i<4;i++) fprintf(oFile1,"%lf\t",x[i]);
  desTraj(1,t,&qa,&qad,&qadd);
  desTraj(2,t,&qb,&qbd,&qbdd);
  fprintf(oFile1,"%lf\t",qa);
  fprintf(oFile1,"%lf\t",qb);
  fprintf(oFile1,"%lf\t",qad);
  fprintf(oFile1,"%lf\t",qbd);
  fprintf(oFile1,"%d\t",leg);
  fprintf(oFile1,"\n");
  fflush(oFile1);
  }
  else if (star == 4)
  {
  fprintf(oFile2,"%lf\t",t);
  for (i=0;i<4;i++) fprintf(oFile2,"%lf\t",x[i]);
  desTraj(1,t,&qa,&qad,&qadd);
  desTraj(2,t,&qb,&qbd,&qbdd);
  fprintf(oFile2,"%lf\t",qa);
  fprintf(oFile2,"%lf\t",qb);
```

```
  fprintf(oFile2,"%lf\t",qad);
  fprintf(oFile2,"%lf\t",qbd);
  fprintf(oFile2,"%d\t",leg);
  fprintf(oFile2,"\n");
  fflush(oFile2);
  }
  else if (star == 6)
  {
  fprintf(oFile3,"%lf\t",t);
  for (i=0;i<4;i++) fprintf(oFile3,"%lf\t",x[i]);
  desTraj(1,t,&qa,&qad,&qadd);
  desTraj(2,t,&qb,&qbd,&qbdd);
  fprintf(oFile3,"%lf\t",qa);
  fprintf(oFile3,"%lf\t",qb);
  fprintf(oFile3,"%lf\t",qad);
  fprintf(oFile3,"%lf\t",qbd);
  fprintf(oFile3,"%d\t",leg);
  fprintf(oFile3,"\n");
  fflush(oFile3);
  }
  else if (star == 8)
  {
  fprintf(oFile4,"%lf\t",t);
  for (i=0;i<4;i++) fprintf(oFile4,"%lf\t",x[i]);
  desTraj(1,t,&qa,&qad,&qadd);
  desTraj(2,t,&qb,&qbd,&qbdd);
  fprintf(oFile4,"%lf\t",qa);
  fprintf(oFile4,"%lf\t",qb);
  fprintf(oFile4,"%lf\t",qad);
  fprintf(oFile4,"%lf\t",qbd);
  fprintf(oFile4,"%d\t",leg);
  fprintf(oFile4,"\n");
  fflush(oFile4);
  }
  else if (star == 1)
{
  fprintf(nFile,"%lf\t",t);
  for (i=0;i<4;i++) fprintf(nFile,"%lf\t",x[i]);
  desTraj(1,t,&qa,&qad,&qadd);
  desTraj(2,t,&qb,&qbd,&qbdd);
  fprintf(nFile,"%lf\t",qa);
  fprintf(nFile,"%lf\t",qb);
  fprintf(nFile,"%lf\t",qad);
  fprintf(nFile,"%lf\t",qbd);
  fprintf(nFile,"%d\t",leg);
  fprintf(nFile,"\n");
  fflush(nFile);
  }
  else if (star == 3)
{
  fprintf(nFile1,"%lf\t",t);
  for (i=0;i<4;i++) fprintf(nFile1,"%lf\t",x[i]);
  desTraj(1,t,&qa,&qad,&qadd);
  desTraj(2,t,&qb,&qbd,&qbdd);
  fprintf(nFile1,"%lf\t",qa);
  fprintf(nFile1,"%lf\t",qb);
  fprintf(nFile1,"%lf\t",qad);
  fprintf(nFile1,"%lf\t",qbd);
  fprintf(nFile1,"%d\t",leg);
  fprintf(nFile1,"\n");
  fflush(nFile1);
  }
  else if (star == 5)
{
  fprintf(nFile2,"%lf\t",t);
  for (i=0;i<4;i++) fprintf(nFile2,"%lf\t",x[i]);
  desTraj(1,t,&qa,&qad,&qadd);
  desTraj(2,t,&qb,&qbd,&qbdd);
  fprintf(nFile2,"%lf\t",qa);
  fprintf(nFile2,"%lf\t",qb);
  fprintf(nFile2,"%lf\t",qad);
  fprintf(nFile2,"%lf\t",qbd);
  fprintf(nFile2,"%d\t",leg);
  fprintf(nFile2,"\n");
  fflush(nFile2);
  }
  else if (star == 7)
{
  fprintf(nFile3,"%lf\t",t);
  for (i=0;i<4;i++) fprintf(nFile3,"%lf\t",x[i]);
  desTraj(1,t,&qa,&qad,&qadd);
  desTraj(2,t,&qb,&qbd,&qbdd);
  fprintf(nFile3,"%lf\t",qa);
  fprintf(nFile3,"%lf\t",qb);
  fprintf(nFile3,"%lf\t",qad);
  fprintf(nFile3,"%lf\t",qbd);
```

```
  fprintf(nFile3,"%d\t",leg);
  fprintf(nFile3,"\n");
  fflush(nFile3);
  }
  else if (star == 9)
{
  fprintf(nFile4,"%lf\t",t);
  for (i=0;i<4;i++) fprintf(nFile4,"%lf\t",x[i]);
  desTraj(1,t,&qa,&qad,&qadd);
  desTraj(2,t,&qb,&qbd,&qbdd);
  fprintf(nFile4,"%lf\t",qa);
  fprintf(nFile4,"%lf\t",qb);
  fprintf(nFile4,"%lf\t",qad);
  fprintf(nFile4,"%lf\t",qbd);
  fprintf(nFile4,"%d\t",leg);
  fprintf(nFile4,"\n");
  fflush(nFile4);
  }
}

void desTraj(int joint, float t, float *q, float *qd, float *qdd)
{
  double tt, tb;
  float mxf, myf, dummy;
  float mmx, mmy, dx, dy, ddx, ddy;
  float q1, q2, a2, dq1, dq2, ddq1, ddq2;
  float picor, rang, rsq, denum;
  float ele1, ele2, ele3, ele4;
  float dj1, dj2, dj3, dj4;
  float term1, term2;
  float mx0 = 0.259725;
  float my0 = 0.420247;
  short count, direct;
  int quee = 0;
  if (modeFlag&EVAL) /*  if modeFlag == EVAL, random star trajectory */
{
     /* dummy = t;
   while (dummy > 26.4) dummy = dummy - 26.4;
   que = ffloor(dummy/3.3);*/
/* direct = dpt[count]; */    /*activate this to produce random star shape*/
count = ffloor(t/3.3);
     direct= count%8;
leg = direct+1.;
tt = fmod(t,3.3);
/*----------- x and y coordinates ----------*/
if(direct < 1) {
  mxf = 0.359725;
  myf = 0.420247;
  }
else if (direct < 2) {
  mxf = 0.33043568;
  myf = 0.49095768;
  }
else if (direct < 3) {
mxf = 0.259725;
  myf = 0.520247;
  }
else if (direct < 4) {
  mxf = 0.18901432;
  myf = 0.49095768;
  }
  else if (direct < 5){
  mxf = 0.159725;
  myf = 0.420247;
  }
else if (direct < 6){
  mxf = 0.18901432;
  myf = 0.34953632;
  }
  else if (direct < 7){
  mxf = 0.259725;
  myf = 0.320247;
  }
  else {
  mxf = 0.33043568;
  myf = 0.34953632;
  }
/*-------- Calculation of Trajectory -------*/
  if (tt<3.3 && tt>2.3) {
    mmx = mx0;
    dx = 0.0;
    ddx = 0.0;
    mmy = my0;
    dy = 0.0;
    ddy = 0.0;
 rang = atan2(mmy,mmx);
```

```
  rsq = mmx*mmx + mmy*mmy;
  q2 = acos( (rsq-0.2245)/0.2244);
  q1 = rang-SGN(q2)*acos( (rsq + 0.1089 - 0.1156)/(0.66*sqrt(rsq)) );
    dq1 = 0.0;
    dq2 = 0.0;
    ddq1 = 0.0;
    ddq2 = 0.0;
    }
   else if (tt>1.65) {
  tb = (tt-1.65)/0.65;
  mmx = mxf - (mxf-mx0)*( 6*pow(tb, 5) - 15*pow(tb, 4) + 10*pow(tb, 3) );
  mmy = myf - (myf-my0)*( 6*pow(tb, 5) - 15*pow(tb, 4) + 10*pow(tb, 3) );
  dx = -(mxf-mx0)*( 30*pow(tb, 4) - 60*pow(tb, 3) + 30*pow(tb, 2) )/0.65;
  ddx = -(mxf-mx0)*( 120*pow(tb, 3) - 180*pow(tb, 2) + 60*tb )/(0.65*0.65);
  dy = -(myf-my0)*( 30*pow(tb, 4) - 60*pow(tb, 3) + 30*pow(tb, 2) )/0.65;
  ddy = -(myf-my0)*( 120*pow(tb, 3) - 180*pow(tb, 2) + 60*tb )/(0.65*0.65);
   rang = atan2(mmy,mmx);
  rsq = mmx*mmx + mmy*mmy;
  q2 = acos( (rsq-0.2245)/0.2244);
  q1 = rang-SGN(q2)*acos( (rsq + 0.1089 - 0.1156)/(0.66*sqrt(rsq)) );
  a2 = q1 + q2;

  ele1 = 0.34*cos(a2);
  ele2 = 0.34*sin(a2);
  ele3 = -0.33*cos(q1) - 0.34*cos(a2);
  ele4 = -0.33*sin(q1) - 0.34*sin(a2);
  denum = 1/(0.1122*sin(q2));
  dq1 = denum * (ele1*dx + ele2*dy);
  dq2 = denum * (ele3*dx + ele4*dy);
/* accelerations */
  term1 = denum * (ele1*ddx + ele2*ddy);
  term2 = denum * (ele3*ddx + ele4*ddy);
  dj1 = -0.33*cos(q1)*dq1 - 0.34*cos(a2)*(dq1 + dq2);
  dj2 = -0.34*cos(a2)*(dq1 + dq2);
  dj3 = -0.33*sin(q1)*dq1 - 0.34*sin(a2)*(dq1 + dq2);
  dj4 = -0.34*sin(a2)*(dq1 + dq2);
  ddq1 = term1-denum*((ele1*dj1 + ele2*dj3)*dq1 + (ele1*dj2 + ele2*dj4)*dq2);
  ddq2 = term2-denum*((ele3*dj1 + ele4*dj3)*dq1 + (ele3*dj2 + ele4*dj4)*dq2);
}
  else if (tt>0.65) {
  mmx = mxf;
  mmy = myf;
  dx = 0.0;
  ddx = 0.0;
  dy = 0.0;
  ddy = 0.0;
rang = atan2(mmy,mmx);
  rsq = mmx*mmx + mmy*mmy;
  q2 = acos( (rsq-0.2245)/0.2244);
  q1 = rang-SGN(q2)*acos( (rsq + 0.1089 - 0.1156)/(0.66*sqrt(rsq)) );
 a2 = q1 + q2;
  dq1 = 0.0;
  ddq1 = 0.0;
  dq2 = 0.0;
  ddq2 = 0.0;
}
  else {
  tb = tt/0.65;
  mmx = mx0 + (mxf-mx0)*( 6*pow(tb, 5) - 15*pow(tb, 4) + 10*pow(tb, 3) );
  mmy = my0 + (myf-my0)*( 6*pow(tb, 5) - 15*pow(tb, 4) + 10*pow(tb, 3) );
  dx = (mxf-mx0)*( 30*pow(tb, 4) - 60*pow(tb, 3) + 30*pow(tb, 2) )/0.65;
  ddx = (mxf-mx0)*( 120*pow(tb, 3) - 180*pow(tb, 2) + 60*tb )/(0.65*0.65);
  dy = (myf-my0)*( 30*pow(tb, 4) - 60*pow(tb, 3) + 30*pow(tb, 2) )/0.65;
  ddy = (myf-my0)*( 120*pow(tb, 3) - 180*pow(tb, 2) + 60*tb )/(0.65*0.65);

   rang = atan2(mmy,mmx);
  rsq = mmx*mmx + mmy*mmy;
  q2 = acos( (rsq-0.2245)/0.2244);
  q1 = rang-SGN(q2)*acos( (rsq + 0.1089 - 0.1156)/(0.66*sqrt(rsq)) );
 a2 = q2 + q1;
  ele1 = 0.34*cos(a2);
  ele2 = 0.34*sin(a2);
  ele3 = -0.33*cos(q1) - 0.34*cos(a2);
  ele4 = -0.33*sin(q1) - 0.34*sin(a2);
  denum = 1/(0.1122*sin(q2));
  dq1 = denum * (ele1*dx + ele2*dy);
  dq2 = denum * (ele3*dx + ele4*dy);
/* accelerations */
  term1 = denum * (ele1*ddx + ele2*ddy);
  term2 = denum * (ele3*ddx + ele4*ddy);
  dj1 =  -0.33*cos(q1)*dq1 - 0.34*cos(a2)*(dq1 + dq2);
  dj2 =  -0.34*cos(a2)*(dq1 + dq2);
  dj3 = -0.33*sin(q1)*dq1 - 0.34*sin(a2)*(dq1 + dq2);
  dj4 = -0.34*sin(a2)*(dq1 + dq2);
 ddq1 = term1-denum*((ele1*dj1 + ele2*dj3)*dq1 + (ele1*dj2 + ele2*dj4)*dq2);
  ddq2 = term2-denum*((ele3*dj1 + ele4*dj3)*dq1 + (ele3*dj2 + ele4*dj4)*dq2);
```

```
}
}

  if (modeFlag&LEARN)
{
quee = (int) ffloor(t/1.65);
 tt = fmod(t,1.65);
/*-------- Calculation of Trajectory -------*/
  if (tt<1.65 && tt>0.65) {
  mmx = dptx[quee+1];
mmy = dpty[quee+1];
  dx = 0.0;
  ddx = 0.0;
  dy = 0.0;
  ddy = 0.0;
    rang = atan2(mmy,mmx);
  rsq = mmx*mmx + mmy*mmy;
  q2 = acos( (rsq-0.2245)/0.2244);
  q1 = rang-SGN(q2)*acos( (rsq + 0.1089 - 0.1156)/(0.66*sqrt(rsq)) );
 a2 = q1 + q2;
  dq1 = 0.0;
  ddq1 = 0.0;
  dq2 = 0.0;
  ddq2 = 0.0;
  }
else {
  tb = tt/0.65;
  mmx = dptx[quee] + (dptx[quee+1]-dptx[quee])*( 6*pow(tb, 5) - 15*pow(tb, 4) + 10*pow(tb, 3) );
  mmy = dpty[quee] + (dpty[quee+1]-dpty[quee])*( 6*pow(tb, 5) - 15*pow(tb,4) + 10*pow(tb, 3) );
  dx = (dptx[quee+1] - dptx[quee])*( 30*pow(tb, 4) - 60*pow(tb, 3) + 30*pow(tb, 2) )/0.65;
  ddx = (dptx[quee+1]- dptx[quee])*( 120*pow(tb, 3) - 180*pow(tb, 2) + 60*tb )/(0.65*0.65);
  dy = (dpty[quee+1]- dpty[quee])*( 30*pow(tb, 4) - 60*pow(tb, 3) + 30*pow(tb, 2) )/0.65;
  ddy = (dpty[quee+1]- dpty[quee])*( 120*pow(tb, 3) - 180*pow(tb, 2) + 60*tb )/(0.65*0.65);
rang = atan2(mmy,mmx);
  rsq = mmx*mmx + mmy*mmy;
  q2 = acos( (rsq-0.2245)/0.2244);
  q1 = rang-SGN(q2)*acos( (rsq + 0.1089 - 0.1156)/(0.66*sqrt(rsq)) );
  a2 = q2 + q1;
  ele1 = 0.34*cos(a2);
  ele2 = 0.34*sin(a2);
  ele3 = -0.33*cos(q1) - 0.34*cos(a2);
  ele4 = -0.33*sin(q1) - 0.34*sin(a2);
  denum = 1/(0.1122*sin(q2));
  dq1 = denum * (ele1*dx + ele2*dy);
  dq2 = denum * (ele3*dx + ele4*dy);
/* accelerations */
  term1 = denum * (ele1*ddx + ele2*ddy);
  term2 = denum * (ele3*ddx + ele4*ddy);
 dj1 =  -0.33*cos(q1)*dq1 - 0.34*cos(a2)*(dq1 + dq2);
  dj2 =  -0.34*cos(a2)*(dq1 + dq2);
  dj3 = -0.33*sin(q1)*dq1 - 0.34*sin(a2)*(dq1 + dq2);
  dj4 = -0.34*sin(a2)*(dq1 + dq2);
  ddq1 = term1-denum*( (ele1*dj1 + ele2*dj3)*dq1 + (ele1*dj2 + ele2*dj4)*dq2);
  ddq2 = term2-denum*( (ele3*dj1 + ele4*dj3)*dq1 + (ele3*dj2 + ele4*dj4)*dq2);
}
}

  if (joint == 1) {
*q  =  q1;
*qd = dq1;
*qdd = ddq1;
      } else if (joint == 2) {
*q  = q2;
*qd = dq2;
*qdd = ddq2;
     } else if (joint == 3) {
*q  = mmx;
*qd = dx;
*qdd = ddx;
     } else if (joint == 4) {
*q  = mmy;
*qd = dy;
*qdd = ddy;
     }
}

void rhs(float t, float *state, float *deriv)
{
/* Parameters for controller        */
static float trunc=2.0,sigma=2.0, delta=.5;
static float r2, dr1, dr12, dr2, dr22, dr3, dr32, dr4, dr42;
static float a1 = 0.306, a2 = 0.074, a3 = 0.095, a4 = 0.0, det;
static float qt1, qt2, qt1d, qt2d, qrd1, qrd2, qrdd1, qrdd2;
static float reg, k, g1, g2, g3;
/* Parameters for plant dynamics */
static float    q1, q2, qd1, qd2, sn2, cn2, h;
```

```
static float q3, sn1, cn1, sn12, cn12;
static float v[2];
static float*  cptr[16];
/* Bookeeping, etc */
register ii, jj, kk;
float *x = state, *dx = deriv;
/* Initialize variables */
for (ii = 0; ii < 2; ii++) {
hhat[ii][0]=hhat[ii][1]= 0.;
chat[ii][0]=chat[ii][1]=0.;
chat[ii][2]=chat[ii][3]=0.;
ehat[ii][0]=ehat[ii][1]= 0.;
}
q1  = *x++;
q2  = *x++;
qd1 = *x++;
qd2 = *x++;
/*
if ( (ABS(q1)>PI)||(ABS(q2)>PI)||(ABS(qd1)>5.*PI)||(ABS(qd2)>5.*PI)) {
printf("General failure: %lf\t%lf\t%lf\t%lf\n",
t,q1,q2,qd1,qd2);
exit(1);
}
*/
/*------------ Compute Desired Trajectory ------------*/
    desTraj(1,t,&q1d,&q1dd,&q1ddd);
   desTraj(2,t,&q2d,&q2dd,&q2ddd);

/*-----------Compute tracking error ----------------*/
qt1  = q1 - q1d;
qt2  = q2 - q2d;
qt1d = qd1 - q1dd;
qt2d = qd2 - q2dd;

/*----- Compute adaptive function estimates ---------*/
#ifdef ADAPTIVE
act = actref;
for (ii = 0; ii < 16; ii++) cptr[ii] = x+ii*NUMNODE;
for (ii = Q1MIN; ii <= Q1MAX; ii++) {
dr1 = (q1 - ii*delta);
for (jj = Q2MIN; jj <= Q2MAX; jj++, act++) {
dr2     = (q2 - jj*delta);
if ((ABS(dr1)>trunc)||(ABS(dr2)>trunc)) {
*act = 0.;
for (kk = 0; kk < 16; kk++) cptr[kk]++;
continue;
}
dr12 = dr1*dr1;
r2     = dr12 + dr2*dr2;
*act  =    exp(-r2*sigma);
hhat[0][0] += (*cptr[0]++)*(*act);
hhat[0][1] += (*cptr[1]++)*(*act);
hhat[1][0] += (*cptr[2]++)*(*act);
hhat[1][1] += (*cptr[3]++)*(*act);
chat[0][0] += (*cptr[4]++)*(*act);
chat[0][1] += (*cptr[5]++)*(*act);
chat[0][2] += (*cptr[6]++)*(*act);
chat[0][3] += (*cptr[7]++)*(*act);
chat[1][0] += (*cptr[8]++)*(*act);
chat[1][1] += (*cptr[9]++)*(*act);
chat[1][2] += (*cptr[10]++)*(*act);
chat[1][3] += (*cptr[11]++)*(*act);
ehat[0][0] += (*cptr[12]++)*(*act);
ehat[0][1] += (*cptr[13]++)*(*act);
ehat[1][0] += (*cptr[14]++)*(*act);
ehat[1][1] += (*cptr[15]++)*(*act);
}
}
#endif

/* Adaptive */
qrd1 = q1dd-lambda[0][0]*qt1 - lambda[0][1]*qt2;
qrd2 = q2dd-lambda[1][0]*qt1 - lambda[1][1]*qt2;
qrdd1 = q1ddd-lambda[0][0]*qt1d - lambda[0][1]*qt2d;
qrdd2 = q2ddd-lambda[1][0]*qt1d - lambda[1][1]*qt2d;
s1 = lambda[0][0]*qt1 + lambda[0][1]*qt2 + qt1d;
s2 = lambda[1][0]*qt1 + lambda[1][1]*qt2 + qt2d;
tor1 = tau[0] = -kd[0][0]*s1 - kd[0][1]*s2 +
  hhat[0][0]*qrdd1+hhat[0][1]*qrdd2 +
  chat[0][0]*qd1*qrd1 + chat[0][1]*qd1*qrd2 +
  chat[0][2]*qd2*qrd1 + chat[0][3]*qd2*qrd2 +
  ehat[0][0]*qd1 + ehat[0][1]*qd2;
tor2 = tau[1] = -kd[1][0]*s1 - kd[1][1]*s2 +
  hhat[1][0]*qrdd1+hhat[1][1]*qrdd2 +
  chat[1][0]*qd1*qrd1 + chat[1][1]*qd1*qrd2 +
  chat[1][2]*qd2*qrd1 + chat[1][3]*qd2*qrd2 +
```

```
  ehat[1][0]*qd1 + ehat[1][1]*qd2;

/* PD */
#ifdef PD
tor1 = tau[0] = -kd[0][0]*qt1d - kd[0][1]*qt2d - kp[0][0]*qt1 - kp[0][1]*qt2;
tor2 = tau[1] = -kd[1][0]*qt1d - kd[1][1]*qt2d - kp[1][0]*qt1 - kp[1][1]*qt2;
#endif

/* Compute robot dynamics */
q3 = q1 + q2;
sn1 = sin(q1); cn1 = cos(q1);
sn2 = sin(q2); cn2 = cos(q2);
sn12 = sin(q3); cn12 = cos(q3);

h=a3*sn2-a4*cn2;
H[0][0] = a1 + 2.*a3*cn2 + 2.*a4*sn2;
H[0][1] = H[1][0] = a2 + a3*cn2 + a4*sn2;
H[1][1] = a2;
MatInv_2(H, Hinv, &det);
if (ABS(det)<0.005) printf("Warning: H is ill-conditioned!\n");

C[0][0] = -h*qd2;
C[0][1] = -h*(qd1+qd2);
C[1][0] = h*qd1;
C[1][1] = 0.;

v[0] = s1;
v[1] = s2;

    Mat2Vec(C, v, v);
    VecSub(tau, v, 2, tau);
  if(modeFlag&FIELD) {
tau[0] += 2.3*qd1 + 0.64*qd2;
tau[1] += -0.9*qd1 - 1.54*qd2;
}

    Mat2Vec(Hinv, tau, tau);
/* Compute dt, dx, dv */
*dx++ = qd1; /* dq1 */
*dx++ = qd2; /* dq2 */
*dx++ = qrdd1+tau[0]; /* ddq1 */
*dx++ = qrdd2+tau[1]; /* ddq1 */

/* Compute dfhat, dghat */
#ifdef ADAPTIVE
act = actref;
for (ii = 0; ii < 16; ii++) cptr[ii] = dx+ii*NUMNODE;
        if (ABS(s1)<.01) s1=0.;
        if (ABS(s2)<.01) s2=0.;
         if (modeFlag&LEARN) {g1=ka1;g2=ka2;g3=ka3;}
         else {g1=0.; g2=0.;g3=0.;}

        for (ii = 0; ii < (NUMNODE); ii++) {
reg = (*act++);
if (reg==0) {
   for (jj = 0; jj < 16; jj++) *cptr[jj]++ = 0.;
  continue;
}
*cptr[0]++ = -g1*s1*qrdd1*reg;
*cptr[1]++ = -g1*s1*qrdd2*reg;
*cptr[2]++ = -g2*s2*qrdd1*reg;
*cptr[3]++ = -g2*s2*qrdd2*reg;
*cptr[4]++ = -g2*s1*qd1*qrd1*reg;
*cptr[5]++ = -g2*s1*qd1*qrd2*reg;
*cptr[6]++ = -g2*s1*qd2*qrd1*reg;
*cptr[7]++ = -g2*s1*qd2*qrd2*reg;
*cptr[8]++ = -g2*s2*qd1*qrd1*reg;
*cptr[9]++ = -g2*s2*qd1*qrd2*reg;
*cptr[10]++ = -g2*s2*qd2*qrd1*reg;
*cptr[11]++ = -g2*s2*qd2*qrd2*reg;
*cptr[12]++ = -g3*s1*qd1*reg;
*cptr[13]++ = -g3*s1*qd2*reg;
*cptr[14]++ = -g3*s2*qd1*reg;
*cptr[15]++ = -g3*s2*qd2*reg;
}
#endif
}

double nrand(double scale) {
double  temp = 0;
temp = (double) random() - (1<<30);
return( (temp/(1<<30))*scale );
}
```

Bibliography

[1] "International space station: Becoming a reality", *Aerospace America*, 37(7), S1-S15, July 1999.

[2] Abend, W.K., Bizz, E., and Morasso, P., "Human arm trajecotry formation", *Brain*, 105, 331-348 1996.

[3] Adams, John, and Payandeh, Shahram, "Meethods for Low-Velocity Friction Compensation: Theory and Experimental Study", *Journal of Robotic Systems*, 13(6), 1996.

[4] Apgar, John, "Design documentation for the universal force reflecting hand controller",B.S. Thesis, Department of Mechanical Engineering, Massachusettes Institute of Technology, 1987.

[5] Armstrong, B., "Friction: Experimental Determination, Modeling and Compensation", *IEEE Proc. of the Inter. Conf. on Robotics and Automation*, 3, 1422-1427, 1988.

[6] Bernstein, N.A., *The coordination and regulation of movements*, New York, Pergamon, 1967.

[7] Bizzi, Emilio, Mussa-Ivaldi, Ferdinand A., Giszter, Simon, "Computations underlying the execution of movement: a biological perspective", *Science*, 253, 287-291.

[8] Caminiti, R., *The Journal of Neuroscience*, 10, 2039, 1990.

[9] Canudas, C., Noel, P., and Brogliato, B., "Adaptive Friction Compensation in Robot Manipulators: Low Velocities", *The International Journal of Robotics Research*, 10(3), 189-199, 1991.

[10] Coiton, Y., Gilhodes, J. C., Velay, J. L., and Roll, J. P., "A neural network model of the intersensory coordination involved in goal-directed movements", *Biological Cybernetics*, 66, 167-176, 1991.

[11] Craig, John J., *Introduction to robotics: mechanics and control-second edition*, Addison Wesley, 1989.

[12] Feldman, A., Serge Adamovich, Mindy Levin, "The relationship between control, kinematic, and electormyographic variables in fast single-joint movements in humans", *Experimental Brain Research*, 103, 440-450, 1995.

[13] Flanagan, J.R., and Ostry, D.J., "Trajectories of human multi-joint arm movements: evidence of joint level planning", *Experimental Robotics* (Edited by Hayward, V.), Springer-Verlag, 1990.

[14] Flash, T., "The control of hand equilibrium trajecotires in multi-joint arm movements", *Biological cybernetics*, 57, 257-274, 1987.

[15] Flash, T., I. Gurevich, "Models of mtor adaptation and impedance control in human arm movements", *Self-organization, computational maps, and motor control*, Elsevier, 1997.

[16] Flash, T., Neville Hogan, "The coordination of arm movements: an experimentally confirmed mathematical model", *The Journal of Neuroscience*, 7, 1688-1703, 1985.

[17] Flash, T., F. Mussa-Ivaldi, "Human arm stiffness characteristics during the maintenance of posture", *Experimental Brain Research*, 82, 315-326, 1990.

[18] Georgopoulos, A.P., Kalaska, J.F., and Massey, J.T., "Spatial trajecotries and reaction times of aimed movments: Effects of practice, uncertainty, and change in target location.", *J. Neurophysiol.*, 46,725-743, 1981.

[19] Georgopoulos, A.P., Kalaska, J.F., Caminiti, R., and Masssey, J.T., "On the relations beween the direction of two-dimensional arm movements and cell discharge in primate motor cortex", *The Journal of Neuroscience*, 2, 1527-1537, 1982.

[20] Georgopoulos, A.P., Kettner, R.E., Schwartz, A.B., *The Journal of Neuroscience*, 8, 2913, 1988.

[21] Gandolfo, F., F. A. Mussa-Ivaldi, E. Bizzi, "Motor learning by field approximation", *Proc. Natl. Acad. Sci.*, 93, 199?.

[22] Giszer, Simon, Ferdinand. Mussa-Ivaldi, Emilio Bizzi, "Convergent force fields organized in the frog's spinal cord", *The Journal of Neuroscience*, 13(2), 467-491, 1993.

[23] Hogan, Neville, "Organizing principle for a class of voluntary Movements", *The Journal of Neuroscience*, 11(4), 2745-2754, 1984.

[24] Kalaska, J.F., Cohen, D.A.D., Hyde, M.L., Prud'homme, M. *The Journal of Neuroscience*, 9, 2080, 1989.

[25] Kandel, Eric R., *Princeiples of neural science-third edition*, Appleton & Lange, 1991.

[26] Lackner J.R. and Dizio P., "Rapid adaptation to coriolis force Perturbations of Arm Trajectory", *Journal of Neurophysiology*, 72, 299-312, 1994.

[27] Latash, Mark L., "Reconstruction of equilibrium trajectories and joint stiffness patterns during single-joint voluntary movements under different instructions", *Biological Cybernetics*, 71, 441-450, 1994.

[28] Latash, Mark L., "Control of fast elbow movement: a study of electromyographic patterns during movements against unexpectedly decreased inertial load", *Experimental Brain Research*, 98, 145-152,1994.

[29] Levin, Mindy , A. Feldman, T. Milner, Y. Lamarre, "Reciprocal and coactivation commands for fast wrist movements", *Experimental Brain Research*, 66, 247-256, 1987.

[30] Matthews, Gary G., *Cellular physiology of nerve and muscle-second edition*, Blackwell Scienctific Publications, 1991.

[31] McIntyre, J., F. A. Mussa-Ivaldi, E. Bizzi, "The control of stable postures in the multijoint arm", *Experimental Brain Research*, 110, 248-264, 1996.

[32] Morasso, P., "Spatial control of arm movements", *Experimental Brain Research*, 42, 223-227, 1981.

[33] Mussa-Ivaldi, Ferdinando A., "From basis functions to basis fields: vector field approximation from sparse data", *Biological Cybernetics*, 67, 479-489, 1992.

[34] Mussa-Ivaldi, Ferdinando A., and Giszter, Simon F., "Vector field approximation: a computational paradigm for motor control and leraning", *Biological Cybernetics*, 67, 491-500, 1992.

[35] Mussa-Ivaldi, Ferdinando A., Hogan, N., Bizzi, E., "Neuroal, mechanical and geometric factors subserving arm posture in humans", *The Journal of Neuroscience*, 5, 2732-2743, 1985.

[36] Mussa-Ivaldi, Ferdinando A., Simon F. Giszter, Emilio Bizzi, "Linear combinations of primitives in vertebrate motor control", *Proc. Natl. Acad. Sci.*, 91, 7534-7538, 1994.

[37] Mustard, B.E.. R. G. Lee, "Relationship between EMG patterns and kinematic properties for flexion movements at the human wrist", *Experimental Brain Research*, 66, 247-256, 1987.

[38] Nicogossian, Arnauld E., "Microgravity simulations and analogues", *Space physiology and medicine-third edition*, Lea & Fabiger, 1993.

[39] Paines, Justin David Billot, *Oprimization of manual control dynamics for space telemanipulation: impedence control of a force reflecting hand controller*, MS thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, 1987.

[40] Portree, David S. F., and Treviño, Robert C., *Walking to olympus: an EVA chronology*, NASA History Office, 1997, Monographs in Aerospace History Series #7.

[41] Sanner, Robert M., Kosha, Makiko, "Preliminary evaluation of a mathematical model of adaptive motor control in humans", *American Controls Conference*, 1995.

[42] Sanner, Robert M., Slotine, J-J., "Stable adaptive control of robot manipulators using "neural" networks", *Neural Comput*, 7, 753-788, 1995.

[43] Sanner, Robert M., Slotine, J-J., "Gaussian networkds for direct adaptive control.", *IEEE Trans. Neural Networks*, 3, 837-863, 1997.

[44] Schumaker, L. L. *Spline Functions Basic Theory*, John Wiley & Sons, 1981.

[45] Schwartz, Eric L., ed., *Computational Neuroscience*, MIT Press, 1990.

[46] Shadmehr, Reza, and Mussa-Ivaldi, Ferdinando A., "Adaptive representation of dynamics during learning of a motor task", *The Journal of Neuroscience*,14(5), 3208-3224, 1994.

[47] Shadmehr, Reza, Ferdinando A. Mussa-Ivaldi, Emilio Bizzi, "Postural force fields of the human arm and their role in generating multijoint movements", *The Journal of Neuroscience*,13(1), 45-62, 1993.

[48] Shepherd, Gordon M., "The significance of real neuron architectures for neural network simulations", *Computational neuroscience*, MIT Press, 1990.

[49] Slotine, Jean-.Jacques E., and LI, Weiping, *Applied Nonlinear Control*, Prentice Hall, New Jersey, 1991.

[50] Soechting, J.F., and Lacquaniti, F., "Invariant characteristics of a pointing movement in man., *The Journal of Neuroscience*, 1, 710-720, 1981.

[51] Winter, David A., *Biomechanics and motor control of human movement*, Wiley-Interscience Publication, 1990.

[52] Wolpoert, Daniel M., Zoubin Ghahramani, Micael I. Jordan, "Perceptual distortion contributes to the curviture of human reaching movements", *Experimental Brain Research*, 98, 153-156, 1994.

[53] Zimmerman, Julianne, "Human postural variations between simulation and on-orbit environments", M.S. Thesis, Department of Aerospace Engineering, University of Maryland, SSL96-009, 1996.