Development and Application of a 2-DOF Impedance Controlled Manipulandum for Human Motor Control Studies

by

Chad MacDonald

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

of

Master of Science

ABSTRACT

This thesis discusses the development of a two degree-of-freedom impedance controlled manipulandum to study human motor learning. In particular, the focus was to develop a low-cost alternative to the traditionally expensive manipulandum systems. The design and verification of the mechanical and software components of the manipulator were completed with the aim of applying the system to several specific experimental situations.

The experimental work consisted of studying motor adaptation during reaching movements in the presence of viscous force fields. The primary difference between this work and previous experiments is that the focus was on reaching movements in a more natural arm orientation, rather than in the traditional 90° abduction setup. Specifically, the presence of an internal model for adaptation is verified by studying the after-effects of adaptation. Learning generalization based on reaching direction was analyzed to verify the large receptive field of the basis functions of this internal model. Finally, a study of adaptation to a random force field was performed to determine how recent motor memories contribute to the performance on a specific movement trial.

In addition to this experimental work, a pilot study was performed to examine the effect of Parkinson disease on motor learning. The initial results suggested that Parkinson disease does not significantly impair the patient's ability to construct an internal model of their predicted arm dynamics.

ACKNOWLEDGEMENTS

First and foremost I would like Dr. Zahra Moussavi for her support and guidance throughout my Masters work. She was the first to introduce me to the field of Biomedical Engineering and she provided the inspiration for this project. I would also like to acknowledge the support and instruction provided by Dr. S. Balakrishnan in the development of the robotic aspects of this thesis.

I especially wish to acknowledge Al Symmons and Cory Smit in the Electrical and Computer Engineering machine shop for their assistance in the design and construction of the mechanical components of the manipulandum. As well, I would like to acknowledge the assistance of Al McKay for his help in acquiring the components for construction and also Ken Tarte for his support in the Mechanical Engineering Robotic Automation lab.

I also wish to thank the many people who volunteered to be test subjects during the experimental periods of this work.

Finally, I would like to thank Tawny Wong, for providing a non-technical review of this manuscript, and Daniel Lockery for his keen editing eye and for being my technical sounding-board for the last several years.

For
My Parents

TABLE OF CONTENTS

APPENDICES

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 Introduction

In recent years, the field of human motor control research has involved the study of how people learn and adapt to novel environments. In particular, there has been much focus on the performance of voluntary upper limb reaching movements in the presence of viscous force fields. Such motor learning experiments have been used to study adaptation to different types of environments, as well as the generalization of such adaptation to other situations.

1.2 Motivation

Motor learning research has several motivating factors, including the desire to further understand the neural mechanisms involved in human motor control. Also, with such an understanding, the development of tools and experiments for the early detection of motor impairments, or for the rehabilitation of such impairments is a major motivation in this field. Specific to the work in this thesis, the motivation was to develop a capable low cost robotic manipulandum and to use this system to study motor adaptation in a more natural arm configuration than previous studies. Also, the focus of this work will eventually lead to the study of the motor learning aspects of Parkinson disease, with the aim of possibly developing diagnostic or rehabilitative tools.

1.3 Goals and Objectives

The objective of this thesis is to design and implement an impedance controlled robotic manipulator that can be used to perform human motor control studies. In particular, the designed system is intended to be a lower-cost system compared to those currently in use elsewhere [Faye86]. The experiments performed are primarily intended to demonstrate that this less expensive manipulandum system can obtain comparable results to past experiments. With these experiments complete, this thesis has the goal of introducing a pilot study on the motor adaptation performance of a patient with Parkinson disease.

1.4 Scope

The scope of this thesis encompasses the design and implementation of the robotic manipulandum, with an additional focus on the experimental application of this system to study human motor learning and generalization.

1.5 Organization of Report

This manuscript is divided into nine sections. This first chapter has presented an introduction to the content, goals, motivation and scope of the project. Chapter two presents background material related to human motor control and motor learning necessary for understanding the concepts addressed in the experimental sections. The third chapter explains the background material on robotics required for the design aspects of the project. Chapter four discusses the design of the kinematics and mechanics of the robotic aspects of this work. Chapter five describes the design of the impedance controller software and experimental interfaces. The system verification is discussed in the sixth chapter. Chapter seven presents the design of motor control experiments, while the results of these experiments are discussed in the following eighth chapter. Chapter nine, the final chapter, draws conclusions based on the previous chapters' results and puts forth recommendations for future work.

CHAPTER II

MOTOR LEARNING BACKGROUND

2.1. The Human Motor System

The human motor system is responsible for the control of goal-directed voluntary movements. These movements involve the selection of a task or action to be performed, the planning of forces required to execute this task and finally the coordination of such forces to put this motor plan into action. This section briefly presents the components of the motor system and then discusses different aspects of the motor control process.

2.1.1 Motor System Components

The human motor system is composed of two major components, the peripheral and central motor systems. The peripheral motor system consists of muscles, motor neurons and muscle afferents. The skeletal muscles are the force-generating actuators of the motor control system while the motor neuron is the control interface from the central nervous system to the muscle. Nerve impulses (action potentials) travel from the spinal cord to the muscles through the motor neurons. These electrical signals cause the actin and myosin proteins that make up the fibers of the muscle to slide with respect to each other, resulting in the contraction of the muscle. Since the muscle can only generate force by contraction, it can only generate force in one direction. For this reason muscles work in agonist-antagonist pairs, such as the biceps and triceps of the upper arm. The agonist muscle is the primary mover, while the antagonist muscle provides a stabilizing force. Muscle afferents are neurons that innervate muscles and provide the central nervous system (CNS) with information regarding the length and force of a given muscle [BaDe85].

The central motor system contains elements throughout the CNS including the motor cortex, basal ganglia, brain stem and spinal cord. The role of the CNS is to plan the forces for a desired movement and to determine the required control signals to be sent to the peripheral motor system. For a more complete discussion of the central motor system and its components, the reader is directed to [WiSh01].

2.1.2 Forward and Inverse Problems

In order to perform desired tasks with a multi-segment articulated limb such as the human arm the controller must solve both the forward and inverse problems. The forward problem involves determining the behaviour of an end-effector (ie: the human hand) based on the given joint positions, velocities and the current muscle forces. This amounts to understanding how the system will react to a given set of inputs. The inverse problem involves finding the muscle forces (dynamics) and joint trajectories (kinematics) required to perform a given task. The solutions to the inverse kinematics and inverse dynamics are used by the CNS to choose the appropriate muscle activation patterns to execute the desired movement [WiSh01].

2.1.3 Feedforward and Feedback Control

In order to accurately solve the inverse dynamics problem, the motor system processes information about its environment. This information is used to predict the expected environmental interaction forces and, based on these predictions, adjust the commanded muscle forces accordingly. Ths predictive process is referred to as feedforward control. An example of such a scenario is a person lifting a box. Based on the size, shape and assumed contents, the person estimates how much effort will be required to move the box. This feedforward process plays an important role in Section 2.2 when discussing motor adaptation [ShMu94].

Once a task is being performed different environmental factors can contribute to an error between the desired and actual movement. At this point the motor system uses feedback control to correct for these errors. In the box-lifting example, if the person overestimates the weight of the box, he/she will apply a larger force than is necessary and the box will start moving faster than desired. As a result, the motor system must compare the actual behaviour to the desired behaviour and generate the appropriate muscle forces to provide corrective movements [WiSh01]. Once the desired behaviour is being performed the feedback mechanism also serves to stabilize the limb about the desired trajectory [ShMu94]. This is performed by making small corrective adjustments to the force outputs and by using the agonist-antagonist relationship of the muscles involved. For example, by activating the triceps in addition to the biceps, while flexing the elbow, any external perturbations will have less of an impact on arm trajectory as a result of the additional stiffness provided by the antagonist triceps [WiSh01].

2.2 Motor Adaptation

Having previously discussed the motor control system, this section expands further upon the specific aspect of motor adaptation. As children grow and the length of their bones increases, their arm dynamics change drastically. As a result, the muscle commands required to generate a specific behaviour will vary. From this, it follows that any control strategy employed by the CNS must be robust enough to handle changes in system dynamics. This robustness is achieved through the process of motor adaptation [ShMu94], which essentially updates the internal model to account for the altered limb dynamics.

2.2.1 Internal Models

The dynamics of a task are solved by applying an 'internal model' of the behaviour. This model encompasses the dynamics of the arm itself along with the dynamics of the environment with which it is interacting. Many researchers have examined voluntary reaching movements of the upper limb to find evidence of this internal model [ShMu94] [KaWo98] [BhSh99] [Muss99]. This model refers to the transformations required to solve both the forward and inverse dynamics problems.

As an example of the evidence for the generation of an internal model, the experiments performed in [ShMu94] study voluntary reaching movements with the upper limb. Subjects were instructed to grasp the handle of a robotic manipulandum and perform a series of movements during which the robot would generate a velocity-dependent perturbing force field. During initial application of the force field subjects exhibited a significant error in their movements when compared to their desired trajectory. This trajectory was assumed to be minimum jerk and along a straight line connecting the start and end points of the movement. Over time, the subjects adapted to the field and regained their null field (no forces applied) performance. The authors suggested that if the subjects were forming an internal model, they would be explicitly generating muscle commands to counteract the encountered environmental forces. This was verified by the sudden removal of the force field in 'after-effect' trials where subjects exhibited a significant movement error that directly mirrored their initial error within the force field. This result was in agreement with previous experiments involving motor adaptation [FlGu92]. The end result showed that the CNS uses an internal model of the expected arm/environment dynamics to predict the forces required to perform a given reaching task.

Another important point demonstrated in the work of [ShMu94] was that the internal model is defined in terms of the intrinsic joint reference frames, as opposed to an extrinsic 'world frame.' This result suggests that the motor control hierarchy is divided in such a way that the higher level structures plan movements in an external frame, "move my hand from point A to point B," while the lower level structures solve the dynamics of the problem in terms of joint torques.

2.2.2 Basis Functions

With experimental evidence of the presence of internal models, the next step is to examine how these models are represented in the human brain. Answering this question amounts to determining the basic building blocks of motor tasks and relating these 'motor primitives' to complex reaching movements. Using control strategies that assign actions to every possible state of the system [Ogat01], researchers have demonstrated that complex behaviours can be learned by tuning the parameters of local controllers. These controllers correspond to predetermined 'basis functions' which can be tuned and combined to perform more complex tasks [ScAt98] [Muss97] [ThSh00]. Each local controller operates within a region of possible states, referred to as the function's 'receptive field.' The nature of how adaptation occurs and how complex movements are learned depends on the size of these receptive fields. Smaller receptive fields allow for more precise learning with the tuning parameters of that region having little effect on neighbouring regions. The tuning of larger receptive fields could potentially cause interference with a person's performance in a neighbouring region [ScAt98]. Previous work has shown that when human subjects learn a novel force field in one region, the learning process causes significant after-effects in other regions [ShMu94] [ShMo00]. This result suggests that the basis functions employed by the human motor control system have large receptive fields.

2.2.3 Adaptation with Uncertainty

Up until now, the bulk of the work discussed has employed a deterministic force field with constant viscosity gain factors. Learning amid uncertainty has been the focus of other recent research in the field of motor adaptation [ScDM01] [DoSh02] [DoFS03]. The main idea of this work is to determine what features affect adaptation. As in previous experiments, all of these researchers had subjects perform reaching tasks within viscous force fields. For example, in [ScDM01] the subjects performed reaching movements in a single direction while experiencing a perpendicular force field with a random magnitude. Subjects appeared to adapt to the approximate mean of the force field, and that movement error (maximum perpendicular displacement) on a given trial depended on the error and gain experienced during the previous trial only. The result was found using a correlation analysis as depicted in Figure 2.1. Figure 2.1A shows that there is significant correlation between movement error and the gain experienced on the previous trial. The dashed lines refer to the 95% confidence interval ($2\sigma/\sqrt{N}$). Figure 2.1-B shows that there is significant correlation between the current movement error and that experienced on the previous trial.

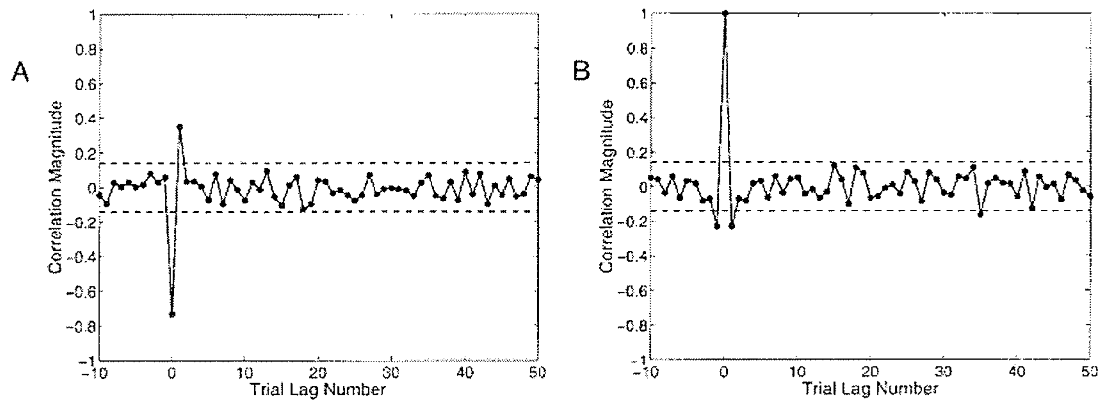


Figure 2.1: Movement error/gain correlations in random field.
A: Error/gain cross-correlation, B: Error autocorrelation.
(Adapted from [ScDM01])

2.3 Learning Generalization

One aspect of motor adaptation that is of particular interest is learning generalization. The goal is to discover the circumstances under which learning in one situation affects motor performance in another. Such generalization and abstraction can be seen in many day-to-day human activities. For example, when someone learns to write on a piece of paper on a desk, they are able to then write on a vertical blackboard without having to relearn any writing tasks.

By studying how motor learning generalizes one can gain insight into how the internal model is represented in the CNS. Several researchers have studied this idea in a variety of scenarios. In [ShMo00], generalization was studied for different arm configurations. Subjects experienced joint-viscous force fields with their arm in different orientations with respect to their body. By studying the after-effects of learning it was found that subjects generalized their motor adaptation from one workspace to another. In [CDGS02] it was found that subjects could generalize their motor performance from their dominant arm to their non-dominant arm, while generalization the other way did not occur.

Much work has been done studying the generalization of motor learning between different reaching directions [DoFS03] [DoSh02] [ThSh00]. This line of research was aimed at quantifying generalization in order to derive a 'generalization function' that could be used to find a possible description of the basis function receptive fields. It was found that the receptive fields appeared 'gaussian-like' and were large enough to permit generalization with a change in direction of 45° from the learned direction.

CHAPTER III

ROBOTICS BACKGROUND

The following chapter is intended to familiarize the reader with the fundamentals of robotic manipulators required for this work. Tools are developed for the position and velocity analysis of manipulators along with the framework of an impedance control strategy to be employed in this work.

3.1 Homogeneous Transformations

In anticipation of the need for transformation matrices in the kinematic analysis of robots, this section discusses the required rotation and translation matrices [Bala03], [Tsai99]. Rotation about the *X*, *Y* and *Z* Cartesian axes can be described in matrix form, as shown in (3.1-3) respectively. If a point in space is defined as a column vector, $P = [P_X \quad P_Y \quad P_Z]^T$, a rotation of *P* can be performed by pre-multiplying by one of the following rotation matrices, where *X*, *Y* and Z are the axes of rotation and ψ , ϕ and ψ are the respective rotation angles.

$$R(X,\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix} \tag{3.1}$$

$$R(Y,\phi) = \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix} \tag{3.2}$$

$$R(Z,\theta)=\begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix} \tag{3.3}$$

The translation of a point in space is performed by adding the column vector, $Q=[Q_X \quad Q_Y \quad Q_Z]^T$ to the point P.

The position of a point expressed in terms of a fixed frame A, ${}^{A}P$, can be described with respect to a moving frame B by ${}^{B}P$. The transformation from ${}^{B}P$ to ${}^{A}P$ can be expressed as a combination of rotations and a translation, as in (3.4).

$${}^{A}P={}^{A}R_{B}\,{}^{B}P+{}^{A}Q \tag{3.4}$$

The rotation matrix, ${}^{A}R_{B}$, describes the orientation of frame B in terms of rotations about frame A using (3.1-3). The translation vector ${}^{A}Q$ denotes the position of the origin of the moving frame in terms of the fixed frame.

The problem with (3.4) is that the transformation is not in a compact form. It is desirable to define a transformation matrix that incorporates both rotation and translation. To this end, homogeneous coordinates and transformations are now introduced. If, as before, the position of a point in three-dimensional (3D) space is $P=[P_X \quad P_Y \quad P_Z]^T$, the homogeneous coordinates of P are:

$$P=[\rho P_X \quad \rho P_Y \quad \rho P_Z \quad \rho]^T \tag{3.5}$$

where ρ is a scaling factor. For simplicity, this factor is chosen to be $\rho=1$. As a result, the homogeneous coordinates of a point in 3D space are given by:

$$P=[P_X \quad P_Y \quad P_Z \quad 1]^T \tag{3.6}$$

With a 4D description of a point in homogeneous coordinates, we now define a 4x4 homogeneous transformation matrix to be used for mapping between coordinate systems. This matrix is composed of four submatrices and is given by:

$$^{A}T_{B} = \begin{bmatrix} ^{A}R_{B}\ (3x3) & ^{A}Q\ (3x1) \\ \gamma\ (1x3) & \rho\ (1x1) \end{bmatrix} \tag{3.7}$$

where $^{A}R_{B}$, ^{A}Q and ρ are as defined previously, and γ is a perspective transformation matrix. For robot kinematic analysis, γ is set to a zero-vector. Using (3.6) and (3.7), the transformation in (3.4) can now be expressed as:

$$^{A}P = {}^{A}T_{B}\,{}^{B}P \tag{3.8}$$

As examples of simple transformations described using (3.7), a rotation about the Z axis is shown in (3.9) and a translation along all three axes is shown in (3.10).

$$R(Z,\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

$$Q = \begin{bmatrix} 1 & 0 & 0 & Q_X \\ 0 & 1 & 0 & Q_Y \\ 0 & 0 & 1 & Q_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

A transformation from one reference frame to another using a series of rotations and translations can be obtained by the multiplication of successive homogeneous transformations. Care must be taken with the order of these multiplications since finite rotations are not commutative.

3.2 Serial and Parallel Manipulators

In general, robotic manipulators can be classified in terms of their kinematic structure as either *serial* or *parallel.* There is also the possibility of having hybrid systems that are a combination of these two types. Parallel manipulators are those that are made up of a closed-loop chain of links, while a serial manipulator contains no closed loops. Examples of serial and parallel robots are shown in Figures 3.1 and 3.2 respectively.

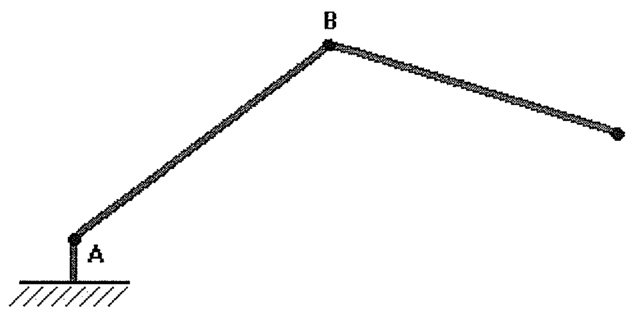


Figure 3.1: Serial manipulator.

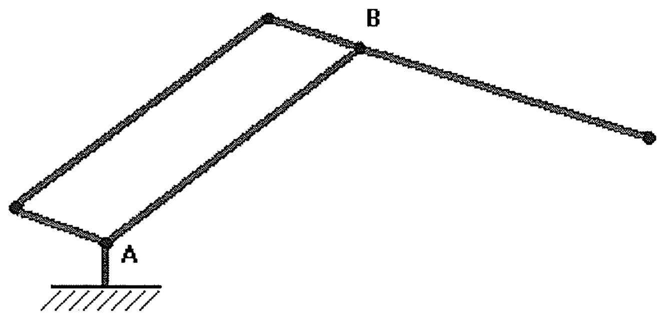


Figure 3.2: Parallel manipulator.

For the serial manipulator in Figure 3.1, the actuators would be placed directly at the joints (points A and B). One major disadvantage of this configuration is that, if the motors used for actuation are relatively large, placing an actuator at B would mean that the motor at A would have to expend power moving the additional load at B. The advantage of such a configuration is that it has a relatively large workspace and the mechanism is much simpler than that of a parallel manipulator [Tsai99].

In the case of the parallel manipulator in Figure 3.2, both joint actuators are placed at the base of the robot (point A). The torque from the motor controlling joint B is transferred through the parallel structure. In general, parallel manipulators have lower inertia (due to actuator placement) and higher stiffness than serial manipulators. However, these come at the expense of a smaller workspace due to the limiting effect of the closed-loop's constraints [Tsai99].

3.3 Robot Kinematics

A kinematic description of a robotic manipulator provides a relationship between the Cartesian position of the end-effector and the joint positions. A kinematic analysis of a specific structure is divided into two problems: Forward Kinematics and Inverse Kinematics. The forward problem involves determining the end-effector location and orientation based on the current joint positions. The inverse problem requires solving for joint positions that will place the robot's end-effector into a desired location and orientation [Bala03] [Bala04]. To allow for simplified analysis several standard methods have been developed to describe robotic linkages [KhDo02]. The differences between them are in how parameters and coordinate systems are assigned to the manipulator linkages. In this work the Denavit-Hartenberg (DH) convention is employed [Tsai99]. The following briefly discusses this convention and shows how it can be used to solve the forward and inverse kinematic problems.

3.3.1 Denavit-Hartenberg Convention

For a general *n*-DOF serial manipulator, a description of the robot's geometry first requires the number of the links and joints of the system. The links are numbered sequentially from 0 to *n* starting from the base, while the joints are numbered from 1 to *n*. The following figure shows a serial 3-DOF planar manipulator with its component links and joints numbered as described above.

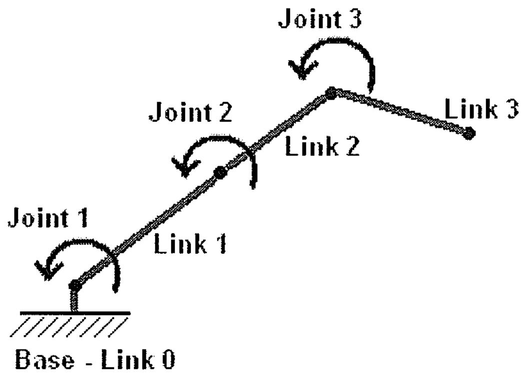


Figure 3.3: Link and joint numbering scheme.

With the joints and links numbered, the next step is to assign a Cartesian coordinate system to each link. The DH-convention attaches coordinate frame *i* to link *i* using the following rules:

- The *Z*-axis of frame *i* is aligned with the axis of the (*i*+1)th joint. The positive *Z* direction can be selected arbitrarily.

- The X-axis of frame i is aligned with the common normal between the ith and (i+1)th joint axes. If the joint axes are parallel, the X-axis is defined to be perpendicular to the joint axes. However, if successive joint axes intersect, the X-axis is defined along the cross product of the two joint axes (or in the opposite direction).
- The Y-axis of frame i is selected according to the right-hand rule.

The base coordinate frame can be defined in any convenient place, as long as the Z-axis is aligned with the axis of joint 1. Finally, a coordinate frame is attached to the end-effector, such that the X-axis is normal to the last joint's axis.

Associated with this frame assignment, the DH convention defines four parameters, which describe the position and orientation of link $i+1$ with respect to link i. If we define H_{i-1} as the intersection of X_i and Z_{i-1} and O_i as the origin of frame i, then the DH parameters can be defined as shown in Figure 3.4. The variable parameters for revolute and prismatic joints are θ_i and d_i respectively. These are referred to as *joint variables*, while the constant terms are called *link parameters*.

Once the link coordinate frames are assigned and the joint variables and link parameters are determined for each joint, the homogeneous transformation matrix from the (i-1)th frame to the ith frame can be defined using the basic transformations from Section 3.1. Frame i-1 is translated along Z_i by a distance of d_i, (3.11), and then the translated frame is rotated about Z_{i-1} by the angle θ_i, (3.12). This is followed by a translation along X_i by a_i, (3.13), and then a rotation about X_i by α_i, (3.14).

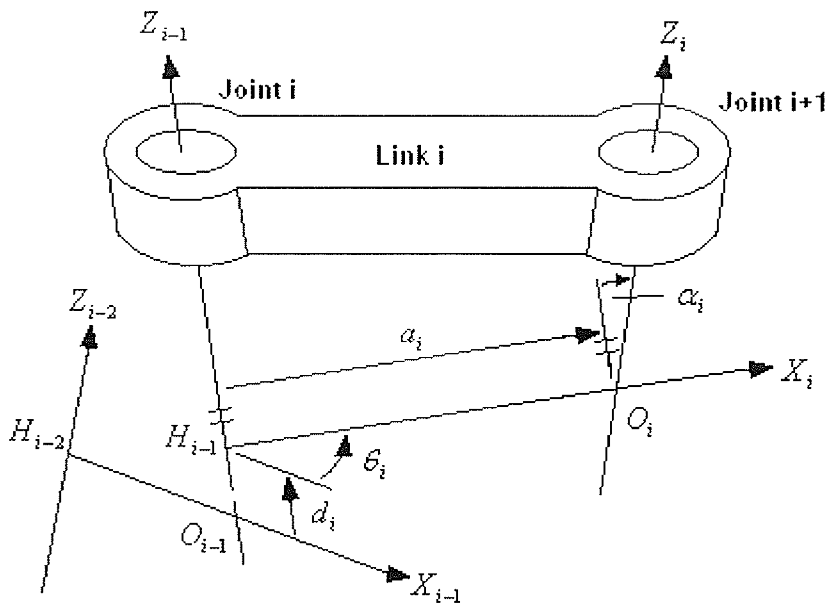


Figure 3.4: DH link parameters.

a_i: Link length, the distance from H_{i-1} to O_i along X_i

d_i: Link offset, the distance from O_{i-1} to H_{i-1} along Z_{i-1}.

α_i: Link twist, the angle required to rotate Z_{i-1} to align with Z_i about X_i, according to the right-hand rule.

θ_i: Joint angle, the angle required to rotate X_{i-1} to align with X_i about Z_{i-1}, according to the right-hand rule.

$$T(Z,d)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & d_i\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{3.11}$$

$$T(Z,\theta)=\begin{bmatrix}\cos\theta_i & -\sin\theta_i & 0 & 0\\ \sin\theta_i & \cos\theta_i & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{3.12}$$

$$T(X,a)=\begin{bmatrix}1 & 0 & 0 & a_i\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{3.13}$$

$$T(X,d)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\alpha_i & -\sin\alpha_i & 0\\ 0 & \sin\alpha_i & \cos\alpha_i & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{3.14}$$

These can be seen as transformations about the moving coordinate axes, and they can be multiplied to obtain the DH transformation matrix ${}^{i\text{-}1}A_i$, given by:

$${}^{i-1}A_i = T(Z,d)T(z,\theta)T(X,a)T(X,\alpha) \tag{3.15}$$

Multiplying (3.15) through, the expanded matrix becomes:

$${}^{i-1}A_i = \begin{bmatrix}\cos\theta_i & -\cos\alpha_i\sin\theta_i & \sin\alpha_i\sin\theta_i & a_i\cos\theta_i\\ \sin\theta_i & \cos\alpha_i\cos\theta_i & -\sin\alpha_i\cos\theta_i & a_i\sin\theta_i\\ 0 & \sin\alpha_i & \cos\alpha_i & d_i\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{3.16}$$

3.3.2 Forward Kinematic Solution

Given joint variables and link parameters for a robotic manipulator, the forward kinematic solution is used to determine position and orientation of the robot end-effector in terms of the base reference frame. The forward kinematics solution for a serial manipulator requires determining the transformation matrix from the base frame to the end-effector frame, ${}^{0}A_n$. This is found by successively multiplying matrix (3.16) of the joints in the manipulator, resulting in:

$$ {}^{0}A_{n} = {}^{0}A_{1}\,{}^{1}A_{2}\cdots{}^{n-2}A_{n-1}\,{}^{n-1}A_{n} = \begin{bmatrix} & {}^{0}R_{n} & & {}^{0}Q_{n} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.17} $$

The 3x3 matrix ${}^{0}R_{n}$ describes the orientation of the robot end-effector, while ${}^{0}Q_{n}$ describes the position of O_n. Both ${}^{0}R_{n}$ and ${}^{0}Q_{n}$ are functions of the joint variables, and can be solved for directly.

3.3.3 Inverse Kinematic Solution

The inverse kinematic solution for a serial manipulator is generally more difficult than the forward solution. Given the desired location and orientation of the end-effector in terms of the base coordinate system, the required joint variables are found. For the inverse case, the elements of the matrix ${}^{0}A_{n}$ in (3.17) are specified and the joint variables on the right hand side of this equation must be solved for. Equation (3.17) provides 12 nontrivial equations. Given a simple enough robot configuration they can be solved directly to find the joint variables. However, this is often not the case and different techniques must be applied to simplify the solution. For example, if there are three intersecting joint axes, one can work about the point of intersection, thus avoiding the variables associated with these axes. Also, if there are parallel joint axes (as with the planar robot of Figure 3.1), their associated joint variables may be combined to simplify the solution process. Finally, one can pre- or post-multiply the left-hand side of (3.17) by the inverse DH transformation to redistribute the unknown variables evenly between both sides of the equation. The inverse DH transformation is formed in a similar manner to its forward counterpart as shown in (3.18).

$$ {}^{i}A_{i-1} = \begin{bmatrix} \cos\theta_i & \sin\theta_i & 0 & -a_i \\ -\cos\alpha_i \sin\theta_i & \cos\alpha_i \cos\theta_i & \sin\alpha_i & -d_i \sin\alpha_i \\ \sin\alpha_i \sin\theta_i & -\sin\alpha_i \cos\theta_i & \cos\alpha_i & -d_i \cos\alpha_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.18} $$

3.4 Jacobian Analysis

Along with position analysis of a robotic manipulator, one must be able to determine the velocity of the end-effector given the velocities of the joints. Let *x* be the vector of *m* position and orientation variables of the robot end-effector and let *q* be the vector of *n* joint positions. From the kinematic analysis, it is seen that each element of *x* is a function of the elements of *q*, as shown in (3.19).

$$ x_i = f_i(q_1, q_2, \ldots, q_n), \quad i = 1,2,\ldots,m \tag{3.19} $$

The time derivative of (3.19) can then be expressed as:

$$ \dot{x}_i = \frac{\partial f_i}{\partial q_1}\dot{q}_1 + \frac{\partial f_i}{\partial q_2}\dot{q}_2 + \cdots + \frac{\partial f_i}{\partial q_n}\dot{q}_n, \quad i = 1,2,\ldots,m \tag{3.20} $$

The matrix form of (3.20) is then:

$$ \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_m \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial q_1} & \frac{\partial f_1}{\partial q_2} & \cdots & \frac{\partial f_1}{\partial q_n} \\ \frac{\partial f_2}{\partial q_1} & \frac{\partial f_2}{\partial q_2} & \cdots & \frac{\partial f_2}{\partial q_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial f_m}{\partial q_1} & \frac{\partial f_m}{\partial q_2} & \cdots & \frac{\partial f_m}{\partial q_n} \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_n \end{bmatrix} \tag{3.21} $$

where the 4x4 matrix is called the Jacobian of the manipulator and is denoted by *J*. Determining the end-effector velocities simply becomes a matter of multiplying the vector of joint positions by the Jacobian.

If the number of position values and joint variables are the same, which is the case for non-redundant manipulators, the Jacobian is a square matrix with $m = n$. In this case, the inverse Jacobian may be used to solve for joint velocities given the end-effector Cartesian and angular velocities as shown in (3.22). Note that if the inverse Jacobian is used, care must be taken to avoid situations where the Jacobian matrix is singular. In these cases its determinant is zero and the inverse Jacobian does not exist.

$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_n \end{bmatrix} = J^{-1} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_m \end{bmatrix} \tag{3.22}$$

3.5 Impedance Control

Since the goal of this work was to develop a robotic manipulator to interact with a nonlinear dynamic environment (human arm) the selection of an appropriate control strategy was required. Similar systems developed in the past [Faye86] have used the concept of impedance control [Hoga85a-c] to produce the desired end-effector interaction forces. The impedance control approach to manipulation was created specifically to handle the interaction between a controlled system (the manipulator) and a dynamic environment (the test subject).

While most control strategies are used to control specific system variables, such as Cartesian position/velocity or joint position/velocity ([KhDo02], [Tsai99], [Bala03-04], [Ogat01]), the aim of impedance control is to manage the relationship between interaction force and trajectory of an interaction port. For the experiments reported within this work, the interaction point is the end-effector grasped by a human subject.

Generally, the relationship between the interaction force and the trajectory of a robot end-effector is described by a set of nonlinear equations that vary with time. The goal is to define a mapping from one vector of variables (trajectory) to another (force).

There are two primary forms of impedance controllers presented in the literature [Hoga85b]. In position-based impedance control, the actuated joint variable is position. In this system, the current end-effector interaction force is measured and the trajectory required to produce the desired impedance relationship is determined. Subsequently, the corresponding commands sent to the joint actuators. In torque-based impedance control, the current position and velocity of the end-effector are determined from the joint trajectories. Then the interaction force required for the desired impedance relationship is found. From this interaction force, the required joint torques and/or forces are calculated and commanded to the joint actuators. Since the goal of this work is to control the apparent stiffness and viscosity experienced by the test subject, the torque-based approach is used in this work. This is also a favourable approach since the determination of joint velocities is generally easier and more feasible than measuring interaction forces at the end-effector.

3.5.1 Impedance Controller

A torque-based impedance controller works by comparing the current end-effector trajectory with a *virtual trajectory* and generating interaction forces that are functions of the difference. As a result, the force generated by the robot goes to zero only when the current trajectory matches the virtual (desired) trajectory. The impedance controller is designed such that it generates forces that attempt to send the end-effector to the virtual position.

The specific relationship implemented by the impedance controller is shown in (3.23). The differences between the current position and velocity (*X* and *V* respectively) and the virtual position and velocity (X_0 and V_0) are determined and the desired stiffness and viscosity values (*K* and *B*) are used to determine the required interaction force.

$$F_{\text{int}} = K(X_0 - X) + B(V_0 - V) \tag{3.23}$$

The stiffness and viscosity parameters are square matrices (assuming the dimensionality of F_{int} and *X* are equal) that define the characteristics of the impedance-controlled system.

3.5.2 Virtual Work

Once the desired interaction force has been determined it is necessary to transform this into required joint torques/forces. This is accomplished through the principle of *virtual work* ([Hoga85b], [Tsai99]), which states that the work (*W*) done for an infinitesimal change in coordinates is equal to zero. This work is the sum of the work performed by the torques at the joints and the output forces at the end-effector, as seen in (3.24).

$$\partial W = \tau^T \partial\theta - F^T \partial X = 0 \tag{3.24}$$

In (3.24), *F* is the force vector at the end-effector, *X* is the end-effector position vector, τ is the vector of joint torques/forces and θ is the joint position vector. Substituting the velocity relationship from (3.21) into (3.24) and rearranging, we get:

$$\tau^T \partial\theta = F^T J \partial\theta \tag{3.25}$$

$$\tau^T = F^T J \tag{3.26}$$

$$\tau = J^T F \tag{3.27}$$

As shown in (3.27), the joint torques/forces required to generate a desired end-effector force vector can be calculated using the Jacobian of the robotic system.

3.5.3 Other Considerations

The impedance controller defined by (3.23) and (3.27) does not include other environmental factors such as gravity and friction. If compensation for joint torques created by gravity or friction is required, the necessary adjustments may be made to the torque values calculated in (3.27). Compensation for gravity can be performed by applying counter-torques to compensate for any acceleration due to gravity experienced by the manipulator linkages.

Compensation for friction effects from the actuators (motor friction, gearheads, etc.) can be performed by applying an appropriate friction model and generating required counter-torques. Some common velocity-based friction models are demonstrated in Figure 3.5 [KhDo02].

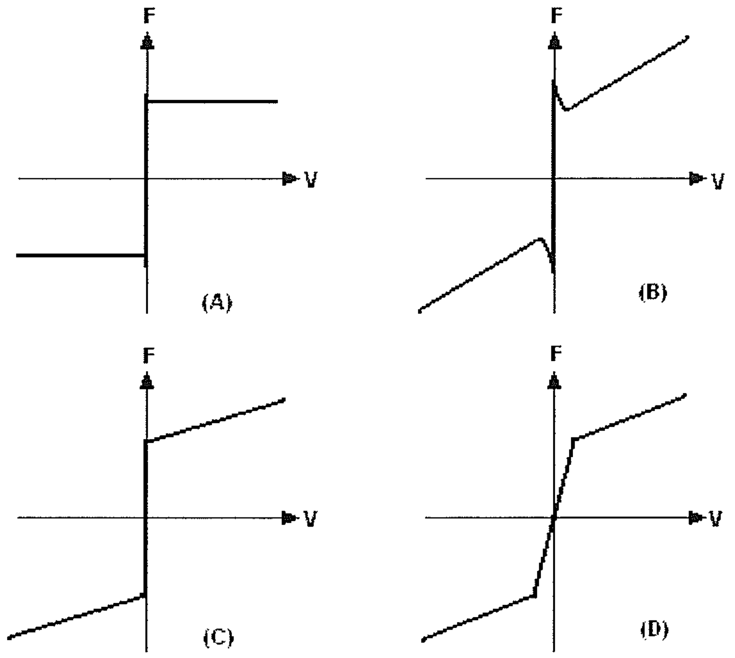


Figure 3.5: Friction compensation torque models (V – velocity, F – friction compecsation torque, A: Coulomb friction (simplest), B: Coulomb+viscous+Stribeck effect, C: Coulomb+viscous friction (most common), D: Piecewise linear approximation.

CHAPTER IV

ROBOTIC DESIGN AND IMPLEMENTATION

This chapter presents the design and implementation details of the robot design implemented in this work. After reviewing the basic design requirements, the linkage design is discussed, during which the kinematic and Jacobian analysis for the chosen structure are presented. After the higher-level structural design is discussed, the practical mechanical design is presented. This covers the physical structure of the linkage as well as the actuator drive system. Finally, the chapter finishes with a discussion of the safety features designed into the system.

4.1 System Overview and Requirements

4.1.1 System Overview

This section gives a brief overview of the entire system being designed. This is intended to give the reader a perspective regarding the location and purpose of each component discussed in this chapter. The overall system diagram for the impedance controlled manipulator is shown below in Figure 4.1. The actual physical manipulator is composed of the Robot Linkage, the support base, joint actuators (motors) and the servo amplifiers. These components, along with their associated kinematics and dynamics are discussed in this chapter. The design and implementation of the control software, and any other design issues on the PC side of the system are presented in Chapter 5.

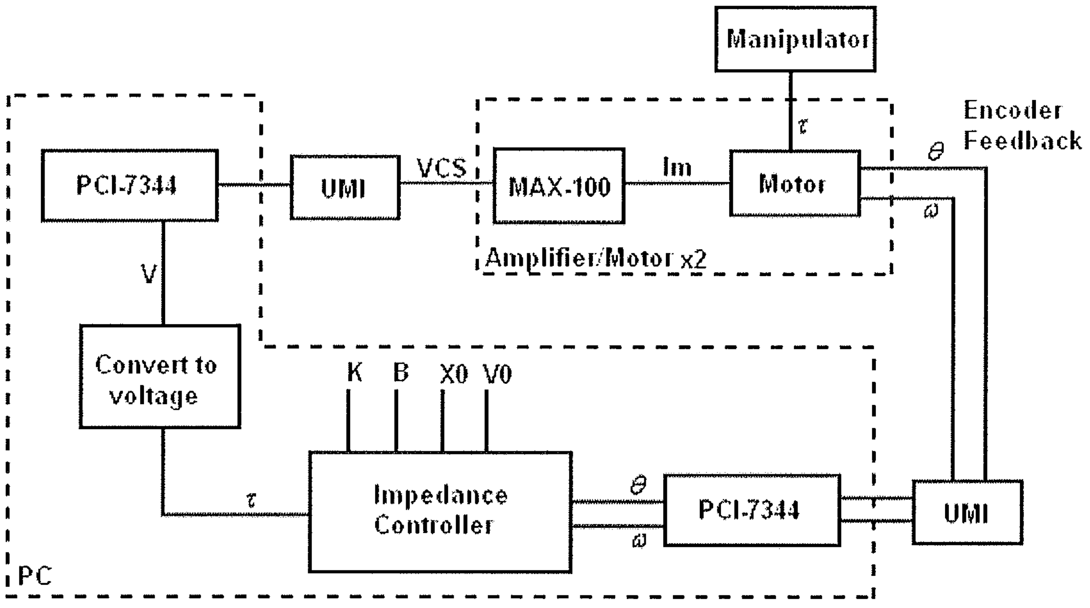


Figure 4.1: Overall system block diagram.

4.1.2 Requirements

Before proceeding to the discussion of the design and implementation of the robotic system, this section discusses some of the requirements that need to be addressed by any chosen design. The first requirement is the size of the workspace needed. In order to perform experiments like those discussed in Chapter II [ShMu94] [MPCS03] [ScDM01] [ShMo00], a square planar (horizontal) workspace with a side length of at least 20 cm is required. With respect to the required force production, with peak velocities in the 0.75-1.5 m/s, [ScDM01], and viscosity matrices with components in the 10-15 N-sec/m range, [ShMu94], the maximum force output should be around 20N in any given direction. Also, the robot's inertia and friction should be low [Faye86] in order to allow for more accurate force production. Finally, in order to facilitate a wider variety of possible experiments, a mechanism for adjusting the height of the robot off of the floor is required.

4.2 Linkage Design

4.2.1 Robot Structure

Due to the inertial requirements of this system, a parallel structure similar to that in Figure 3.2 was selected. This allowed the placement of the elbow actuator at the base of the robot. This planar linkage, Figure 4.2, was oriented so that it operates in the horizontal plane.

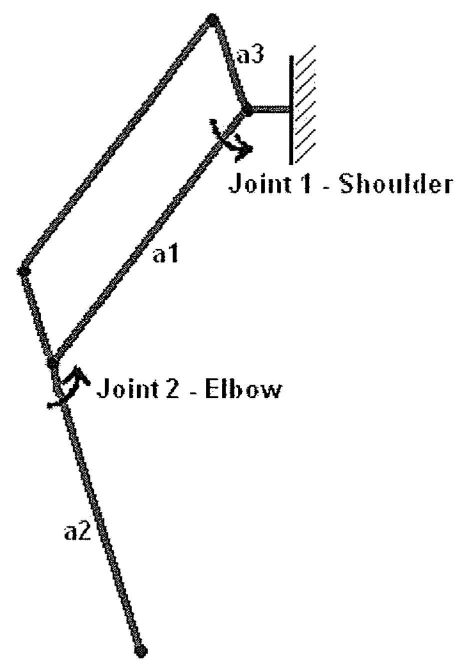


Figure 4.2: Basic parallel linkage structure.

By selecting the link lengths shown in (4.1), the workspace in Figure 4.3 was made possible. If the centre of the square operating region is selected to be at (x,y) coordinates of (40,0) cm, the potential operating region is a square with a side length of around 25 cm.

$$a1 = 35cm,\ a2 = 25cm,\ a3 = 10cm \tag{4.1}$$

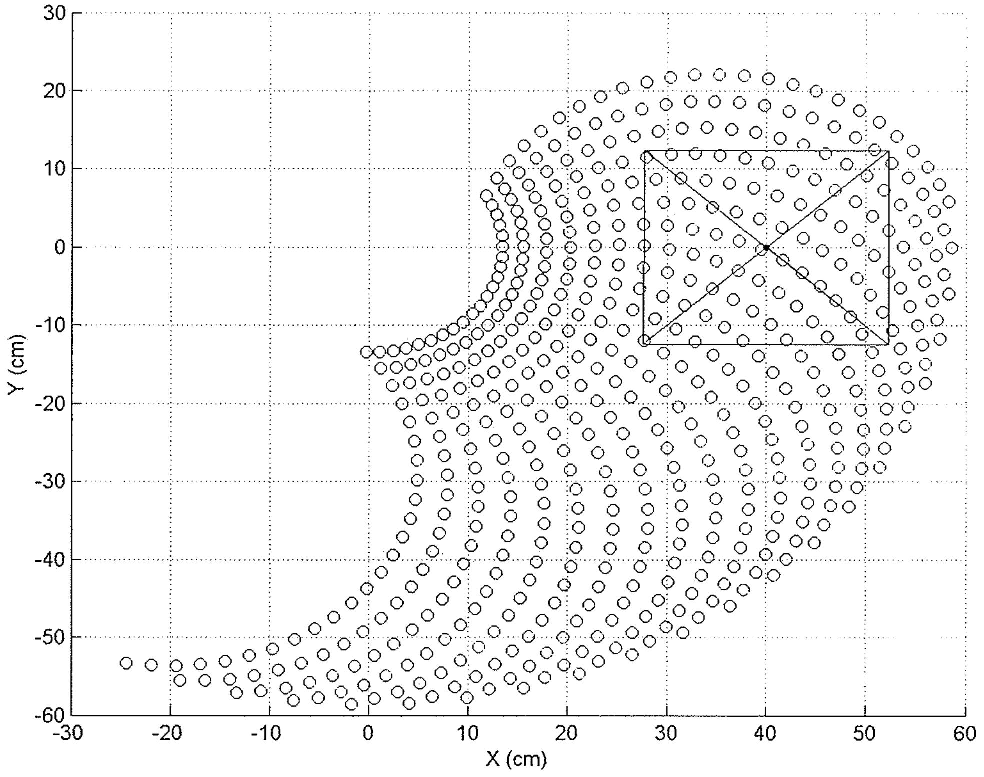


Figure 4.3: Manipulator workspace. Square operating region marked.

4.2.2 Forward Kinematics

The first stage of the forward kinematic solution is to assign coordinate frames to the linkage in Figure 4.2 using the DH convention outlined in Section 3.3.1. Doing so results in the following frame assignment:

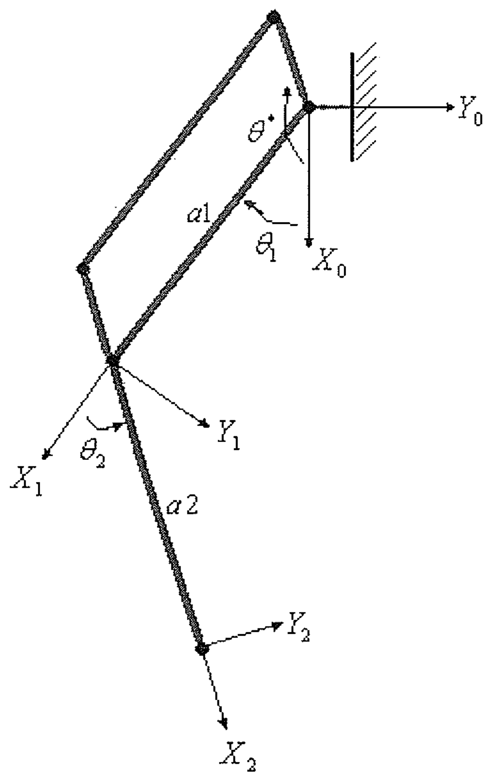


Figure 4.4: Manipulator DH frame assignment.

The parallel loop in this manipulator simply serves to transfer torque from the base to the elbow, and does not actually affect the kinematic analysis of the manipulator. As a result, the forward kinematics can be solved in the same way as they would be for a 2-DOF serial manipulator. The DH parameters for this linkage are presented in Table 4.1:

Table 4.1: DH parameters

Joint	α_i	a_i	d_i	θ_i
1	0	a_1	0	θ_1
2	0	a_2	0	θ_2

Substituting these values into (3.16) gives the following transformation matrices:

$$
{}^{0}A_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & a_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & 0 & a_1\sin\theta_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}
$$

$$
{}^{1}A_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & a_2\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & a_2\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}
$$

Multiplying these according to (3.17) gives a forward DH transform of:

$$
{}^{0}A_2 = \begin{bmatrix} \cos\theta_{12} & -\sin\theta_{12} & 0 & a_1\cos\theta_1 + a_2\cos\theta_{12} \\ \sin\theta_{12} & \cos\theta_{12} & 0 & a_1\sin\theta_1 + a_2\sin\theta_{12} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4}
$$

Using (4.4), one can solve for the location of the end-effector, *Q* given the current joint angles.

$$
\begin{aligned} Q_X &= a_1\cos\theta_1 + a_2\cos\theta_{12} \\ Q_Y &= a_1\sin\theta_1 + a_2\sin\theta_{12} \end{aligned} \tag{4.5}
$$

Since the angle of joint 2 is not measured directly, it must be calculated using the measured values of θ_1 andθ^*:

$$
\theta_2 = \theta^* - \theta_1 - \pi \tag{4.6}
$$

4.2.3 Inverse Kinematics

Given the end-effector location, the inverse kinematics can be completed by solving (4.5) to find θ_1 andθ_2. The first step is to realize that the distance from the base to the end-effector is independent ofθ_1, meaning that θ_1 can be eliminated by summing the squares of Q_X and Q_Y.

$$Q_X^2 + Q_Y^2 = a_1^2 + a_2^2 + 2a_1a_2\cos\theta_2 \tag{4.7}$$

Then (4.7) can be solved to find θ_2.

$$\theta_2 = \cos^{-1}\left(\frac{Q_X^2 + Q_Y^2 - a_1^2 - a_2^2}{2a_1a_2}\right) \tag{4.8}$$

There are two potential solutions to (4.8), however due to the constraints imposed by the parallel linkage; only one solution will be valid. Given θ_2, the solution for θ_1 can be found by expanding the $\cos\theta_{12}$ and $\sin\theta_{12}$ terms in (4.5) and rearranging the result to get:

$$\begin{aligned} Q_X &= (a_1 + a_2\cos\theta_2)\cos\theta_1 - (a_2\sin\theta_2)\sin\theta_1 \\ Q_Y &= (a_2\sin\theta_2)\cos\theta_1 + (a_1 + a_2\cos\theta_2)\sin\theta_1 \end{aligned} \tag{4.9}$$

This can then be solved for $\cos\theta_1$ and $\sin\theta_1$,

$$\begin{aligned} \cos\theta_1 &= \frac{Q_X(a_1 + a_2\cos\theta_2) + Q_Y a_2\sin\theta_2}{a_1^2 + a_2^2 + 2a_1a_2\cos\theta_2} \\ \sin\theta_1 &= \frac{-Q_X a_2\sin\theta_2 + Q_Y(a_1 + a_2\cos\theta_2)}{a_1^2 + a_2^2 + 2a_1a_2\cos\theta_2} \end{aligned} \tag{4.10}$$

Finally, θ_1 can be found from:

$$\theta_1 = A\tan 2(\sin\theta_1, \cos\theta_1) \tag{4.11}$$

4.2.4 Jacobian Analysis

The first stage of the Jacobian analysis is to rewrite (3.20) for the system being analyzed. For the chosen parallel linkage, this gives:

$$\begin{bmatrix} \dot{Q}_X \\ \dot{Q}_Y \end{bmatrix} = \begin{bmatrix} \frac{\partial Q_X}{\partial\theta_1} & \frac{\partial Q_X}{\partial\theta_2} \\ \frac{\partial Q_Y}{\partial\theta_1} & \frac{\partial Q_Y}{\partial\theta_2} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \tag{4.12}$$

The partial derivatives in the Jacobian are found to be:

$$
\begin{aligned}
\frac{\partial Q_X}{\partial \theta_1} &= -(a_1 \sin\theta_1 + a_2 \sin\theta_{12}) \\
\frac{\partial Q_X}{\partial \theta_2} &= -a_2 \sin\theta_{12} \\
\frac{\partial Q_Y}{\partial \theta_1} &= (a_1 \cos\theta_1 + a_2 \cos\theta_{12}) \\
\frac{\partial Q_Y}{\partial \theta_2} &= a_2 \cos\theta_{12}
\end{aligned}
\tag{4.13}
$$

Substituting (4.12) into (4.13), the final Jacobian equation becomes:

$$
\begin{bmatrix} \dot{Q}_X \\ \dot{Q}_Y \end{bmatrix} = \begin{bmatrix} -(a_1 \sin\theta_1 + a_2 \sin\theta_{12}) & -a_2 \sin\theta_{12} \\ (a_1 \cos\theta_1 + a_2 \cos\theta_{12}) & a_2 \cos\theta_{12} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \tag{4.14}
$$

Once again, since the value ofθ_2 is not measured directly, the velocity of joint 2 is found by taking the time-derivative of (4.6) to get:

$$
\dot{\theta}_2 = \dot{\theta}^* - \dot{\theta}_1 \tag{4.15}
$$

4.3 Mechanical Design

4.3.1 Linkage

Figure 4.5 contains a simple diagram of the final mechanical design viewed from above, while Figure 4.6 shows a picture of the actual manipulator. The links are constructed out of ¼" aluminum, with low friction bearings and ½" diameter pins at each joint.

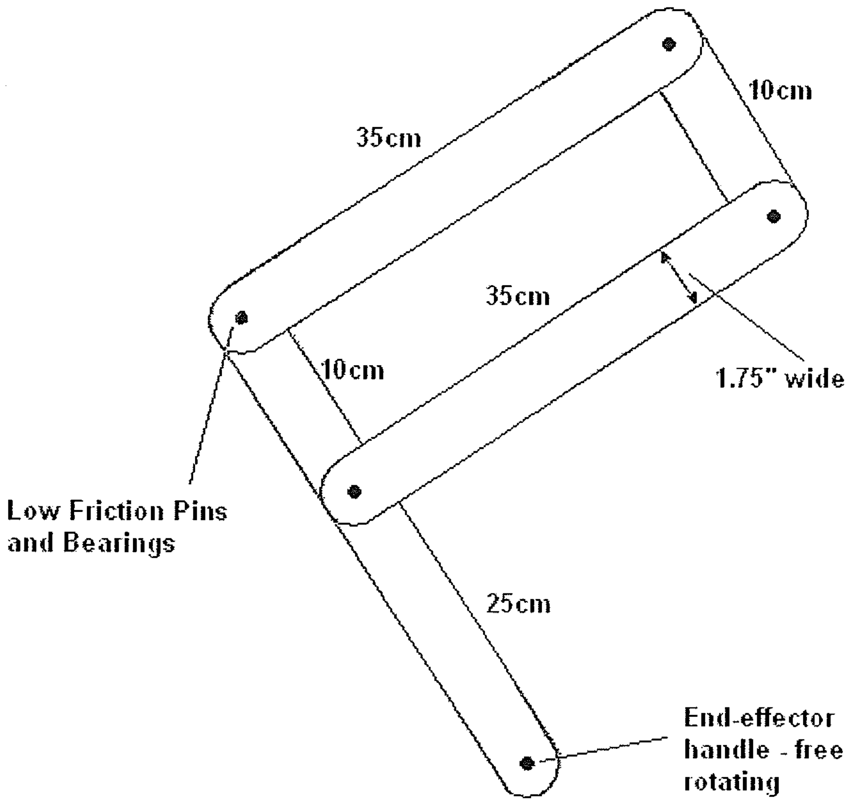

35cm
10cm
35cm
10cm
1.75" wide
Low Friction Pins and Bearings
25cm
End-effector handle - free rotating

Figure 4.5: Manipulator diagram.

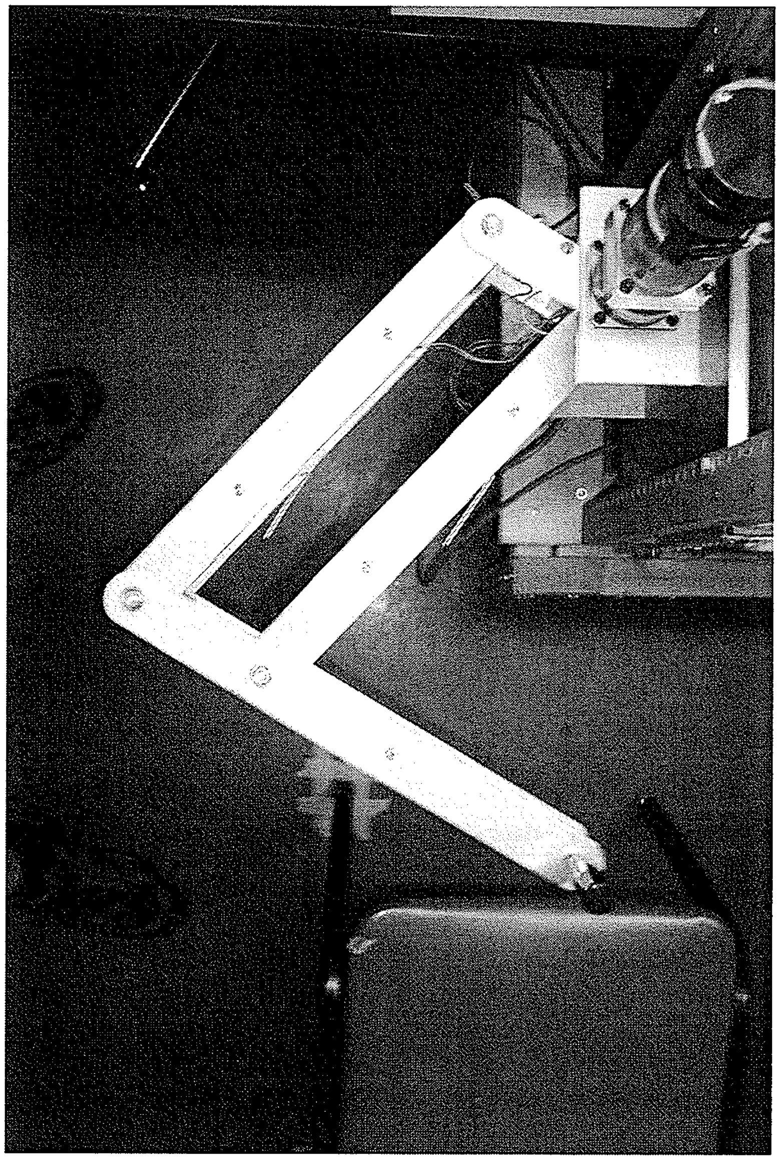


Figure 4.6: Manipulator image – Overhead view.

Figures 4.7-10 show a series of images that highlight the features of the manipulator. Note that each link is constructed from two ¼" aluminum pieces connected by pins along the length. This design was selected to prevent bending in the vertical direction, while at the same time limiting the weight of each link.

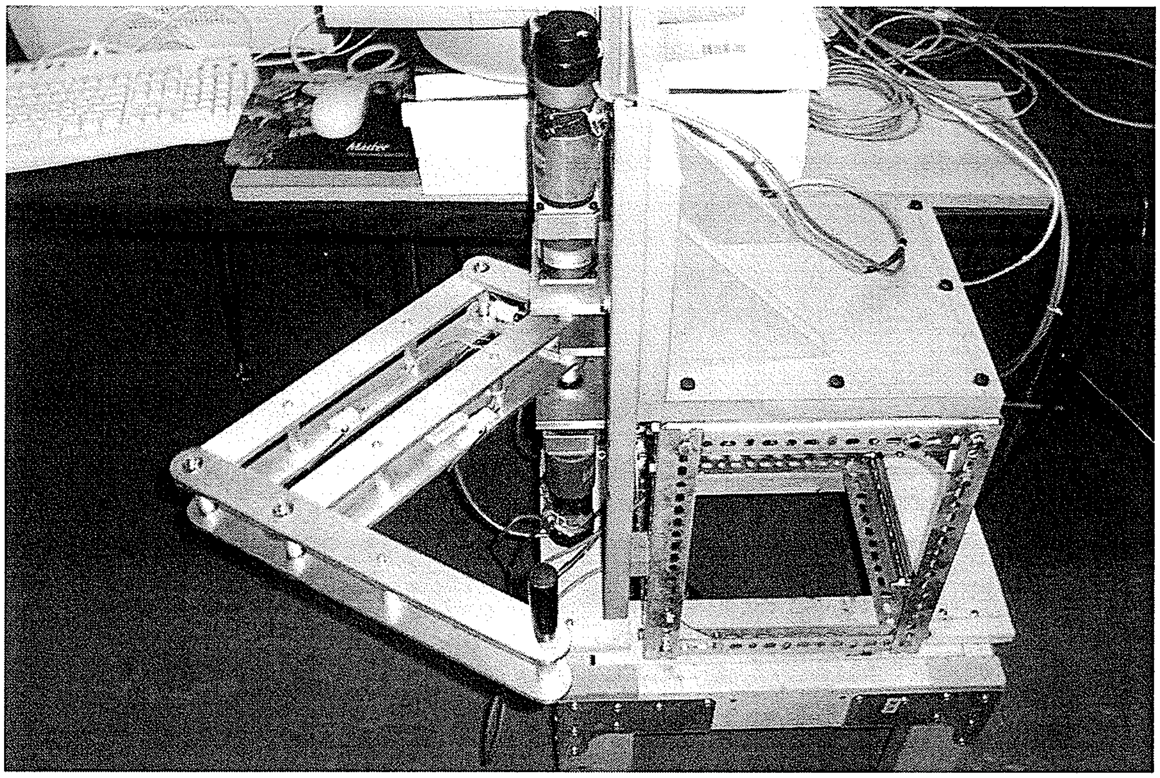


Figure 4.7: Overview of entire manipulator/support base system.

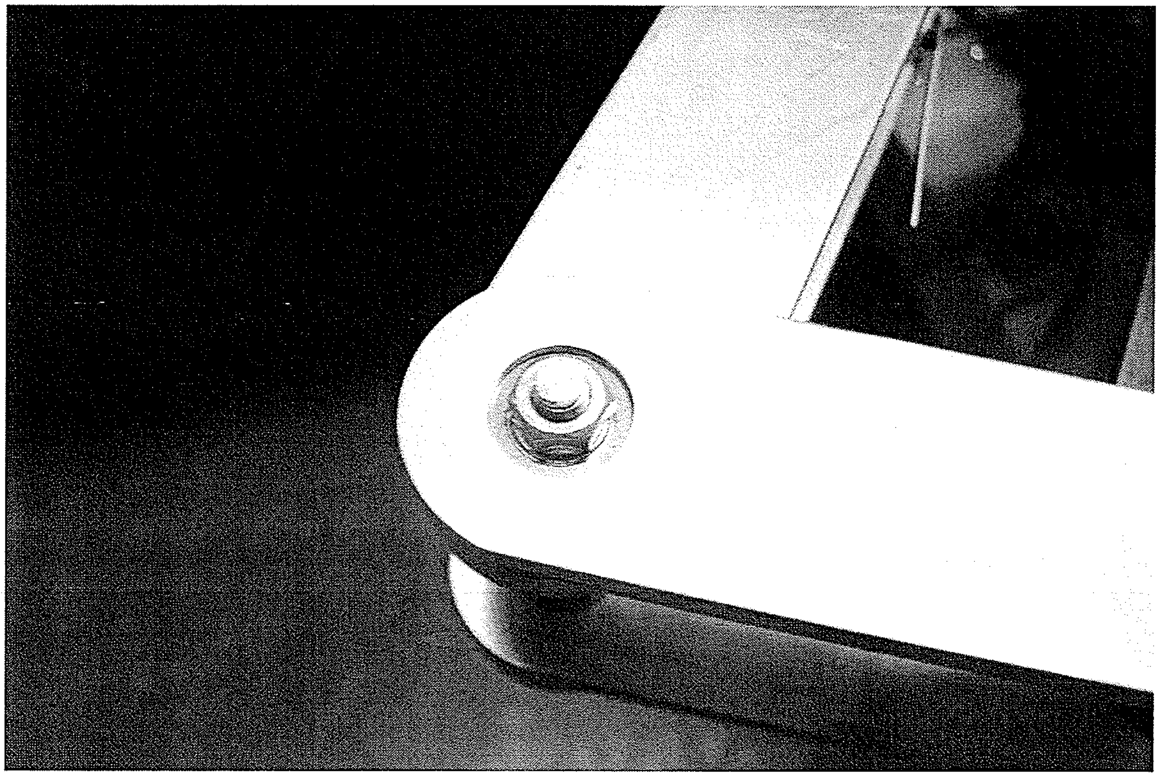


Figure 4.8: Low friction joint pinning.

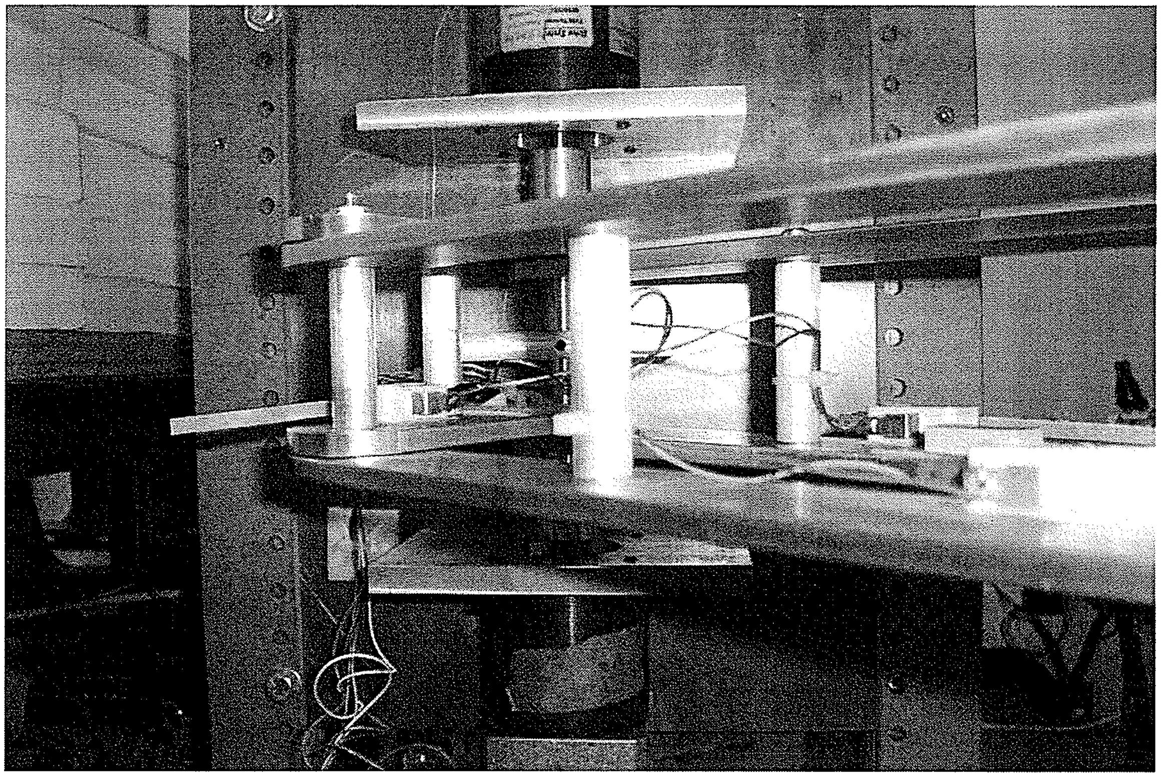


Figure 4.9: Side view of linkage.

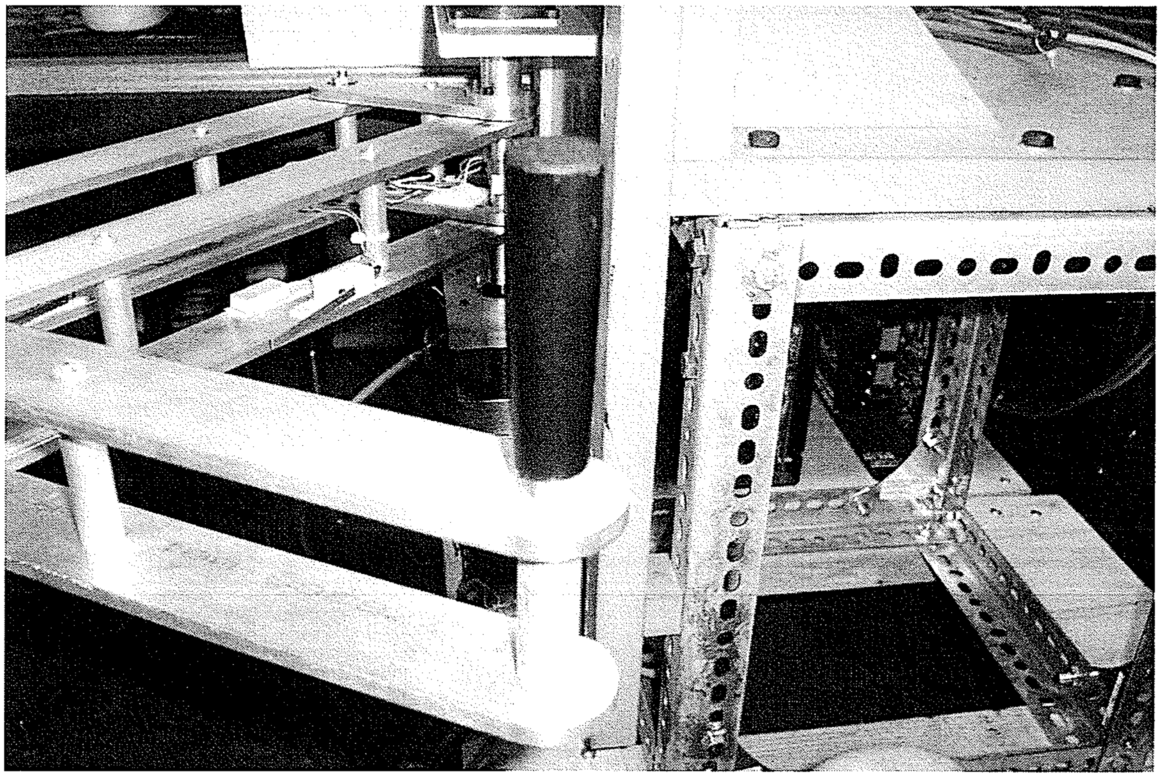


Figure 4.10: End-effector handle.

The wooden/steel support base to which the manipulandum is attached (Figure 4.7) was constructed to be of a considerable weight. This prevented any undesired shifting of the manipulator support during experiments. The series of holes tracking up the sides of the support in Figure 4.9 are mounting holes for changing the height of the manipulator with respect to the ground. This mechanism allows for the adjustment of the operating plane depending on the height of the test subjects or for different types of experiments. The black portion of the end-effector handle in Figure 4.10 is free to rotate about an axis perpendicular to the plane of operation.

4.3.2 Amplifiers and Motors

Since one of the goals of this work was to produce the manipulator at a low cost, the DC servo-motors and their accompanying drive systems were borrowed. The motors that were made available are the Electrocraft E560 brushed DC servo-motors. The amplifier drive system used is the Max-100 PWM servo drive, also from Electrocraft. The amplifier has a built in current control loop and can be configured to operate in 'torque' mode to control the current delivered to the motor. A block diagram of this current controller is shown in Figure 4.11, while a more complete description of the motor and amplifier interconnections can be found in Appendix A. For the manipulator dimensions in Figure 4.5, the torque generation capabilities of the manipulator were examined to determine what the minimum required joint torques would be to generate a at least 20N of force at the end-effector (in any direction). A series of test points were selected and analyzed using the inverse kinematic expressions from (4.8) and (4.11) to calculate the corresponding joint positions for each test point. These were then input

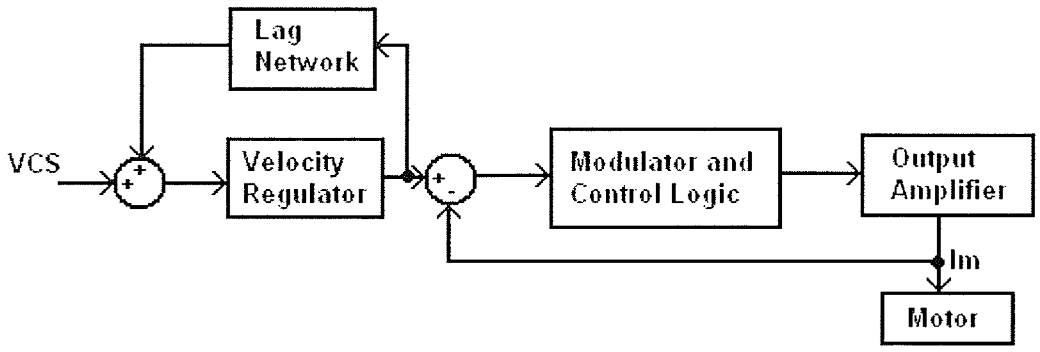


Figure 4.11: Max-100 current control circuit. [Reli92]
VCS: +/-10V control signal, Im: +/-3A motor current.

into the Jacobian of a virtual work calculation, (3.27), to determine the torques required to generate a given force in several different directions. The complete program listing and tabulated test results can be found in Appendix B. The results of the experiments showed that a torque of around 10 Nm would be needed. Since the available motors could not generate the required torque on their own, a planetary gearhead was added to each actuator. The specifications for the motors specified that their maximum torque output was 0.15 Nm. Therefore a gear reduction ratio of 70:1 was selected to satisfy the torque requirements.

With the motors and gearing now specified, the motor mounting system was designed to attach each actuator to the appropriate manipulator link. This mounting system is displayed in Figures 4.12-13.

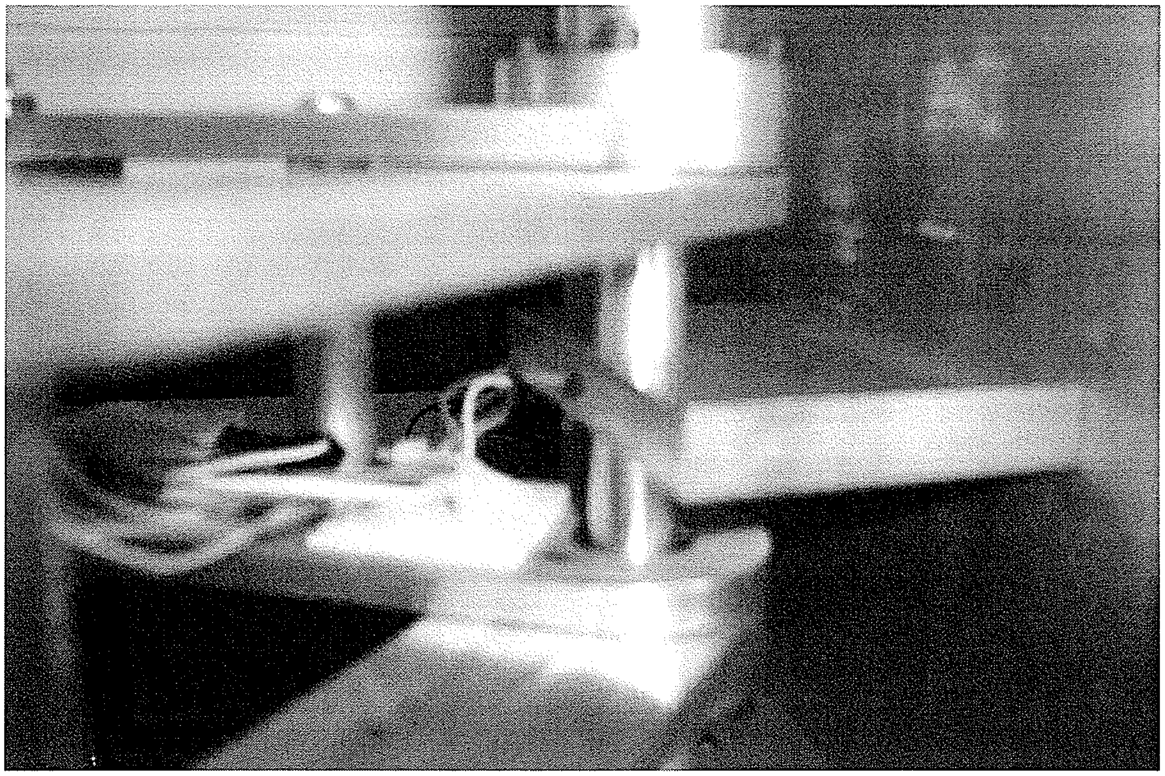


Figure 4.12: Base joint mounting location. Notice the shaft mount at the top of the image for connecting the link to the gearhead output shaft.

The final aspect of the drive system to consider is the friction compensation for the actuators. Since the output of each motor is passed through a gear reduction, any static and viscous friction effects from the motors are amplified by the gear reduction. Also, there are additional frictional losses due to the gearing components themselves. In order to keep the friction compensation as straightforward as possible, the coulomb friction model was used (Figure 3.5A), with the low velocity performance modified to use a piecewise linear model similar to Figure 3.5D. The final friction model selected is shown in Figure 4.14.

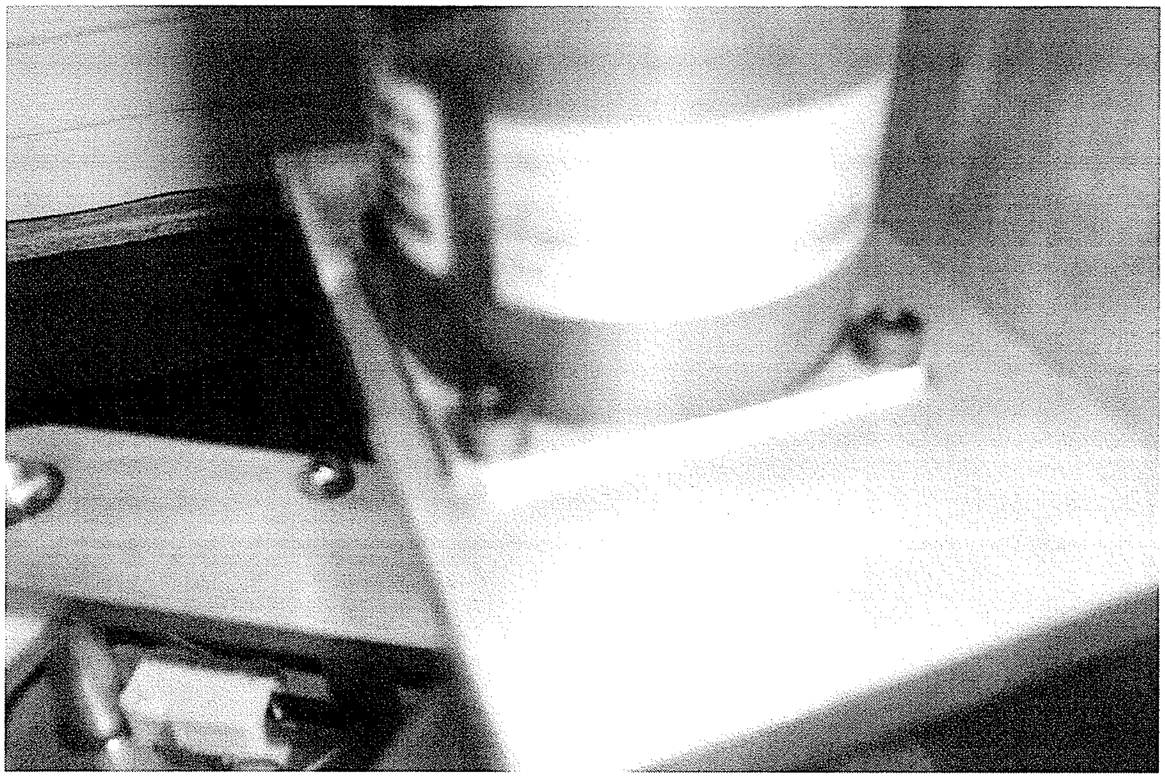


Figure 4.13: Top view of gearhead mount at manipulator base.
Output shaft feeds through to the shaft mount seen in Figure 4.11.

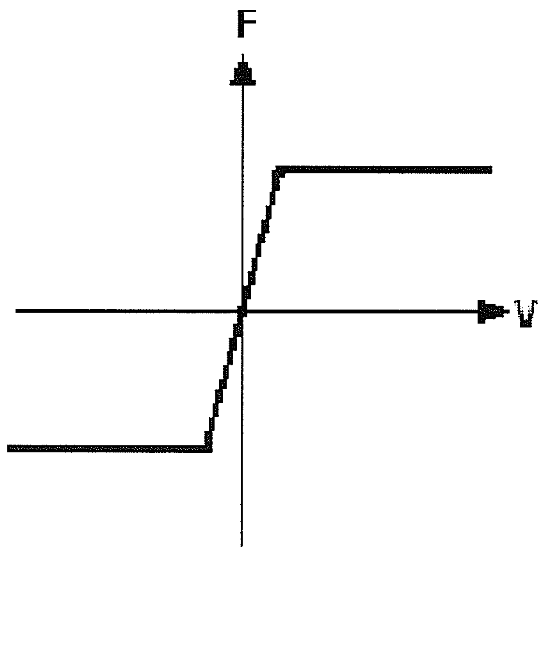


Figure 4.14: Selected friction compensation model.
Piecewise linear approximation to coulomb friction model.
V – velocity, F – friction compensation torque.

The plateau coulomb friction values were calculated by measuring the breakaway torque at several locations throughout the gearhead's rotation (with no links attached to the output shaft) and averaging the result. The breakaway torque was found by gradually increasing the current commanded to the motor and recording the current value for which the encoders on the motors detected motion [Arms88]. The results were internally consistent for each gearhead, with the breakaway torques found to be +/-0.86 Nm for the shoulder actuator and +/-1.02 Nm for the elbow actuator. The piecewise linear approximation for lower velocities was used to make the application of counter torques less sudden when starting a movement and were adjusted by 'feel' for each axis. The slopes of this centre line for the shoulder and elbow actuators were 3.8 N-sec and 3.5 N-sec respectively.

4.4 Safety Considerations

Since this manipulator is intended to interact with human test subjects, certain safety features were included during the design process. First there are forward and reverse limit switches for each joint. The Max-100 amplifier provides limit switch input terminals for the both the forward and reverse directions. The switches, which are visible in Figures 4.9-10, are attached to the manipulator linkages so that when a link comes into contact with a solid object at either end of its range of operation, the normally-closed (NC) contact is opened and the output stage of the amplifier is shut down for the direction associated with that switch. The forward limit switch input for the elbow motor amplifier and the reverse limit switch input for the shoulder amplifier are both connected to the same switch. This covers the case when the two 35 cm links of the parallel loop come into contact with each other. For a detailed circuit diagram of the limit switch connections, the reader is directed to Appendix A.

The other important safety feature is the kill switch, which is connected to the inhibit (INH) input on both amplifiers. If for some reason there is a catastrophic failure with any component within the system, the kill switch allows the experiment supervisor to disable the entire output stage of both servo drives, preventing injury to the subject and further damage to the equipment. The kill switch wiring is also presented in Appendix A.

CHAPTER V

SOFTWARE DESIGN AND IMPLEMENTATION

5.1 Hardware Interface

The impedance controller software is interfaced with the manipulator through a PCI-7344 Motion Controller from National Instruments [Nati01b]. The control signals from the PC are sent through a digital-to-analog, D/A, converter, while the position feedback from the motors is inputted to the PC through an encoder resource. The D/A takes a 16-bit digital signal and converts it to an industry standard +/-10V analog control signal, giving a quantization step size around 0.305 mV. The motors produce a quadrature encoder feedback signal that is used by the PCI-7344 to provide position and velocity information to the control software. The encoder produces 4000 counts/revolution which, once the 70:1 gear ratio is considered, provides a 280000 counts/revolution for the joint axes.

5.2 Software System Overview

5.2.1 Class Structure

The impedance controller software contains four main component objects (Figure 5.1). First, the Manipulator object is the software interface to the PCI motion control card. Second, the ControllerThread class is the actual impedance controller. Separate from this controller, the ExperimentThread object is responsible for the management of the test subject visual interface along with any data logging operations. Finally, the main

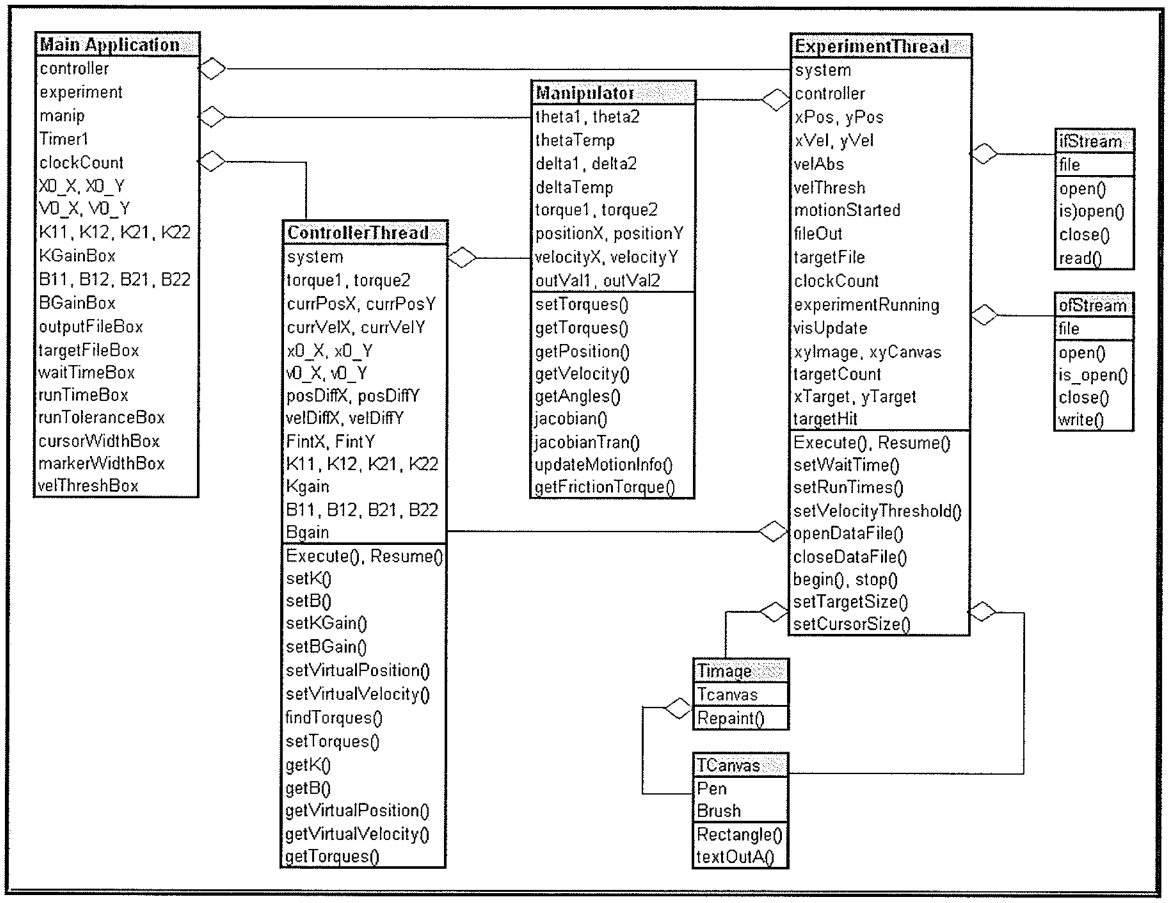


Figure 5.1: Software class diagram.

Win32 application system ties the other components together and manages the timer object and event handlers.

5.2.2 Thread Management

The controller application contains three main thread components, in addition to the main application thread, that interact to provide the desired system behaviour. Figure 5.2 shows the thread communication and timing relationships. The timer thread generates an event every 10 ms, which triggers the update of the controller and experiment threads. This results in a 100 Hz update rate for the system.

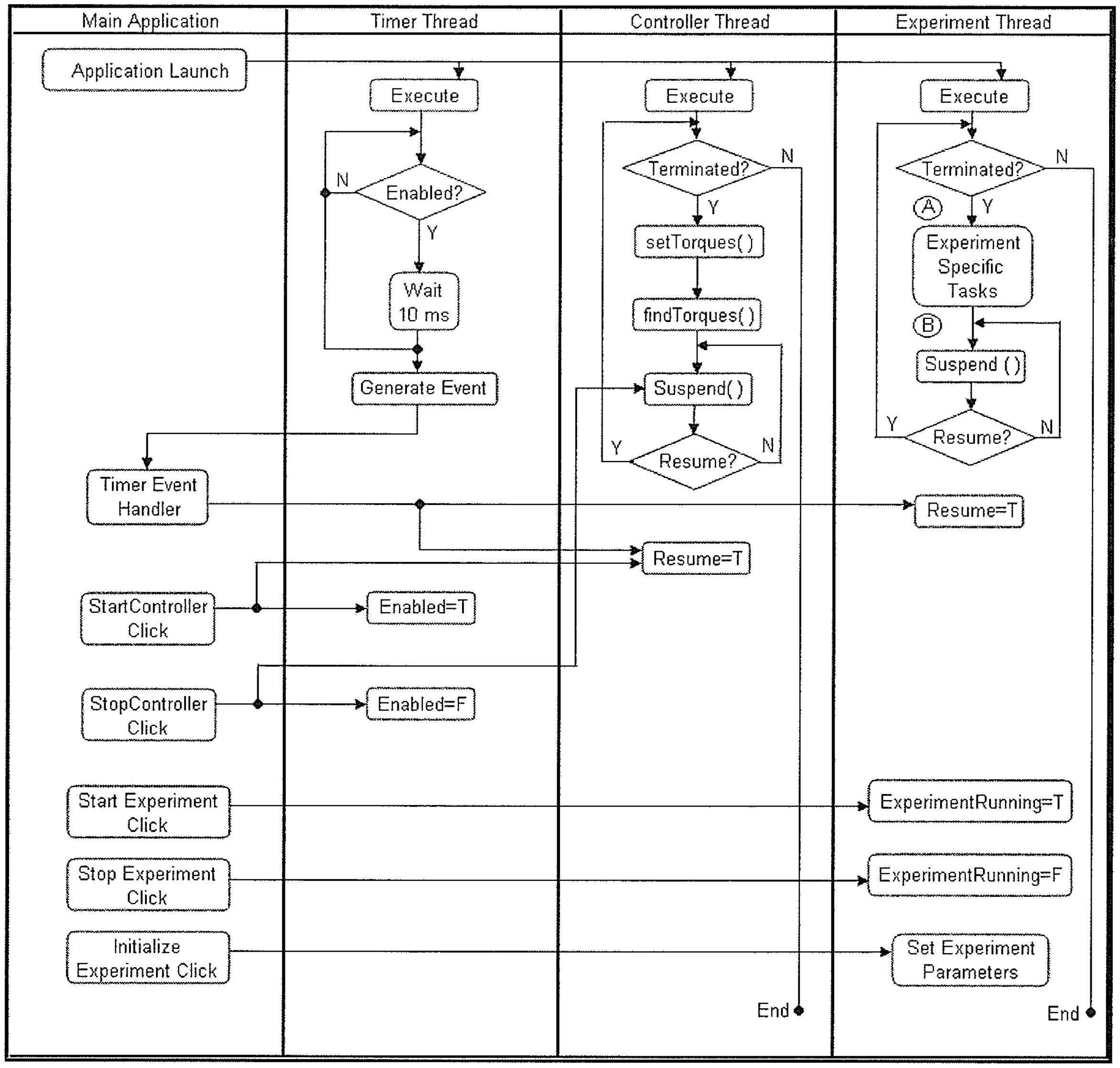


Figure 5.2: Thread timing and communication.

5.3 Manipulator Object

The source code for the Manipulator object, *Manipulator.hpp* and *Manipulator.cpp*, can be found in Appendix C. This class makes use of the National Instruments FlexMotion C++ API to interface with the PCI-7344 card [Nati01a]. The Manipulator class is also responsible for performing the kinematic and Jacobian calculations based on the current joint trajectories, as presented in Section 4.2. The

Cartesian positions and velocities resulting from these calculations are then stored so that they may be accessed by the controller and experiment threads as required.

5.4 Controller Thread

The source code for the impedance control object, *ControllerThread.h* and *ControllerThread.cpp*, is presented in Appendix D. This software component contains a reference to the Manipulator object instance, for which it implements the impedance control algorithm of Section 3.5. When the controller receives notification of a timer event, the torque values calculated during the previous control loop are sent to the Manipulator object. This is followed by the collection of position and velocity information from the Manipulator and the calculation of the required torques to output during the next control loop. This basic operation was shown previously in the 'Controller Thread' section of Figure 5.2.

In order to allow the software user to manually update the impedance control parameters, the visual interface for the program contains input fields for the virtual position/velocity and the stiffness and viscosity matrices (Figure 5.3). The K and B can be selected by changing the matrix parameters, and also can be adjusted by a separate gain factor. Figure 5.3 also shows the Start/Stop buttons for the impedance controller. When the controller is stopped, no torque values are sent to the manipulator, hence there is no friction compensation performed by the controller.

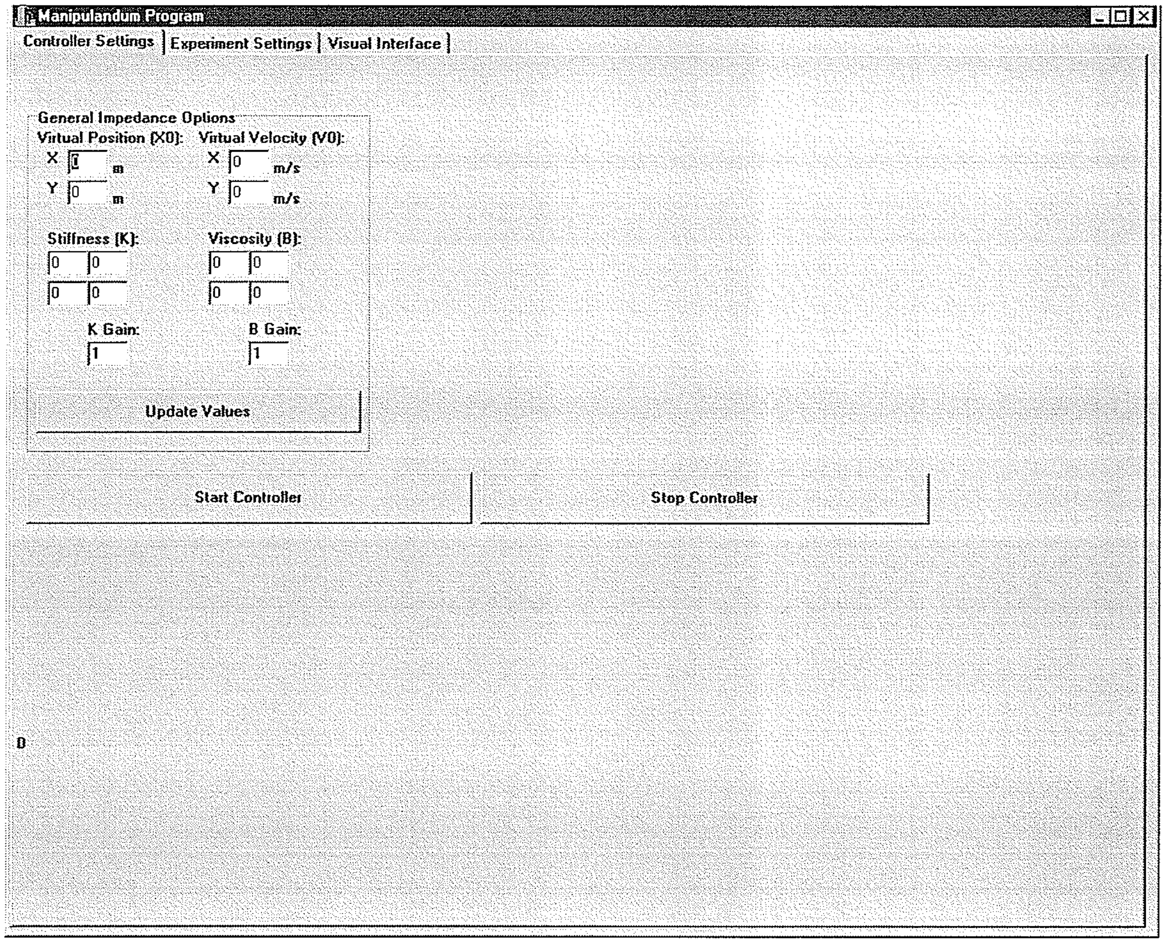


Figure 5.3: Impedance controller interface.

5.5 Experiment Thread

The experiment source code, *ExperimentThread.h* and *ExperimentThread.cpp*, can be found in Appendix E. This class is responsible for performing any experiment specific tasks (see the 'Experiment Thread' section of Figure 5.2). In particular, this involves the collection and storing of trajectory and timing data in data files. The basic procedure is the same for each of the experiments performed in this work, and can be seen in Figure 5.4. The common experiment behaviour is discussed in this section, while the differences are outlined in Chapter 7.

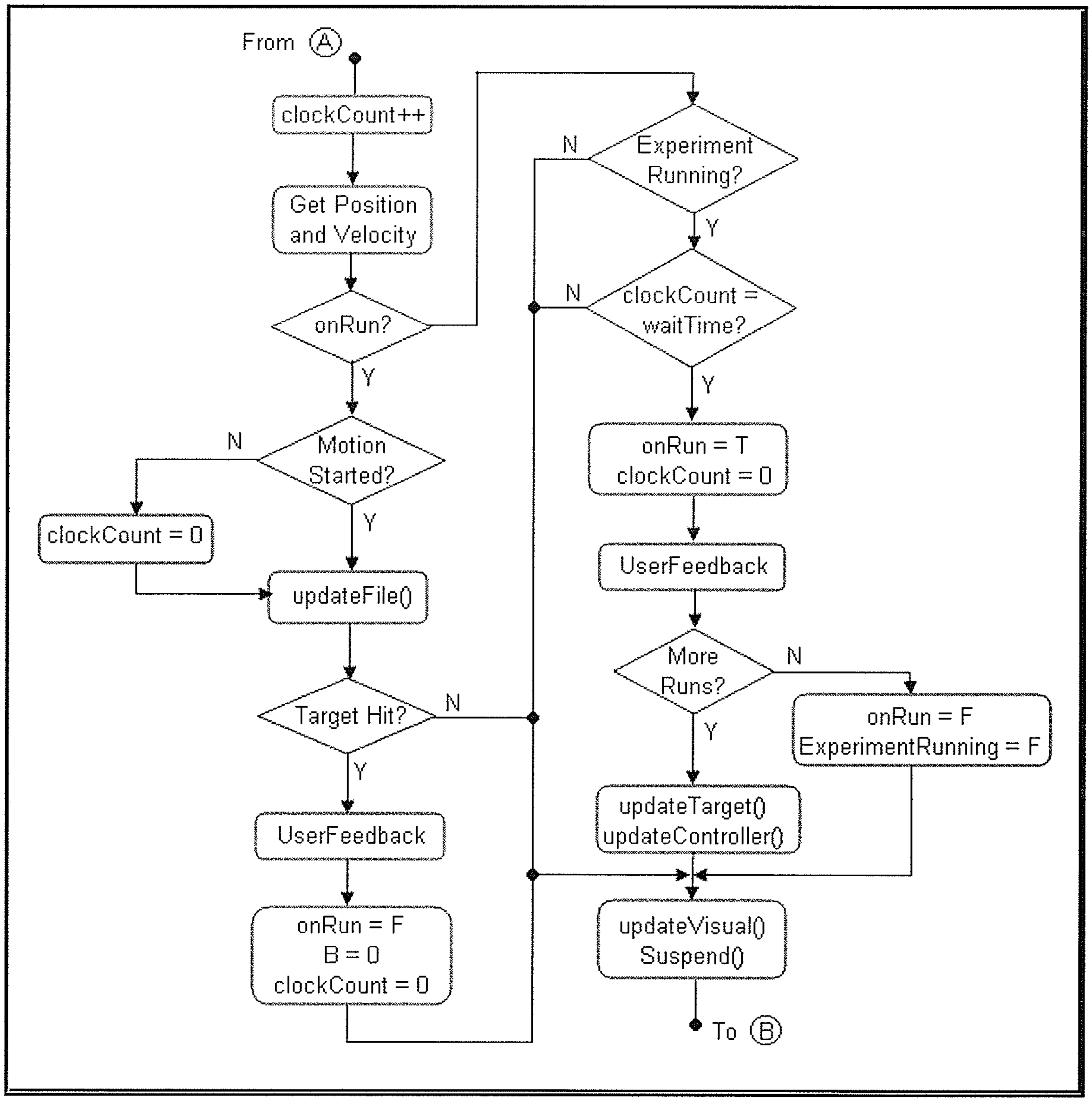


Figure 5.4: Experiment procedure program flow.

For each experiment, the test subjects are presented with a visual interface as shown in Figure 5.5. The centre of this workspace corresponds to $(x, y) = (40,0)$ cm in the robot's coordinate frame. The subjects are shown a series of movement targets and are instructed to move their cursor (small white square), which corresponds to the location of the robot handle, to each target location (large blue square) within 0.4 to 0.6 seconds. This timing of each trial starts once the subject's hand velocity exceeds an

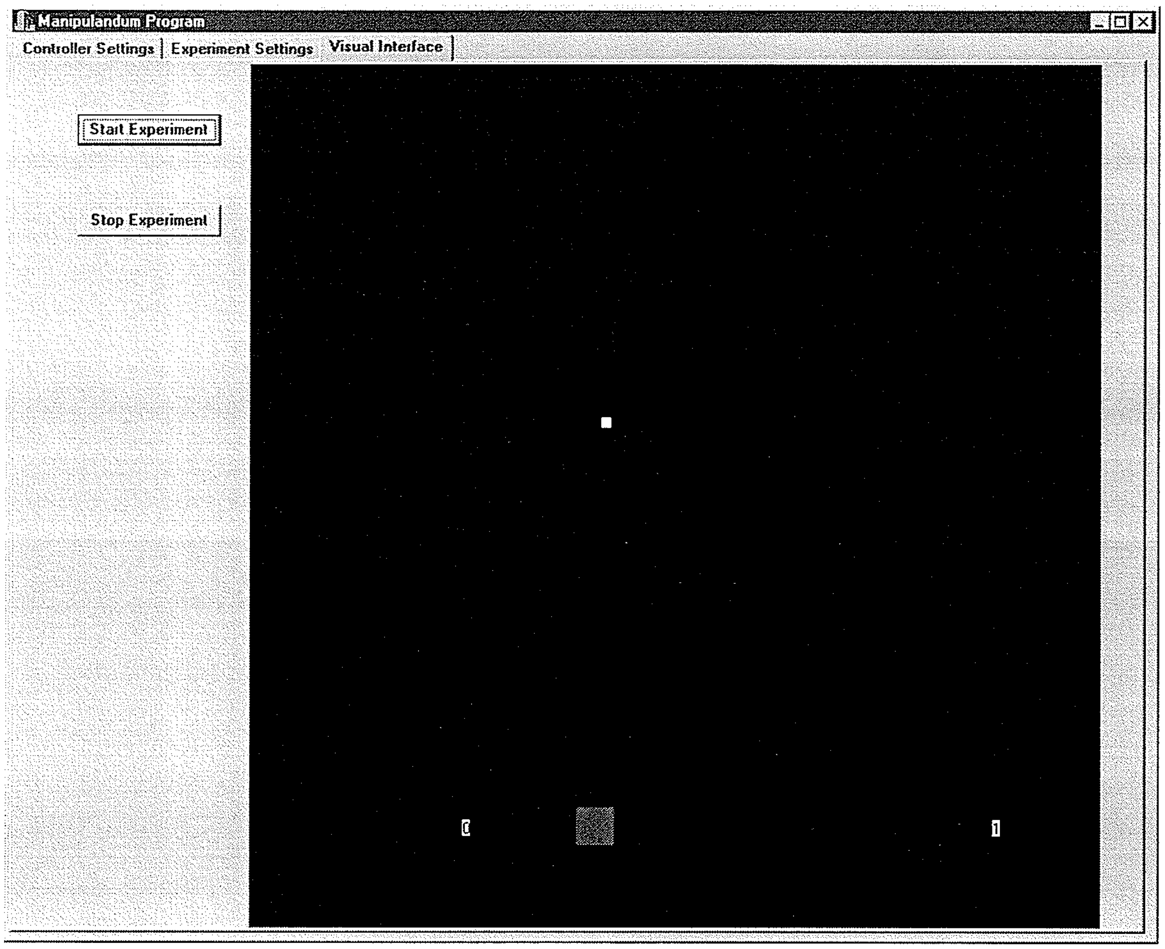
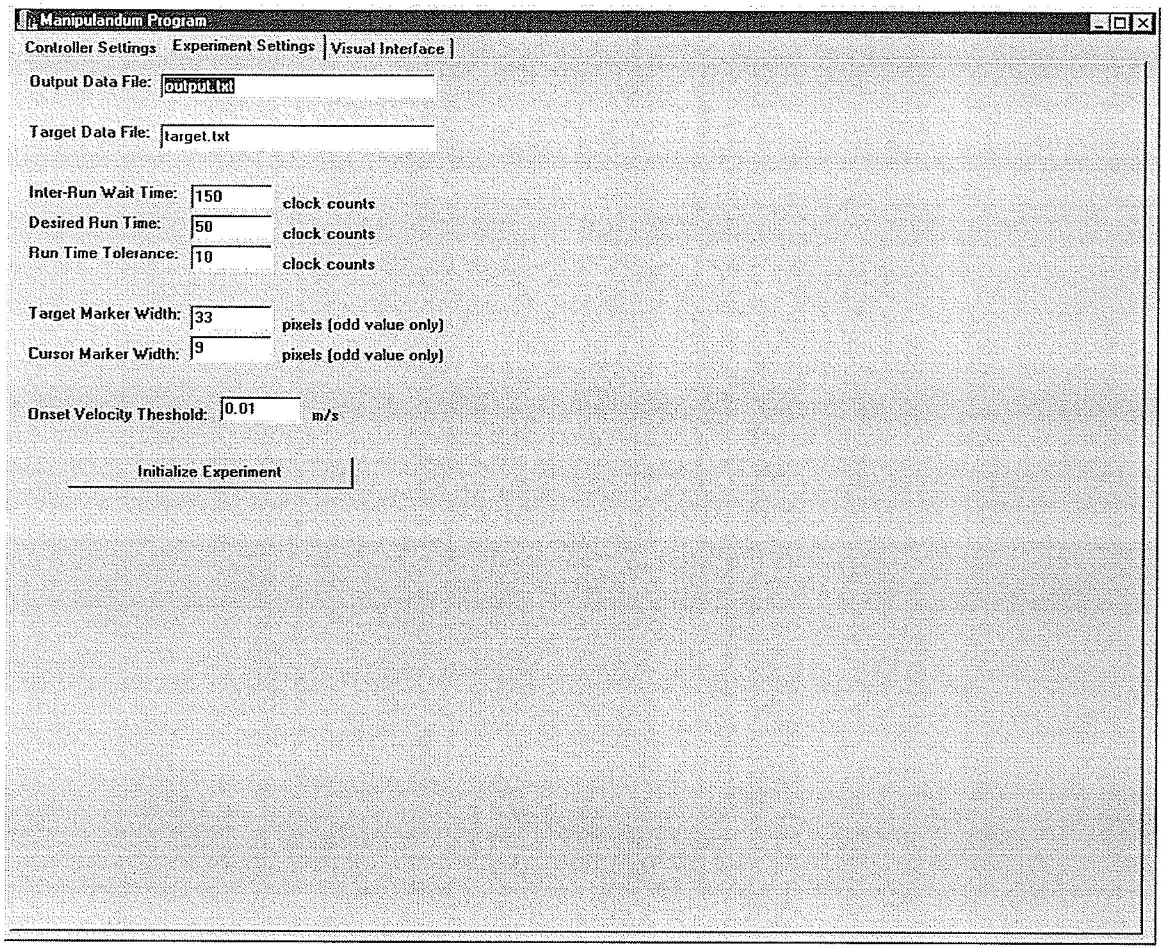


Figure 5.5: Experiment visual interface.

onset threshold of 0.01 m/sec. If the subject reaches the target within the desired time, the target turns from blue to green. If the subject is too slow or too fast, the target turns either yellow or red respectively. Along with this visual feedback, the subjects also receive an audio indicator of their performance. When each target appears on the screen, a tone is emitted by the PC, and when the subject reaches the target, he/she hears a second tone. If the subject achieves the desired movement time, the second tone is the same frequency as the first. If the movement is too fast or too slow, the tone goes respectively up or down in frequency. These colour and tone parameters are specified during the *UserFeedback* blocks in Figure 5.4.



Output Data File: output.txt

Target Data File: target.txt

Inter-Run Wait Time: 150 clock counts
Desired Run Time: 50 clock counts
Run Time Tolerance: 10 clock counts

Target Marker Width: 33 pixels (odd value only)
Cursor Marker Width: 9 pixels (odd value only)

Onset Velocity Theshold: 0.01 m/s

Figure 5.6: Experiment settings interface.

In order to allow for some experiment flexibility, the experiment supervisor is able to change the desired run time, the target or cursor size, the inter-target wait time and the velocity-onset threshold using the interface shown in Figure 5.6. The timing parameters are specified in clock counts, with each count representing 10 ms. The desired run time range is controlled by setting a target time and specifying a desired tolerance range. The inter-run wait time is the amount of time the system waits after the subject reaches a target before presenting the next target. The cursor and target sizes are given by their width in pixels.

During each target trial, the manipulandum is programmed to exert a specific velocity-based force field to perturb the subject's motion. The *ExperimentThread* class is responsible for determining what the parameters of this field will be depending on the conditions of the experiment (ie: target directions).

The experiment control software is also responsible for logging the position, velocity and timing information for each trial. The output file is specified by the experiment supervisor using the interface in Figure 5.6. For each 10 ms clock cycle, the clock count, position and velocity are written to the next row of the file in ASCII format with the following format:

clockCount X-position Y-position X-velocity Y-velocity

CHAPTER VI

SYSTEM VERIFICATION

This chapter presents the details and results of the experiments performed to characterize the system performance. Since several of components were donated, the first verification stage was to verify the characteristics of the individual components. This performance was then compared to that described in the specifications provided with the equipment. The resulting verified performance attributes were integrated into the manipulator control code in Appendix C. These verification experiments are then followed by an examination of the friction properties and performance of the drive system and then by a discussion of the overall system performance.

6.1 Amplifier Characteristics

The specifications for the MAX-100 amplifiers provided in [Reli92] state that the $VCS = \pm 10V$ control signal corresponds to an approximate motor current of $I_m = \pm 3A$ (see Figure 5.11). This meant that the expected amplifier gain was $0.3A/V$. To verify this relationship, each amplifier was connected configured in a 'torque mode' of operation and connected to a motor (see Appendix A). An ammeter was inserted in series with the motor to measure the actual current delivered by the amplifier. The output shafts of the motors were fixed in order to allow for an analysis of the full output range of the amplifiers. Plots of the resulting experimental data are shown in Figures 6.1 and 6.2.

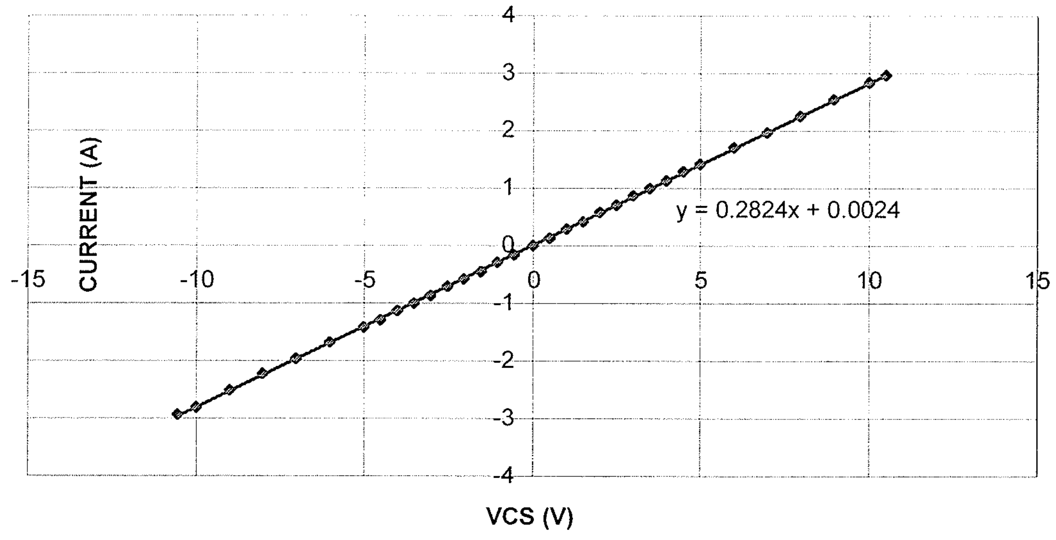


Figure 6.1: Amplifier #1 current vs. voltage characteristic.

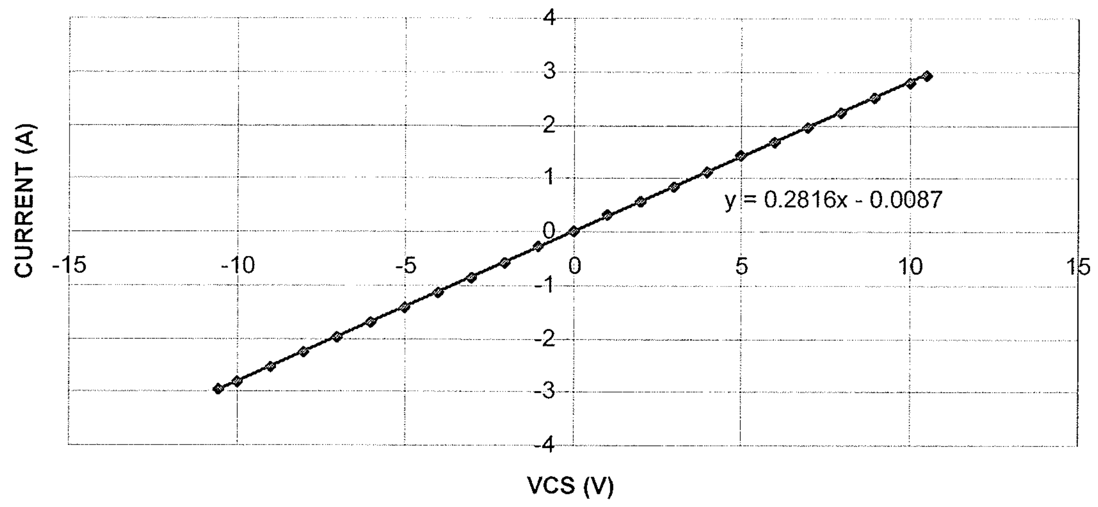


Figure 6.2: Amplifier #2 current vs. voltage characteristic

The linear trendlines fitted to the amplifier characteristic plots provide the experimentally determined amplifier gains. These values are $2.824A/V$ and $2.816A/V$ for amplifiers 1 and 2 respectively, which are close to the expected value reported in the literature. As a result of this analysis, the amplifier gain used in the control software was the average of these two values, 2.82*A/V*.

6.2 Encoder Feedback

The rotary encoders provided with the E560 motors supply a quadrature output signal with 4000 counts per revolution (counts/rev). This signal is sent to the encoder input of the PCI-7344 Motion Controller from National Instruments [Nati01b]. Using the software API for this card the current position and velocity of the axis are available in counts and counts/rev respectively. This section discusses the verification of these values returned from the motion controller.

6.2.1 Position

To verify the position data, the output shaft of the motor was rotated to several locations and the actual angular position was recorded. This value was then compared to the encoder count value returned by the controller API. The resulting data is presented in Table 6.1. The average error of the returned values was 1.35° or 15 encoder counts. This error could be attributed to the inability to place the motor in the exact desired location and may not be the result of the motion controller processing. The end result of this verification was the determination that the motion controller software was providing accurate position data. This analysis was also repeated while the 70:1 gearhead was attached to the motor. This configuration provided an encoder resolution of 280,000 counts/rev. The results of this analysis are presented in Table 6.2. The average error in this case was found to be 0.09° or 70 encoder counts.

Table 6.1: Encoder position data

Actual Angle (degrees - ccw)	Encoder Value (counts)	Measured Angle (degrees - ccw)	Actual Angle (degrees - cw)	Encoder Value (counts)	Measured Angle (degrees - cw)
10	-115	10.35	10	130	11.7
20	-219	19.71	20	226	20.34
30	-335	30.15	30	325	29.25
40	-428	38.52	40	420	37.8
50	-533	47.97	50	553	49.77
60	-643	57.87	60	633	56.97
70	-740	66.6	70	739	66.51
80	-862	77.58	80	880	79.2
90	-968	87.12	90	1002	90.18
120	-1314	118.26	120	1345	121.05
150	-1653	148.77	150	1679	151.11
180	-1976	177.84	180	2000	180
270	-2993	269.37	270	2987	268.83
360	-3991	359.19	360	3999	359.91
540	-5980	538.2	540	6040	543.6
720	-7984	718.56	720	8000	720
1080	-11994	1079.46	1080	12002	1080.18

Table 6.2: Encoder position data with gearhead

Actual Angle (degrees - ccw)	Encoder Value (counts)	Measured Angle (degrees - ccw)	Actual Angle (degrees - cw)	Encoder Value (counts)	Measured Angle (degrees - cw)
90	-69944	89.93	90	69999	90.00
180	-139939	179.92	180	139998	180.00
360	-280179	360.23	360	279866	358.66

6.2.2 Velocity

With the accuracy of the position data determined, a brief experiment was performed to find the accuracy of the velocity values. The motor, with gearhead attached, was mounted with the shaft positioned vertically (upward). The motor was run at several speeds and the corresponding velocity value from the motion controller was recorded. This value was determined by averaging the instantaneous velocity measurements over the run time. The actual velocity was measured by using a stopwatch to find the time taken for a given number of revolutions. The resulting data is presented in Table 6.3. The average error in this data was found to be 0.37 revolutions per minute (rpm). Again, this is a low error, considering the errors possible in measuring the actual velocity. Therefore, it was verified that there is accurate velocity data being returned by the motion controller API.

Table 6.3: Encoder velocity data with gearhead

Actual Velocity (rpm – ccw)	**Encoder Value (counts/sec)**	**Measured Velocity (rpm – ccw)**	**Actual Velocity (rpm – cw)**	**Encoder Value (counts/sec)**	**Measured Velocity (rpm – ccw)**
30.7	144667	31	20.6	93333	20
66.2	307533	65.9	55.6	261333	56
94.6	443800	95.1	87.1	406000	87

6.3 Motor Torque Constant

Due to a lack of torque measuring devices, the motor torque constant was determined by the following experiment. The motor, without a gearhead, was mounted with the output shaft oriented horizontally and leveled. A piece of dowelling (17 mm radius) was mounted on the shaft and a known mass was suspended from the dowel (Figure 6.3). The current corresponding to the minimum torque required to hold the mass was measured and the relationship between torque and current was found. The resulting data for the two motors is shown in Figure 6.4 and 6.5. The slopes of the trendlines in these plots indicate the experimentally determined torque constants for the motors

(0.055*Nm/A*). This was consistent with the 0.0542 *Nm/A* torque constant specified in the literature [Reli92]. It was decided that due to potential errors in the experimental process, it would be reasonable to use the manufacturer's supplied torque constant in the manipulator control software.

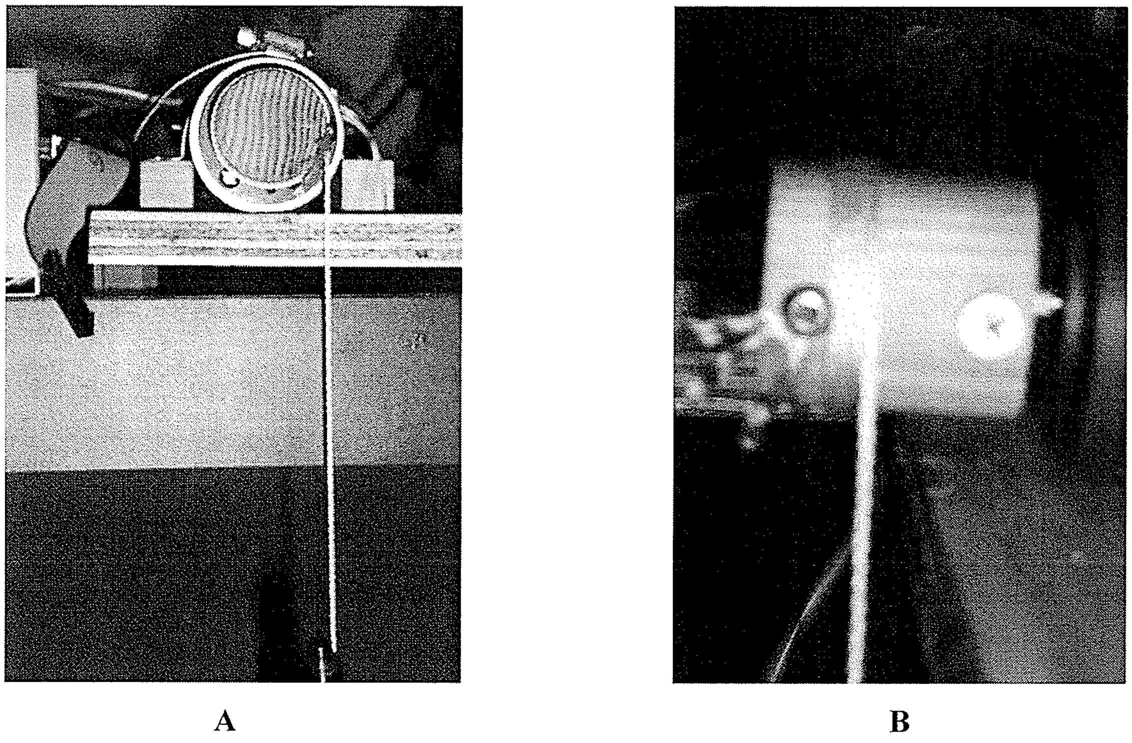


Figure 6.3: Torque constant apparatus **A:** Front view, **B:** Side view

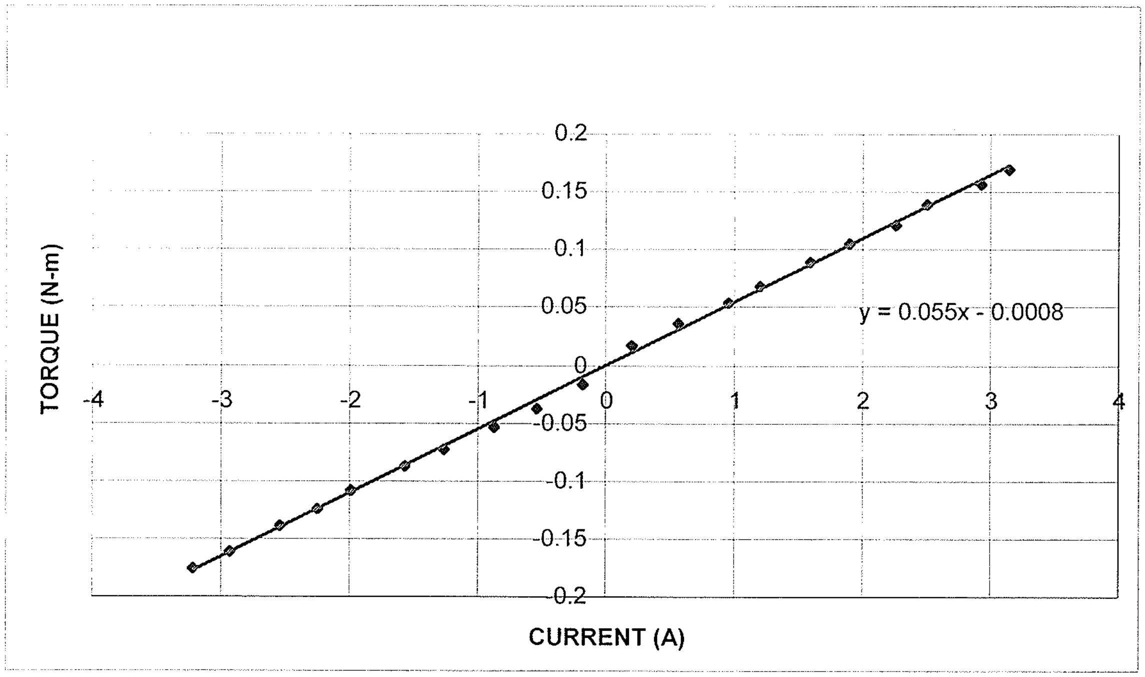


Figure 6.4: Motor #1 torque vs. current characteristic

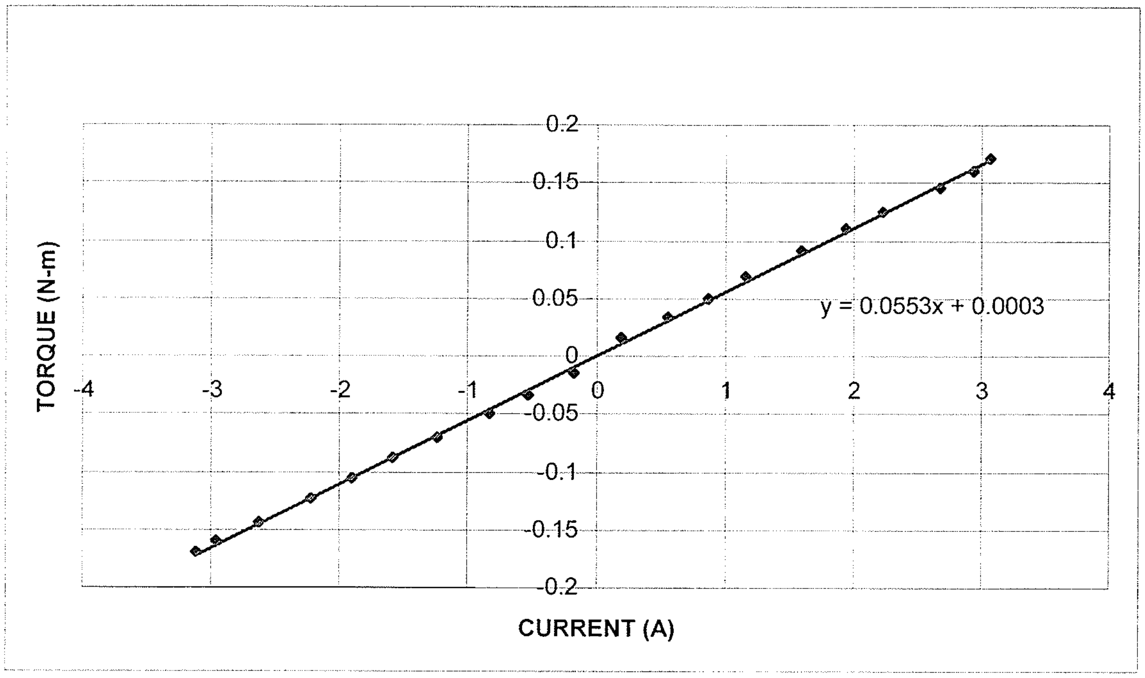


Figure 6.5: Motor #2 torque vs. current characteristic

6.4 Motor and Gearhead Friction

The plateau friction value from Figure 4.14 was determined by calculating the breakaway torque values for the motor/gearhead combination. The final measured breakaway torque for each motor was selected to be the average of eight values taken at different output shaft orientations. The breakaway torque was found by gradually increasing the torque commanded to the motors, with gearheads attached, in increments of 0.01 Nm and recording the torque at which movement of the output shaft started [Arms88]. The results for the two motors, shoulder (#1) and elbow (#2), can be found in Tables 6.4 and 6.5 respectively. The average breakaway torque for the shoulder motor was found to be 0.94 Nm, while the average for the elbow motor was 1.14 Nm.

Table 6.4: Motor #1 breakaway torques

Breakaway Torque – cw (Nm)	**Breakaway Torque – ccw (Nm)**
0.86	0.99
0.97	1.08
1.01	0.88
0.89	0.86

Table 6.5: Motor #2 breakaway torques

Breakaway Torque – cw (Nm)	**Breakaway Torque – ccw (Nm)**
1.16	1.18
1.14	1.10
1.18	1.13
1.12	1.1

Once the overall system was constructed and tested, these values were found to be too high. Since the subject moving the handle was responsible for starting the motion of the robot instead of the motors, these breakaway values generated a torque that 'helped' the subject. This meant that the subject did not feel a near-frictionless movement at the end-effector, but rather felt like the robot was pulling their hand faster. To compensate for this, the torque values were reduced by feel and the final torque values for the shoulder and elbow were 0.86 Nm and 1.02 Nm respectively.

6.5 Amplifier Step Response

The step response of the motor/amplifier combination was found by applying a step input to the amplifier input and measuring the rise time for the amplifier output current. The rise time was found by connecting an oscilloscope to the MCO output of the amplifier, which provides a voltage signal proportional to the motor current. The step response, Figure 6.6, was found to be a first order rise with a full rise time of 2ms.

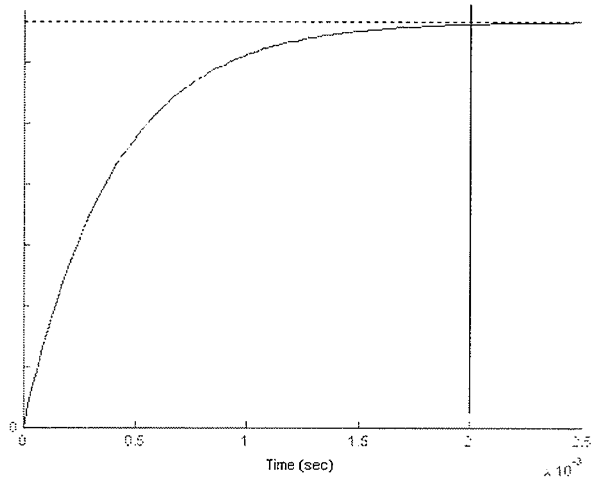


Figure 6.6: Amplifier step response.

This rise time is lower than the sampling period of the controller (10 ms), so that by the time the next control loop starts, the previously commanded currents have settled to their steady-state values.

6.6 Overall Performance Limitations

Due to the gearing used on the motors, the simplified friction model used and the fact that the inertial properties of the robot linkage were ignored by the control software, there are several limitations to the performance of the manipulandum system.

6.6.1 Gearhead Backlash

The gearheads used for this system have a backlash of 16 arcmin or 0.267°. This fairly large backlash is due to the larger gear ratio selected to accommodate the torque requirements of the experiments. This backlash means that there is a 'dead-zone' of rotation at the output shaft of each gearhead. This causes related dead-zones in the X and Y directions seen at the end-effector. The size of this zone when the robot handle is placed in the middle of the workspace, X = 40 cm, Y = 0 cm, is 1 mm in the X direction and 2.4 mm in the Y direction. Note that these are the worst-case displacements at this location. The main effect of this backlash from a performance perspective is that the subjects have problems starting and stopping their movements in a smooth manner. However, since the movement error calculation does not involve these particular points this effect should be minimized. But, there is still potential for additional error during a given trial, which should result in a larger standard error in the movement error calculations.

6.6.2 Force Field Generation

Since the friction model used only handles coulomb friction, the model is not completely accurate. As a result, the subjects do not encounter a seemingly 'frictionless' environment. This means that the interaction forces caused by the friction torques will affect the force fields generated by the manipulandum. In particular, for some directions, the subject's motion may be made to follow the robot's preferred direction to a certain extent. This can occur because the force applied by the subject may not be high enough to overcome the friction of one link (ie: the upper arm of the robot). In this case, the outer limb of the robot may simply rotate about the nearly fixed distal end of the upper arm link and cause the subject to follow a slightly curved path. This is particularly evident for Directions #4 and #8 (see next chapter). The combined effect of the backlash and friction issues made movement in directions 1 and 5 quite difficult and inconsistent between subjects, so these directions were excluded from the experimental procedures described in the next chapter.

CHAPTER VII

EXPERIMENT DESIGN

7.1 Experimental Setup

7.1.1 Physical Configuration

The experimental setup is presented in Figure 7.1. A computer monitor displaying the interface from Figure 5.5 was positioned such that the centre of the workspace was approximately level with subject's eyes. The height of the manipuladnum was set in a subjectively 'comfortable' position for each subject (approximately level with the *solar-plexus*). This was one of the primary differences between this work and past experiments. Previous studies were performed with the subject's arm artificially abducted $90°$, in order to keep the subject's arm in the same plane as the robot. However, the movements performed in this abducted position are not very natural and can be quite difficult in some directions. The configuration used in this work allowed for a study of adaptation in a more natural position that more closely emulates everyday reaching movements.

7.1.2 Target Directions

As mentioned in Section 5.4, the task being learned by the test subjects was to reach a displayed target within 0.5 ± 0.1 seconds. These targets were in one of six possible directions as shown in Figure 7.2. The horizontal directions (dashed lines) in this figure, labeled as 1 and 5, were not used in this experiment due to the performance limitations listed in the previous chapter. It should be noted that the arrangement of

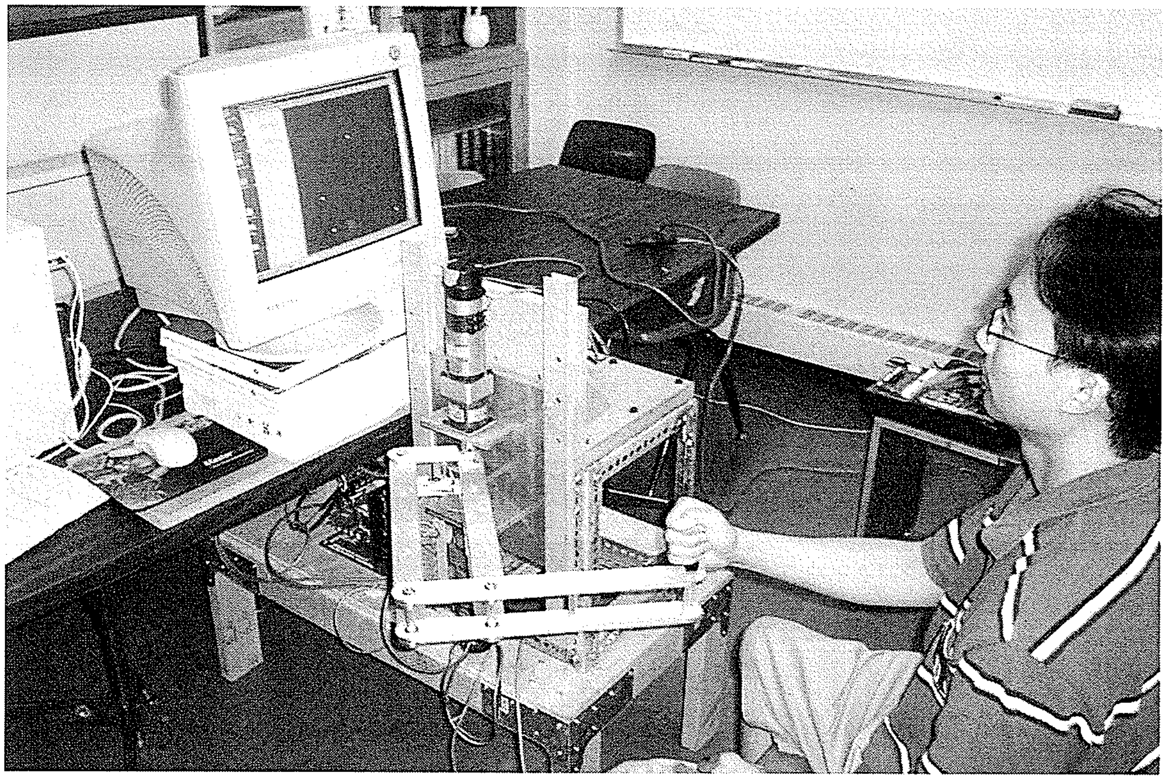


Figure 7.1: Experimental setup.

target directions shown in Figure 7.2 does not represent the configuration of targets actually shown to the subjects. In order to keep the robot dynamics consistent in each direction and to keep the movements closer to the centre of the robot's workspace, every odd target trial began at a given starting location and moved outwards. The following trial was selected to be in the opposite direction moving from the previous target location back to the starting position (Figure 7.3). As a result, all movements in the same direction had the same starting and ending locations. The centre starting point was offset from the centre of the Figure 5.5 workspace by moving it 2 cm up and 2 cm to the left. This offset insured that the outward movements did not end too close to the limits of the robot workspace.

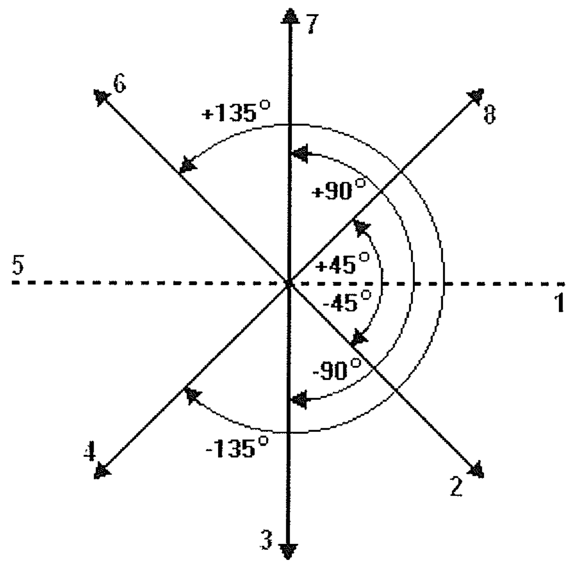


Figure 7.2: Target directions.

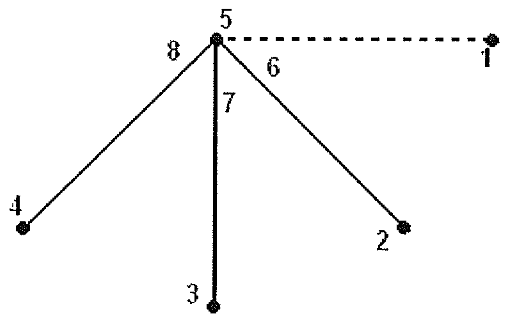


Figure 7.3: Actual target configuration seen by subjects.

7.1.3 Force Field Design

For movements in each direction a velocity-based viscous force field was applied to perturb the subject's movements. Since the force field is based on velocity, independent of position, the impedance controller equation from (3.23) has $K = 0$ and $V_0 = 0$, resulting in the following force/velocity relationship.

$$F_{\text{int}} = -BV \tag{7.1}$$

The viscosity matrix, B, was selected such that the perturbing force was perpendicular to the direction of motion, with a magnitude proportional to the velocity along the straight line joining the beginning and end points of each movement. For the directions parallel to the robot's base coordinate frame, the viscosity matrix is:

$$B = B_i \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix} N\sec/m\,, \quad \textit{for Directions \#1 and \#5} \tag{7.2}$$

$$B = B_i \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} N\sec/m\,, \quad \textit{for Directions \#3 and \#7} \tag{7.3}$$

where, B_i, is the matrix gain. For the diagonal directions, the velocity vector must be rotated by -45° to align with one of the parallel axes before determining the desired force. Once the force is found, the result is rotated back to the original direction. Using (3.3), the rotation matrices required for these operations are determined to be:

$$\begin{aligned} R(Z,-45^\circ) &= \begin{bmatrix} \cos(-45^\circ) & -\sin(-45^\circ) \\ \sin(-45^\circ) & \cos(-45^\circ) \end{bmatrix} = \frac{\sqrt{2}}{2}\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \\ R(Z,45^\circ) &= \begin{bmatrix} \cos(45^\circ) & -\sin(45^\circ) \\ \sin(45^\circ) & \cos(45^\circ) \end{bmatrix} = \frac{\sqrt{2}}{2}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \end{aligned} \tag{7.4}$$

Movements in Directions #2 (-45°) and #6 (-135°) are therefore rotated to align with Directions #3 (-90°) and #7 ($+90^\circ$), so that, using (7.3) and (7.4), the viscosity matrix becomes:

$$B = B_i R(Z,45^\circ)\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} R(Z,45^\circ) \tag{7.5}$$

$$B = \frac{B_i}{2}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \tag{7.6}$$

$$B = B_i \begin{bmatrix} -0.5 & -0.5 \\ 0.5 & 0.5 \end{bmatrix} N\sec/m\,, \quad \textit{for Directions \#2 and \#6} \qquad (7.7)$$

Similarly, by aligning Directions #4 (-135°) and #8 ($+45^\circ$) with Directions #1 (-0°) and #5 (180°), one gets the viscosity matrix:

$$B = B_i \begin{bmatrix} 0.5 & -0.5 \\ 0.5 & -0.5 \end{bmatrix} N\sec/m\,, \quad \textit{for Directions \#4 and \#8} \qquad (7.8)$$

The viscosity gain, B_i, is set for each run based on the experiment protocol described in the next section.

7.2 Group #1

The first experiment was intended to study adaptation to endpoint-viscous force fields (dependent on the end-effector velocity) in a manner similar to that presented in [ShMu94]. This work was intended to verify that the designed system could reasonably reproduce the results of past motor learning experiments. Five healthy test subjects with no known neuromotor impairments
were used in this study.

7.2.1 Experimental Procedure

The test subjects were presented with five target sets, each consisting of 180 trials, with 30 trials in each direction. In between each 180 trial set, the subjects were instructed to rest for several minutes to prevent fatigue. The directions of the odd numbered trials were selected randomly from Directions #2 (-45°), #3 (-90°) and #4 (-135°). As mentioned previously, the even trials were then selected to return to the starting location (Directions #6 ($+135^\circ$), #7 ($+90^\circ$) and #8 ($+45^\circ$) respectively). The first target set was null field training to familiarize the subjects with the dynamics of the robot linkage. The force field in this case had $B_i = 0$ for all of the trials. The second set of targets was used to determine the force field baseline behaviour. A force with $B_i = 12N\sec/m$ was applied to five randomly chosen trials in each direction, with all other trials remaining in the null field. These trials provided a measure of the subject's performance in the force field without any opportunity for adaptation. The third and fourth target sets were the adaptation stage, in which all 180 trials had a force field with $B_i = 12N\sec/m$ applied. These trials allowed the subjects to experience the force field consistently and provided the opportunity for adaptation. The final target set had the force field applied for all but five trials in each direction. These randomly selected trials were catch trials intended to measure the after-effects of the subject's adaptation.

7.3 Group #2

The second experiment was designed to study learning generalization based on movement direction as demonstrated in [DoSh02] and [DoFS03]. The experimental setup, target directions and force field design for this experiment are all the same as for the first experiment. Also, as in the previous experiment, there were five other healthy test subjects with no known neuromotor impairments. No subjects participated in more than one experiment group.

7.3.1 Experimental Procedure

In this experiment, the subjects were presented with four sets of targets, again consisting of 180 trials each, with 30 trials in each direction. The first target set was the same as the null field set presented at the start of the first experiment. The second and third target sets represented the adaptation phase of the experiment and had the force field ($B_i = 12N\sec/m$) applied for all of the trials. However, in these two sets, the movements were restricted to Directions #3 (-90°) and #7 ($+90^\circ$). Finally, the fourth target set had 180 trials, with 30 in each direction. For this set, the force field was applied for all of the trials.

To determine whether the subjects in this experiment had generalized their adaptation from Directions #3 (-90°) and #7 ($+90^\circ$) to the other four directions, their performance in the final target set was compared to the performance of the Group #1 subjects in their first adaptation set. Since this was the first time the subjects from the first group experienced the consistent force field, they could be treated as naïve control subjects for the second group's experiment.

7.4 Group #3

The third and final experiment involved studying adaptation in a force field with a random magnitude in a manner similar to that in [ScDM01], where subjects performed reaching movements in a single directions while experiencing a random magnitude force field. Again, the experimental setup, target directions and force field design are all the same as those used for the previous two experiments. Also, as before, there were five test subjects with no known neuromotor impairments.

7.4.1 Experimental Procedure

Each of the subjects in this experiment experienced five sets of trials. The first set was the same null field target set from the first two groups, which allowed the subjects to learn the robot dynamics in all six directions. The next three sets were the adaptation sets, with each set consisting of 180 movements with varying force field magnitudes. These trials consisted of 30 movements in each of the six possible directions. During each trial, the gain value was set to a value selected randomly from the set:

$$B_i = \{8, 10, 12, 14, 16\}\, N\sec/m \tag{7.9}$$

It is worth noting that the mean value of the gain values from (7.9) is $12N\sec/m$, which is the force field magnitude from the first experiment. The final set of trials was the same as the previous three except that, for five trials in each direction, the gain magnitude was set to the null field value. This provided a measure of the after-effects of any adaptation to the random field, which could then be compared with the after-effect results from Group #1. Table 7.1 summarizes the experimental procedures for all three groups.

Table 7.1: Experimental procedure summary.

	Null	**Basline**	**Training**	**Testing**
Group #1	-No force field -180 trials -Directions 2, 3, 4, 6, 7, 8	-No force field -180 trials -Directions 2, 3, 4, 6, 7, 8 -Constant field for 5 random trials in each direction	-Constant force field -360 trials -Directions 2, 3, 4, 6, 7, 8	-Constant force field -180 trials -Directions 2, 3, 4, 6, 7, 8 -No field for 5 random trials in each direction
Group #2	-Same as above	-None	-Constant force field -360 trials -Directions 3 and 7 only	-Constant force field -180 trials -Directions 2, 3, 4, 6, 7, 8
Group #3	-Same as above	-None	-Random magnitude force field -540 trials -Directions 2, 3, 4, 6, 7, 8	-Random magnitude force field -180 trials -No field for 5 random trials in each direction

CHAPTER VIII

EXPERIMENTAL RESULTS AND DISCUSSION

8.1 Data Analysis

8.1.1 Data Conversion

The raw data files logged by the experiment software are not in a state that is appropriate for the analysis and display programs. Therefore the first stage of analyzing the data from the above experiments ~~is to~~ transform the raw data files into a suitable format. This transformation occurs in two stages, the first of which is to remove all of the non-trial run points, since these are the only points that need to be analyzed. These are the points in the data file for which *clockCount* is zero. Second, the position and velocity data that is logged by the experiment software is expressed in terms of the robot's base reference frame. The directions of the X- and Y-axes in this frame do not match the X- (horizontal) and Y- (vertical) axes of the visual interface in Figure 5.5. Therefore, each recorded data point is transformed to the display coordinate frame. This transformation is performed in the *transformFile.m* Matlab function in Appendix G.

8.1.2 Trajectory Trimming

It was found that, due to the backlash problems discussed in the Chapter 6, subjects had some difficulty bringing the robot handle to a smooth and immediate stop. In order to avoid any potential movement error miscalculations resulting from this issue, each movement trajectory was trimmed to eliminate any data points after the first transient minimum in the trial's velocity profile [ScDM01]. This transient minimum corresponds to the point at which the trajectory changes direction as a result of the subject's corrective actions. An example trajectory, along with its associated velocity profile, is shown in Figure 8.1. Note that this velocity profile represents the magnitude of the velocity. The transient minimum in the velocity profile is shown with a vertical line and the corresponding point on the trajectory is highlighted with an arrow. The Matlab program to perform this operation, *trimTrajectories.m*, is presented in Appendix H.

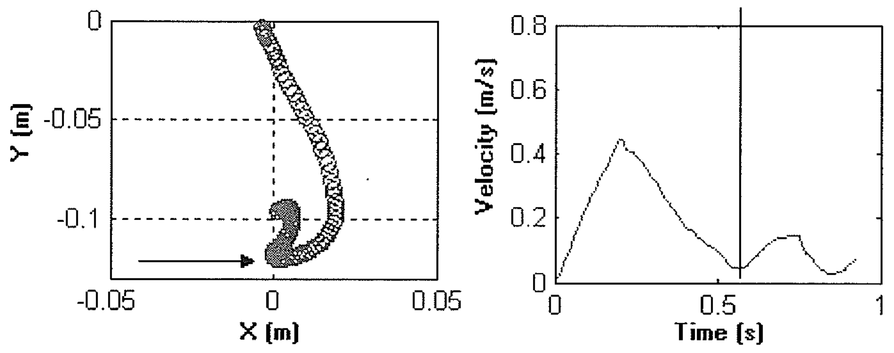


Figure 8.1: Typical movement trajectory (left) and velocity profile (right).

8.1.3 Movement Error

In order to quantify the performance of the test subjects, a measure of movement error was developed. For each trial the movement error was defined as the maximum perpendicular displacement from the straight line joining the starting and target locations. Note that this assumes, as discussed in the Motor Learning Background chapter, that the subject's desired trajectory is along such a straight line [ShMu94]. Upon examination of the trajectory plots in subsequent sections, one can see that the error caused by the perpendicular forces creates an error towards the end of the trimmed trajectories. Because of this, the movement error was calculated using the final 100 ms of each trimmed trajectory. The positive error direction was chosen to coincide with the direction of the perturbing force field. The Matlab functions used to calculate this movement error, *findMovementErrors.m* and *findDisp.m*, can be found in Appendix I.

8.1.4 Measure of Significance

In order to quantify the amount of generalization that occurs in between directions in Group #2, a measure of significance was employed. Specifically, Student's t-test, [MoRH94], was used to compare the average movement errors between the naïve control subjects from Group #1 and those in Group #2. In this test, the null hypothesis that the two sample means are the same is either accepted or rejected with a given level of confidence. In these experiments a 95% confidence value was selected. If the null hypothesis is rejected, then one can state with 95% confidence that the sample means are significantly different and that some sort of generalization must have occurred. The t-test itself was performed using the *ttest2.m* function from the Matlab Statistical Toolbox.

8.1.5 Correlation Analysis

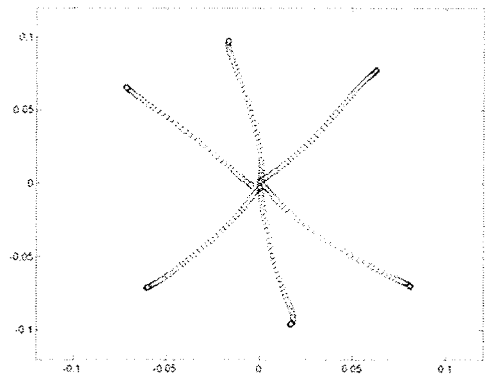
As presented in [ScDM01], the relationships between current and past movement errors and between current movement errors and past force field magnitudes can be determined using a correlation analysis. Specifically, the relationship between current and past errors can be found using the autocorrelation of the recorded error sequence. If the subjects are not simply adapting to some perceived constant field strength, but rather they are anticipating the force field strength based, in part, on past errors, there should be a significant correlation with past movement errors seen in the autocorrelation sequence.

The 95% confidence interval was used for a measure of significant correlation. The size of the interval was based on the $2\sigma/\sqrt{N}$ limits, where N is the length of the signals being correlated. Similarly, a cross-correlation between the movement errors and the force field magnitudes was used to determine whether a subject's performance on any given trial was dependent on the previously encountered forces. The Matlab function for performing the correlation analysis, *corrAnalysis.m*, can be found in Appendix J.

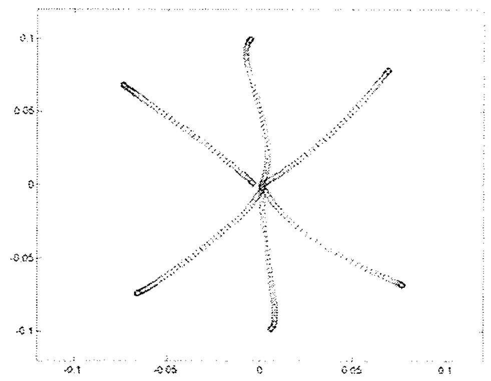
8.2 Group #1

8.2.1 Results

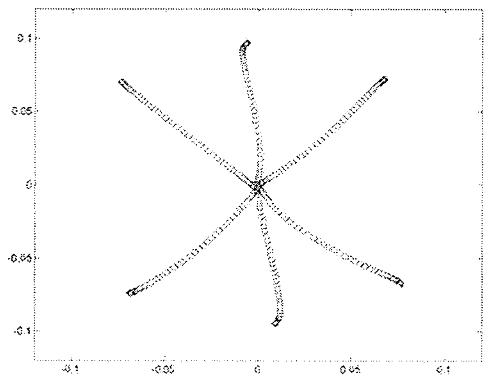
The results presented for this section will include a full collection of data from one subject so that the reader can gain a full appreciation of results for a typical subject. The results for the other four subjects will be included in a more condensed form. The adaptation of Subject #1 is illustrated in Figure 8.2. Each plot contains the average of 10 movements in each direction. The sequence of plots shows the subject's adaptation through their 360 adaptation trials. The average baseline force field performance for Subject #1, along with their average after-effect performance, is shown in Figure 8.3. Figure 8.3A shows the initial performance of the subject without an opportunity for adaptation. The null field results in Figure 8.3B represent the performance of the subject after adaptation, but without a force field present. The movement error data for each of the directions are shown in Figures 8.4-9. Each image contains the average movement error for each subject (10 move bins) and the average error for all subjects in a given direction. Finally, the average baseline error and average after-effect error values for all of the subjects are shown in Tables 8.1 and 8.2 respectively.



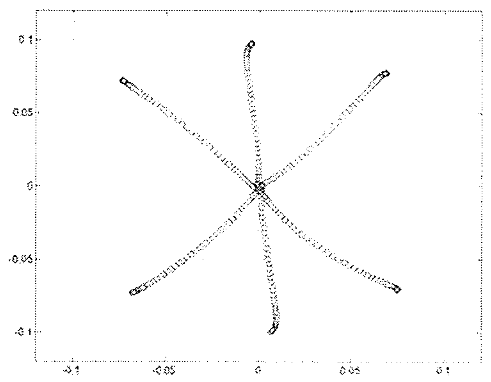
A



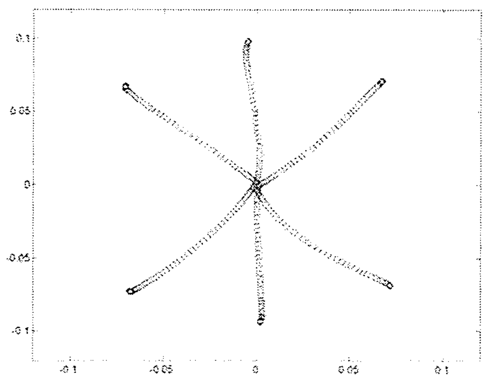
B



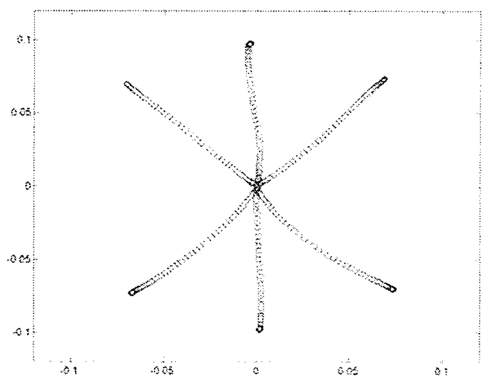
C



D

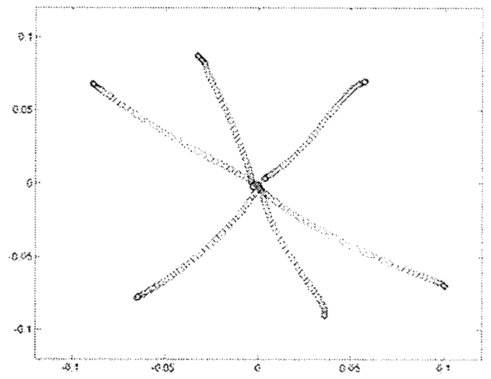


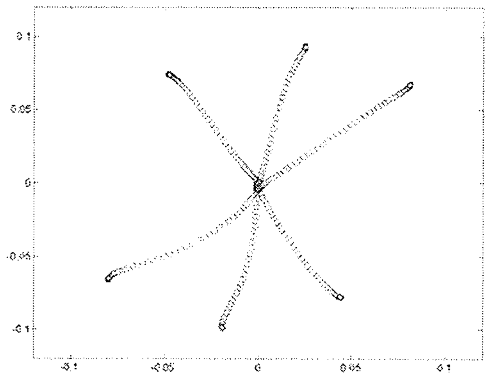
E



F

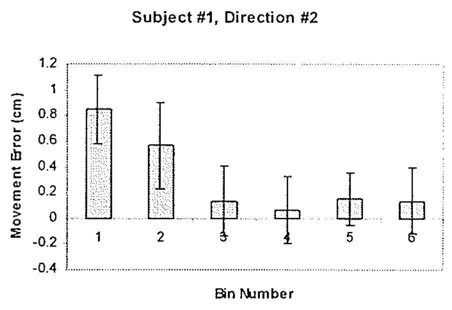
Figure 8.2: Subject #1 average trajectories during adaptation.
Axis units in metres, X-horizontal, Y-vertical.
A: First 60 moves (10 each direction), F: Final 60 moves.

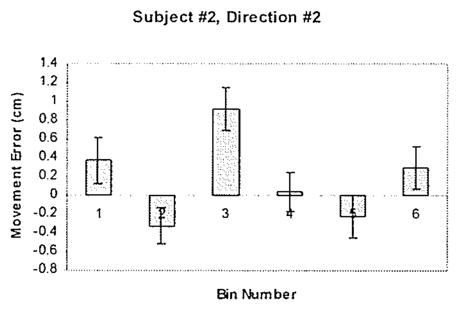




A B

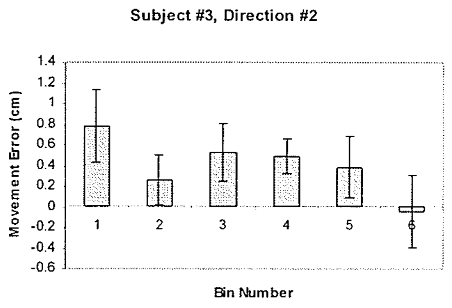
Figure 8.3: Subject #1 baseline and after-effect performance.
Axis units in metres, X-horizontal, Y-vertical.
A: Force field baseline, B: After-effects.

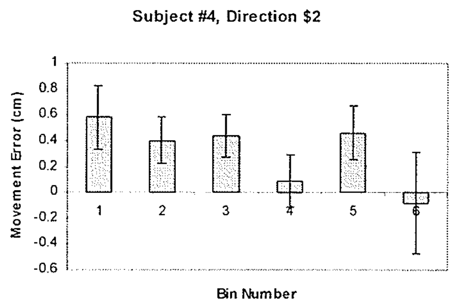




A

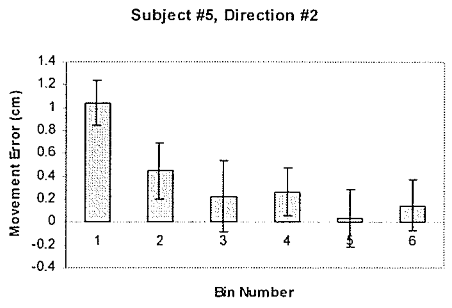
B

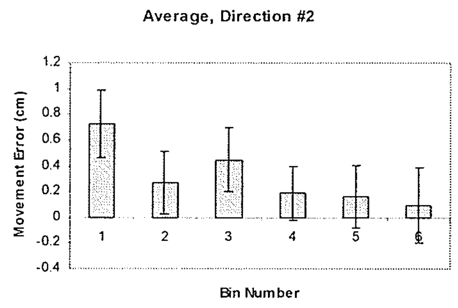




C

D

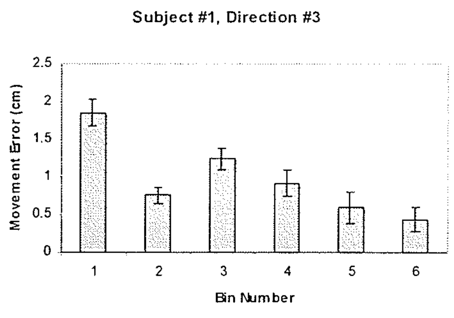




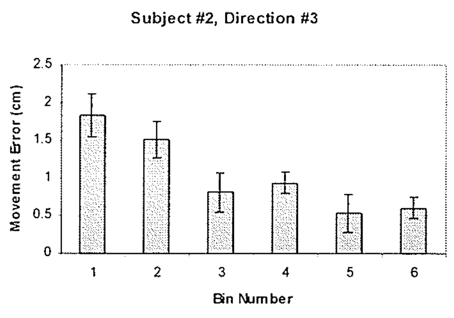
E

F

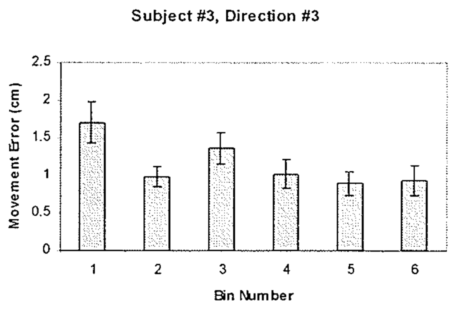
Figure 8.4: Direction #2 (– 45°) movement errors. Each bin represents the average for 10 trials, with bars showing standard deviation. A: Subject #1, B: Subject #2, C: Subject #3 D: Subject #4, E: Subject #5, F: Average between subjects



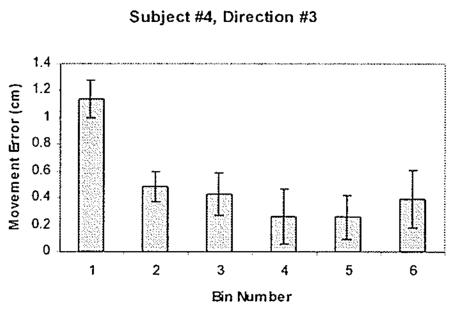
A



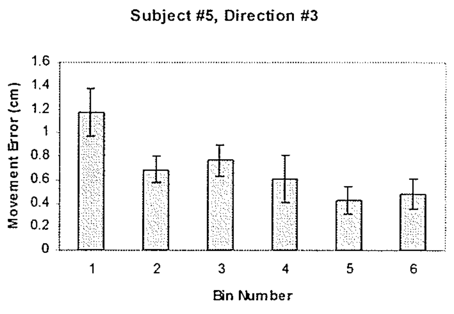
B



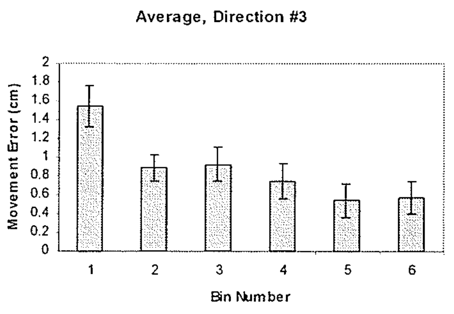
C



D



E



F

Figure 8.5: Direction #3 (-90°) movement errors. Each bin represents the average for 10 trials, with bars showing standard deviation. A: Subject #1, B: Subject #2, C: Subject #3 D: Subject #4, E: Subject #5, F: Average between subjects

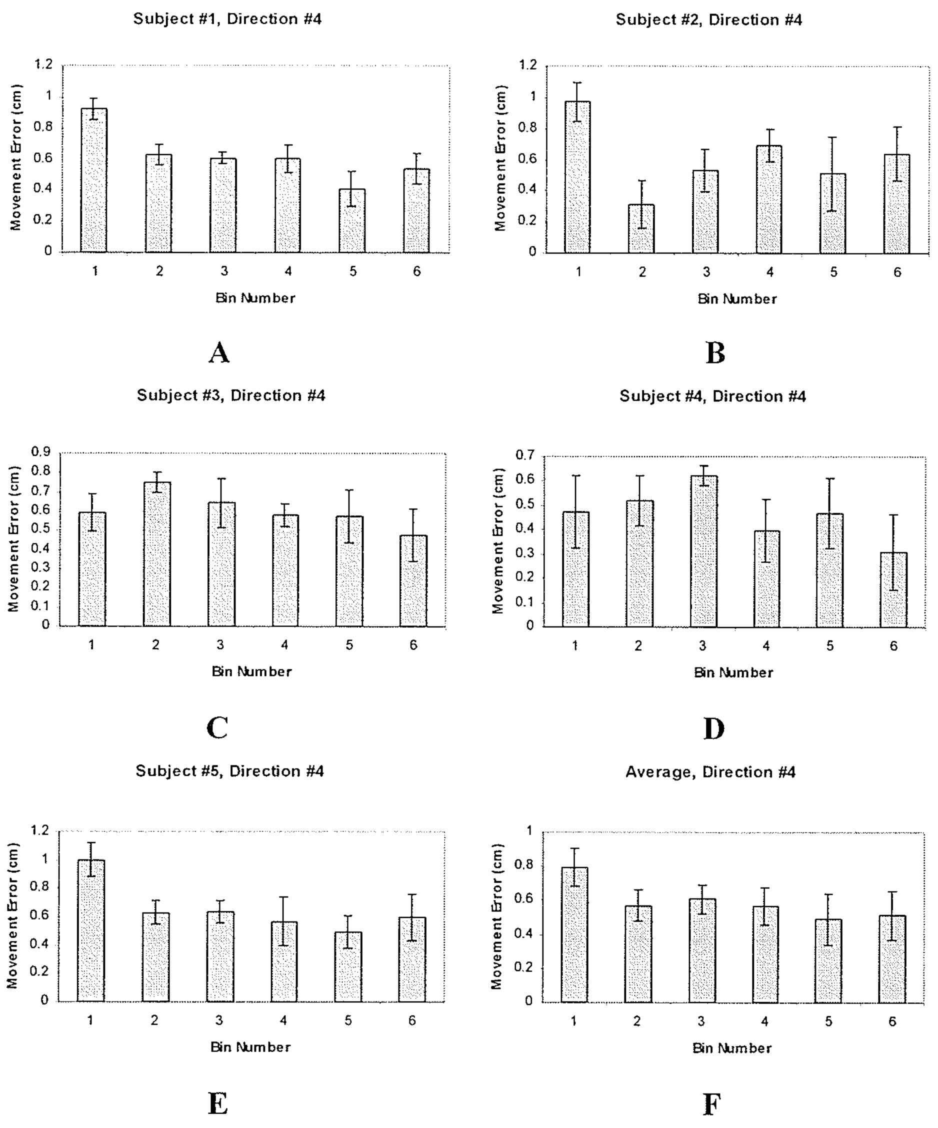


Figure 8.6: Direction #4 (-135°) movement errors. Each bin represents the average for 10 trials, with bars showing standard deviation. A: Subject #1, B: Subject #2, C: Subject #3 D: Subject #4, E: Subject #5, F: Average between subjects

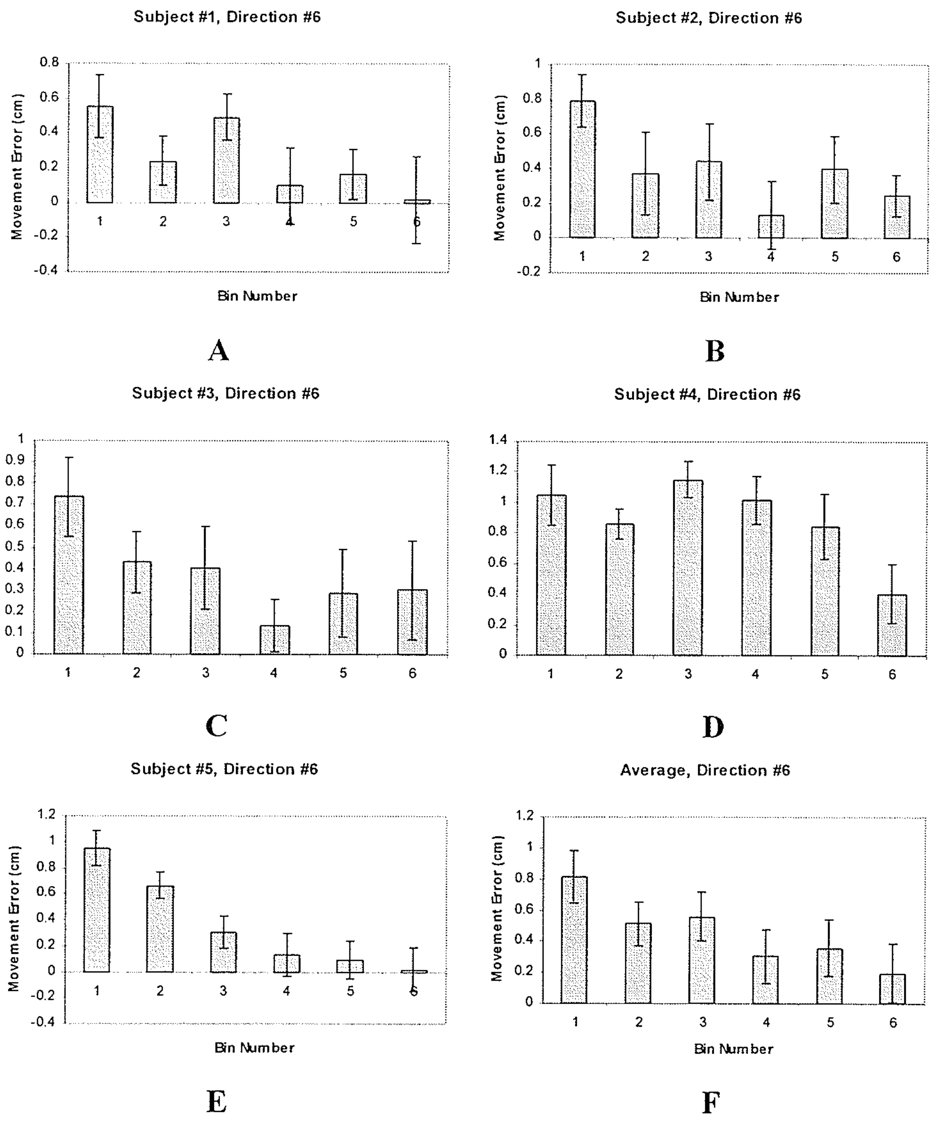
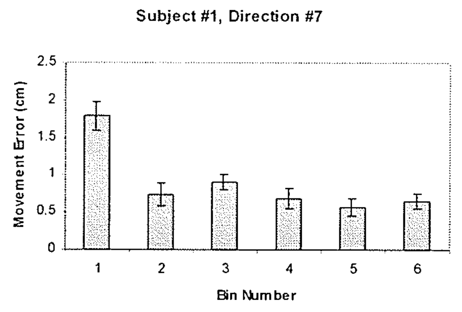
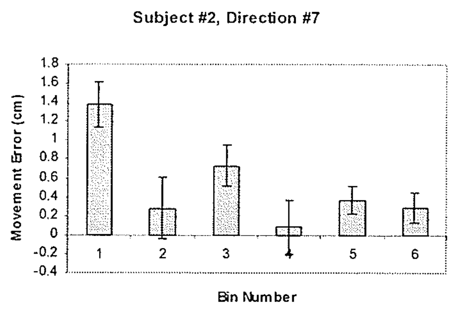


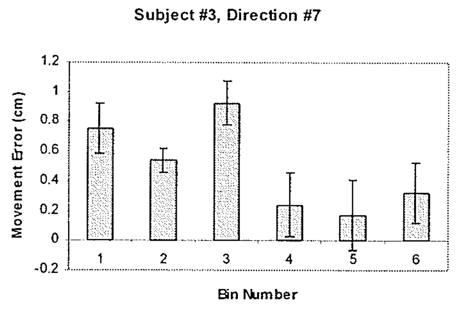
Figure 8.7: Direction #6 (+135°) movement errors. Each bin represents the average for 10 trials, with bars showing standard deviation. A: Subject #1, B: Subject #2, C: Subject #3 D: Subject #4, E: Subject #5, F: Average between subjects



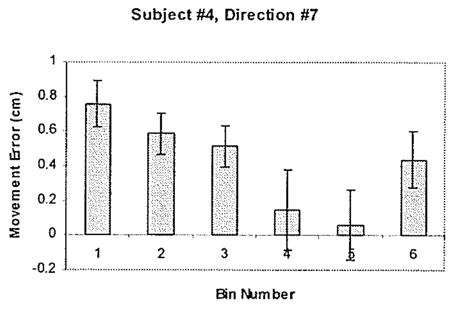
A



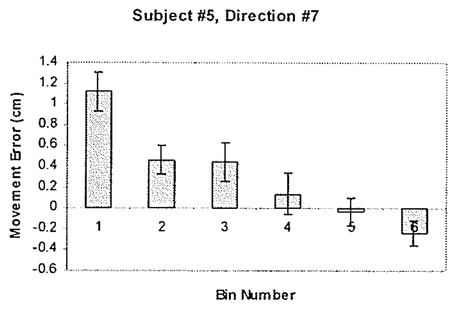
B



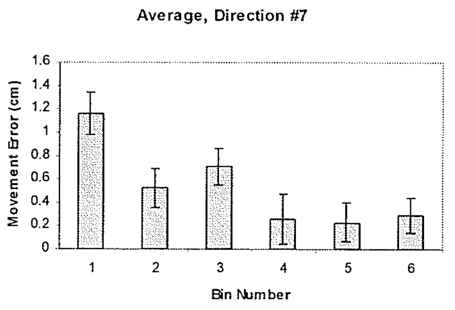
C



D



E



F

Figure 8.8: Direction #7 (+90°) movement errors. Each bin represents the average for 10 trials, with bars showing standard deviation. A: Subject #1, B: Subject #2, C: Subject #3 D: Subject #4, E: Subject #5, F: Average between subjects

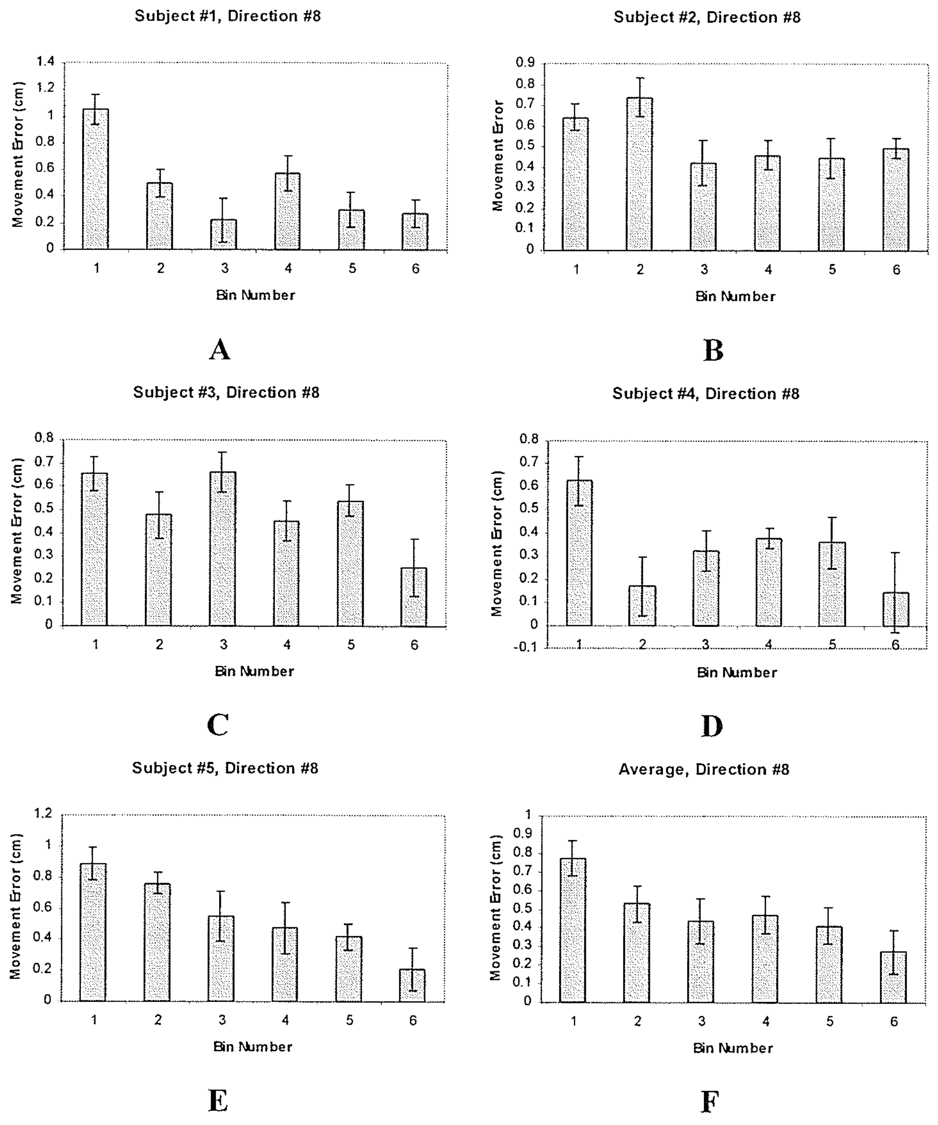


Figure 8.9: Direction #8 (+ 45°) movement errors. Each bin represents the average for 10 trials, with bars showing standard deviation. A: Subject #1, B: Subject #2, C: Subject #3 D: Subject #4, E: Subject #5, F: Average between subjects

Table 8.1: Average baseline errors (cm).

	Sub #1	Sub #2	Sub #3	Sub #4	Sub #5	Ave.	Std. Dev.
Direction #2	2.19	2.91	1.65	1.90	1.66	2.06	0.23
Direction #3	3.66	2.83	2.74	2.88	1.96	2.81	0.27
Direction #4	1.07	1.36	0.82	0.84	0.87	0.99	0.10
Direction #6	1.50	1.67	1.59	1.55	1.47	1.56	0.04
Direction #7	3.33	3.18	2.19	2.19	1.83	2.54	0.30
Direction #8	1.05	1.04	0.83	0.90	0.99	0.96	0.04

Table 8.2: Average after-effect errors (cm).

	Sub #1	Sub #2	Sub #3	Sub #4	Sub #5	Ave.	Std. Dev.
Direction #2	-2.48	-1.89	-2.24	-2.13	-1.65	-2.08	0.14
Direction #3	-1.94	-1.71	-1.09	-2.51	-1.67	-1.78	0.23
Direction #4	-1.03	-1.51	-0.49	-0.77	-0.88	-0.94	0.17
Direction #6	-1.90	-1.95	-1.34	-1.90	-1.50	-1.72	0.12
Direction #7	-2.51	-2.80	-1.95	-1.92	-2.00	-2.24	0.18
Direction #8	-1.14	-1.96	-1.14	-1.38	-0.95	-1.31	0.18

8.2.2 Discussion

Examination of Figure 8.2 shows that, as the adaptation trials progressed, the average performance of Subject #1 approached a near-straight path (for all directions). The change is particularly obvious in Directions #3 (-90°) and #7 ($+90^\circ$). This apparent improvement in performance should correspond to an associated decrease in movement error with adaptation. This is indeed the case for Subject #1, as illustrated in Figures 8.4-8.9 (Plot A). The movement errors for the other subjects also show a general decrease with adaptation as can be seen in Plot F in Figures 8.4-8.9 showing the average movement error for each direction. The average performance of Subject #1 suggests, the most drastic improvement occurs in Directions #3 (-90°) and #7 ($+90^\circ$), as seen in Figures 8.5 and 8.8 respectively. In these directions, all of the test subjects exhibit a consistent and significant improvement in movement error as they adapt to the force field. For Directions #2 (-45°) and #6 ($+135^\circ$), seen in Figures 8.4 and 8.7, an obvious decrease in movement error occurs for four out of five subjects. In Direction #2 (-45°), Subject #2 shows no clear change in his/her performance as a result of adaptation. Subject #4 shows an improvement in performance for Direction #6 ($+135^\circ$), although it is not as significant as the improvement for the other four subjects. Examination of Figure 8.9 shows that all of the test subjects show an improvement in movement error for Direction #8 ($+45^\circ$), although the decrease is less significant for Subject #2 and Subject #3. Also, the improvement in Direction 8 ($+45^\circ$) is less significant than those of Directions #2 (-45°), #3 (-90°), #6 ($+135^\circ$) and #7 ($+90^\circ$). In the case of Direction #4 (-135°), Figure 8.6, there is a small improvement visible. This improvement is, however, not particularly significant as expected. Since Direction #8 ($+45^\circ$) is directly opposite to Direction #4 (-135°), it is understandable that if one of these directions shows less significant improvement, it should be reflected on the opposite direction as well.

Comparing the results for Direction #4 (-135°) with those of the other directions, one can see that the subjects generally had a poorer performance in this direction. This lack of improvement appears to be the result of the performance limitations discussed in Chapter 6, since the path followed by the subjects is curved. The friction and backlash issues result in the upper-arm link of the manipulator remaining relatively motionless during the movement, while the forearm link rotates about the elbow joint.

The general improvement in performance exhibited by the test subjects suggests that they were adapting to the applied force field. In order to verify the use of an internal model in the adaptation process, the after-effects of adaptation were compared with the force field baseline performance. The average baseline and after-effect trajectories of Subject #1 are shown in Figure 8.3A and 8.3B respectively. It can be seen that, when the force field was randomly removed during the after-effect trials, the subjects moved along a trajectory that mirrored (approximately) their initial performance in the force field. The resulting average movement errors are presented in Tables 8.1 (baseline performance) and 8.2 (after-effects). It can be seen that the after-effect errors are opposite in sign and of the same approximate amplitude as the force field baseline errors. This suggests that the subjects were predicting what forces they would encounter on the next trial and were generating joint torques to compensate for the expected force. This provides evidence that there is an internal model being used in the adaptation process as expected.

Overall, this experiment showed that, for reaching movements performed in a more natural, non-abducted arm orientation, subjects still exhibited clear evidence of adaptation through the use of an internal model. These results are in line with those reported in previous work such as [ShMu94]. Also, it was seen that, even though the manipulandum used in this work was lower in cost and quality than other similar systems, it could still be used to determine the general adaptation strategies employed by the test subjects.

8.3 Group #2

8.3.1 Results

The primary objective for the second experiment was to view the performance of test subjects in the same field used during the first adaptation set of Group #1, but after training in a two direction training set. With this is mind, the average performance of the Group #2 subjects for each of the six directions is shown in Figure 8.10. These averages are based on 10 move bins for each direction.

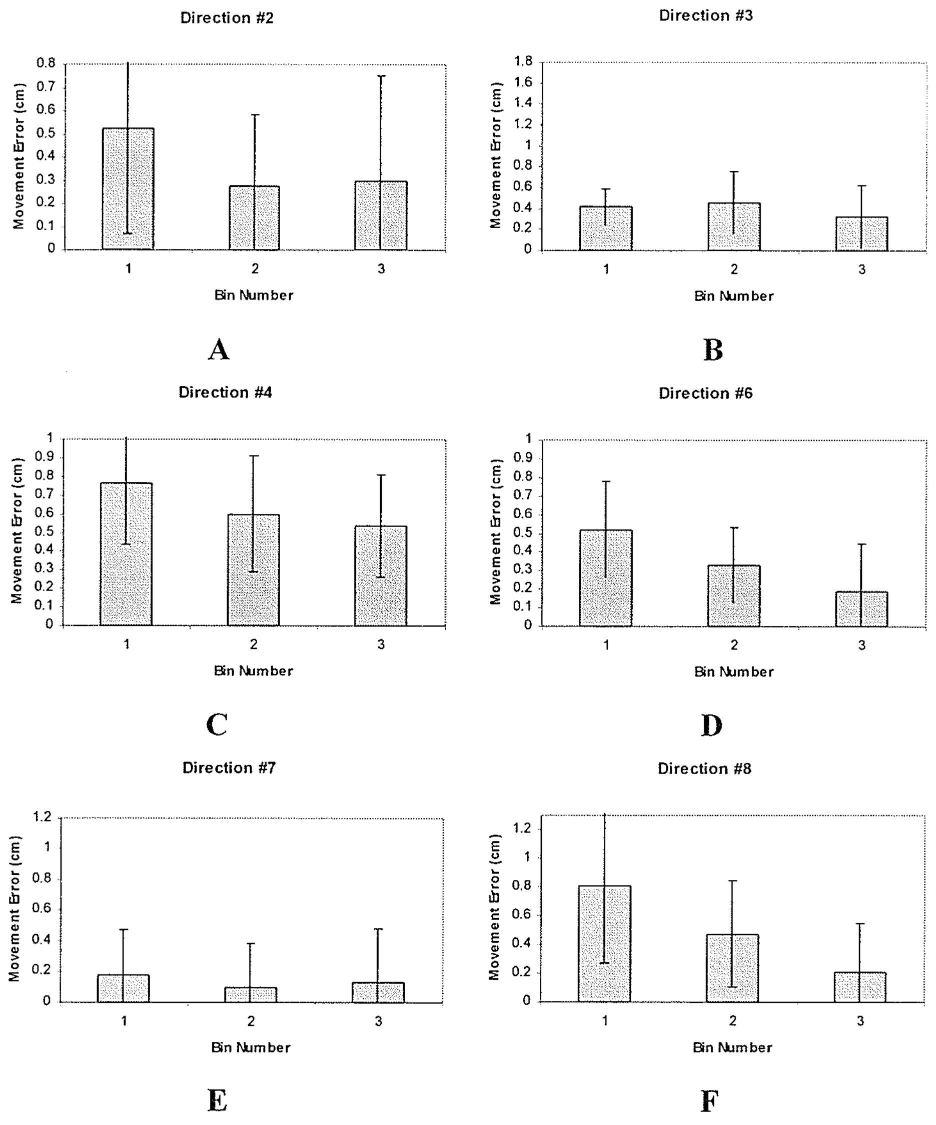


Figure 8.10: Average performance for Group #2 after training in Directions #3 and #7. Each bin represents the average for 10 trials, with bars showing standard error. A: Direction #2 (-45°), B: Direction #3(-90°), C: Direction #4 (-135°), D: Direction #6 (-135°), E: Direction #7 ($+90^\circ$), F: Direction #8 ($+45^\circ$).

8.3.2 Discussion

The average performance of the Group #2 subjects in all six directions after training in Directions #3 (-90°) and #7 ($+90^\circ$) is seen in Figure 8.10. In order to determine if there was any generalization of learning based on direction, these results were compared with the naïve control subjects from Group #1. The initial force field performance for each direction shown in Plot F, Bin #1 of Figures 8.4-8.9 was compared with the corresponding Bin #1 performance in Figure 8.10.

For each direction, a Student's t-test was performed to find the significance of any change in movement error. The two sampled data sets being compared are the average performances of the subjects in each experiment (Bin #1). Table 8.3 shows the probability that the null-hypothesis should be accepted (performance differences are due to chance) and also shows whether these probabilities are significant with either an 85% or 95% confidence interval.

Table 8.3: Direction generalization significance values between Group #1 control subjects and Group #2 test subjects. Based on performance of subjects during initial exposure to the consistent force field in all directions (average error during first bin).

	Dir 2	**Dir 3**	**Dir 4**	**Dir 6**	**Dir 7**	**Dir 8**
P-value	0.15	0.0004	0.885	0.0628	0.0054	0.8404
Significant? (95%)	N	Y	N	N	Y	N
Significant? (85%)	Y	Y	N	Y	Y	N

Since Directions #4 (-135°) and #8 ($+45^\circ$) showed less significant improvement in performance due to adaptation in Group #1, it was expected that there would be less chance of any learning generalization to these directions. This expectation was supported by the probability values in Table 8.3, since the probability of accepting the null hypothesis was above 80% for each of these directions. Since Directions #3 (-90°) and

#7 ($+90^\circ$) were the training directions, the very low probability values in the above table were expected. Obviously, training in these directions gave the Group #2 subjects a significantly lower movement error than the untrained Group #1 subjects.

For Directions #2 (-45°) and #6 ($+135^\circ$), it can be seen that the null-hypothesis cannot be rejected with 95% confidence. Instead, the confidence interval must be lowered to 85% for rejection. This suggests that there may have been generalization to Directions 2 (-45°) and 6 ($+135^\circ$), as predicted by [DoFS03], but it was not at the expected significance level. This reduced significance could be the result of the extra uncertainty added by the friction and backlash from the gearheads. Also, since only 4 out of 5 subjects showed clear adaptation in these directions during Group #1, one could expect to see a slightly lower chance of learning and generalization in these directions. Overall, these results suggest that there is generalization from Directions #3 (-90°) and #7 ($+90^\circ$) to Directions #2 (-45°) and #6 ($+135^\circ$), which supports the idea of basis functions with large receptive fields reported in the past [DoFS03] [Muss99].

8.4 Group #3

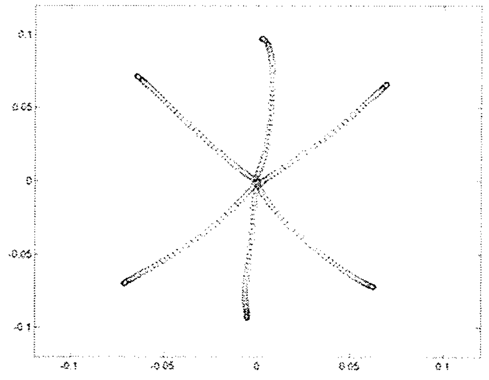
8.4.1 Results

The primary objective of the Group #3 experiments was to see how subjects adapted to an uncertain force field. The first question to consider is to what approximate field did the subjects adapt? To quantify this in some way, the average after-effect errors were determined. The resulting values are displayed in Table 8.4. Next, the average trajectories for each of the possible gain magnitudes are shown for a typical subject in Figure 8.11. These plots show the average movement for all of the trials in each direction at each gain level. Figures 8.12 and 8.13 show the overall error series and direction-based error series respectively for a typical subject.

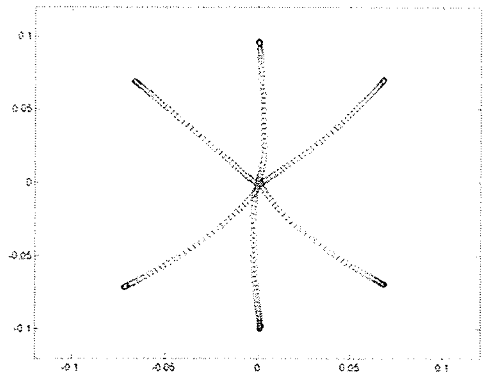
Table 8.4: Average after-effect errors in random field (cm).

	Sub #1	Sub #2	Sub #3	Sub #4	Sub #5	Ave.	Std. Dev.
Direction #2	-2.13	-2.39	-2.74	-2.57	-2.17	-2.40	0.12
Direction #3	-2.80	-1.69	-1.45	-2.31	-1.36	-1.92	0.28
Direction #4	-1.42	-0.33	0.01	0.27	-1.02	-0.50	0.32
Direction #6	-1.43	-1.27	-1.90	-1.76	-1.59	-1.59	0.11
Direction #7	-2.00	-2.07	-1.41	-2.34	-3.29	-2.22	0.31
Direction #8	-1.51	-0.38	-0.74	-1.76	-1.44	-1.17	0.26

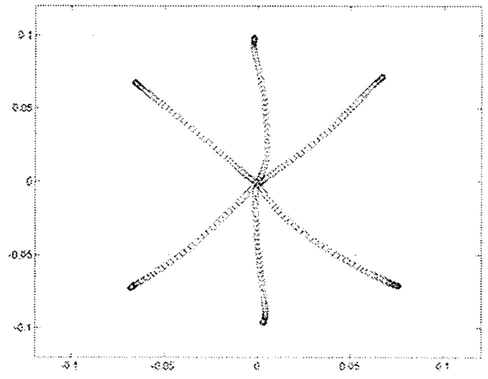
. The results of the overall error/error autocorrelation and error/gain cross-correlation analyses are presented in Figure 8.14 for all subjects. The results for the individual directions are shown in Figures 8.15-19. Each figure from Figure 8.15-19 contains correlations for each direction for each subject. The top plot for each pair is the error autocorrelation, while the bottom plot is the gain/error correlation. The dashed horizontal lines in each plot represent the 95% confidence interval limits.



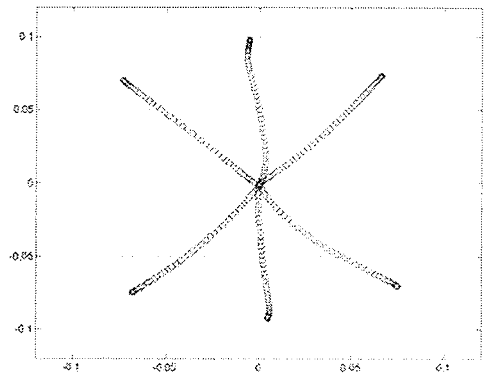
A



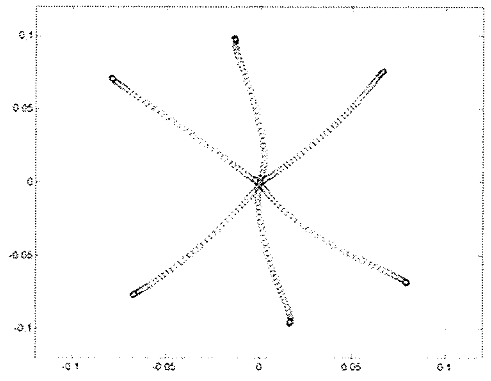
B



C



D



E

Figure 8.11: Subject #1 of Group #3 average performance in different field strengths. A: 8 Nsec/m, B: 10 Nsec/m, C: 12 Nsec/m D: 14 Nsec/m, E: 16 Nsec/m

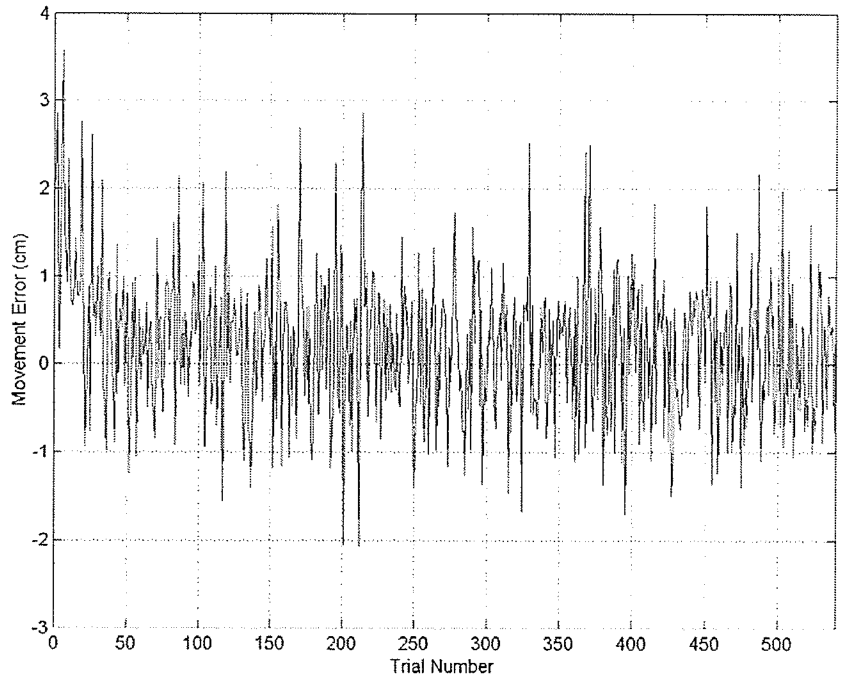


Figure 8.12: Subject #1 of Group #3 movement errors over all directions.

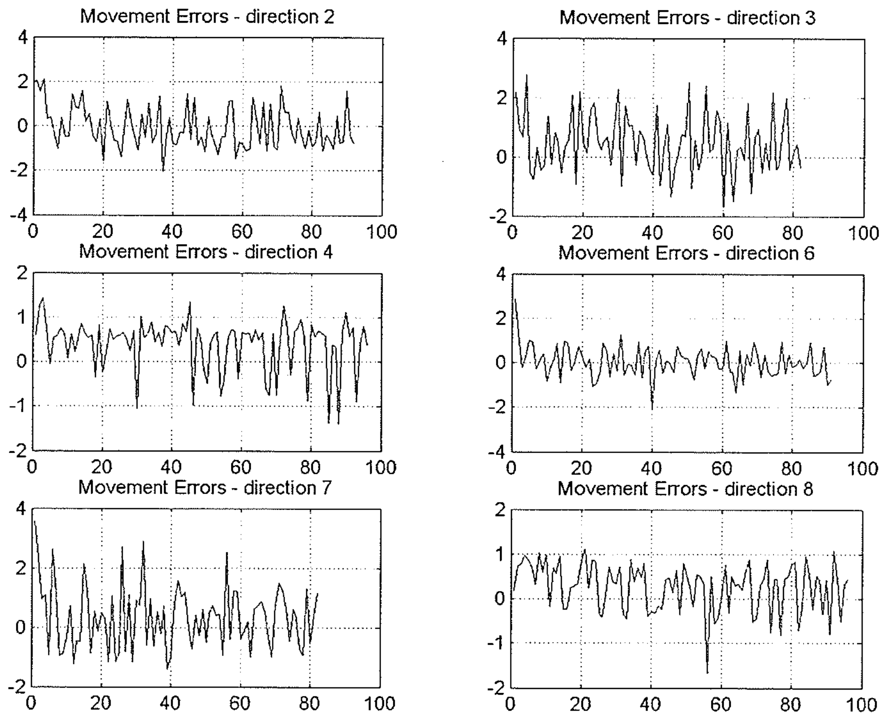
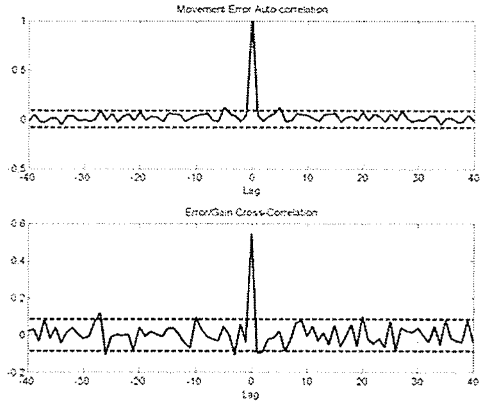
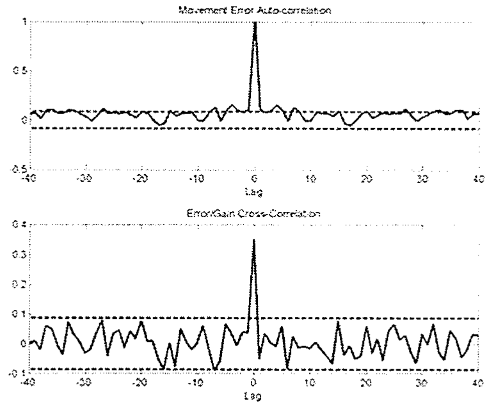


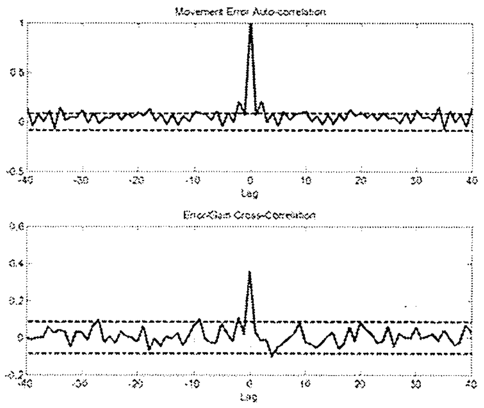
Figure 8.13: Subject #1 of Group #3 movement errors in each direction.



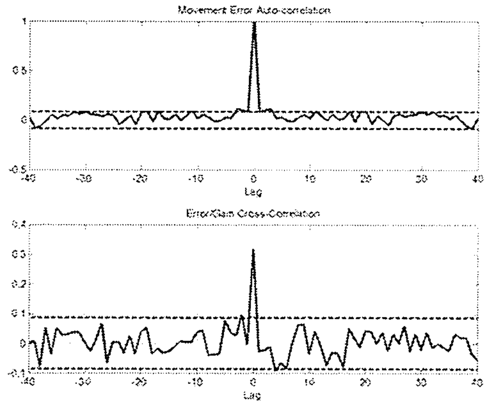
A



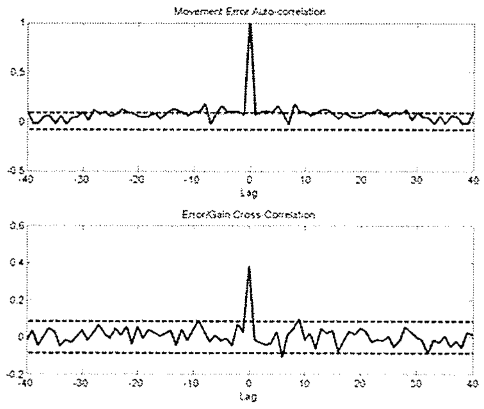
B



C

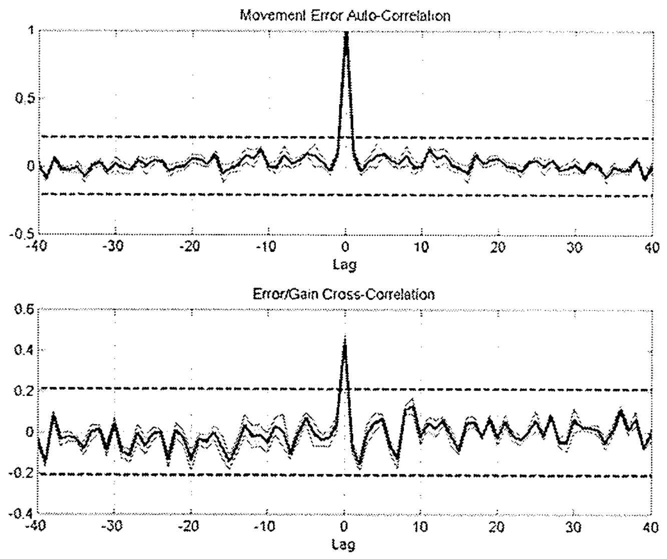


D

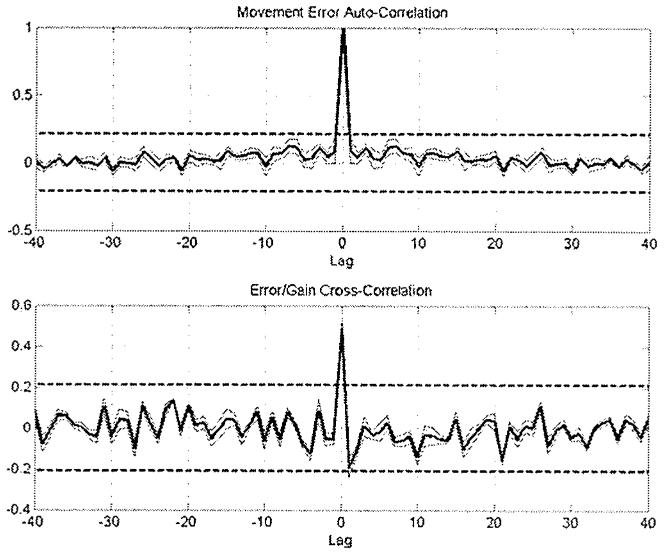


E

Figure 8.14: Correlation analysis over all directions for each subject. Top Image – Error auto-correlation, Bottom Image – Error/Gain Cross-correlation, A: Subject #1, B: Subject #2, C: Subject #3 D: Subject #4, E: Subject #5

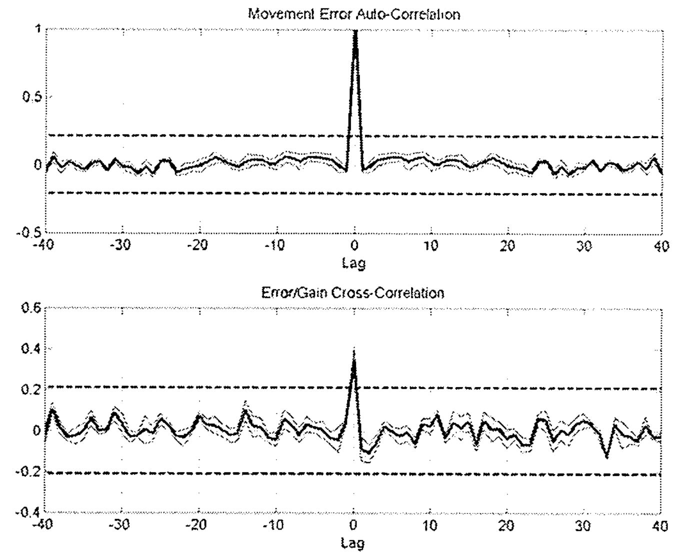


A

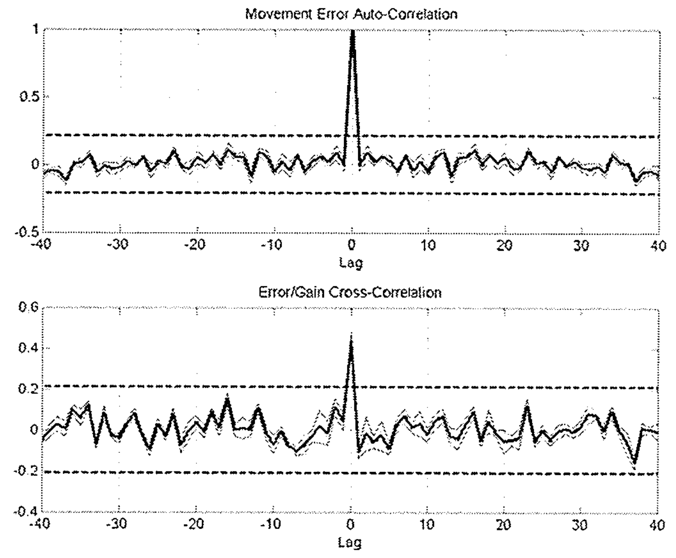


B

Figure 8.15: Correlation analysis for Directions #2 and #3. Averaged between all subjects, Gray - standard deviation. A: Direction 2 (-45°), B: Direction 3 (-90°)

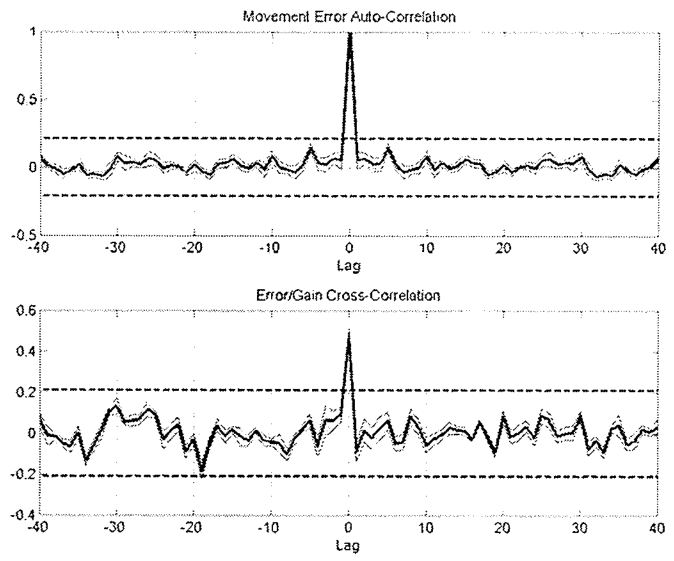


A

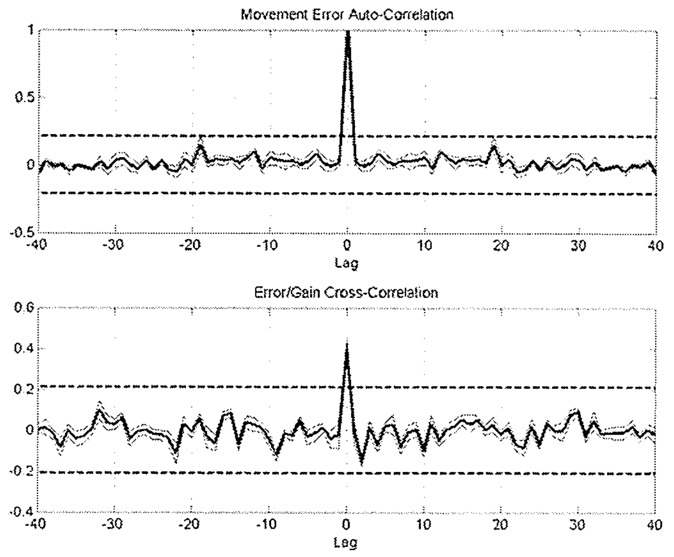


B

Figure 8.16: Correlation analysis for directions #4 and #6. Averaged between all subjects, Gray - standard deviation. A: Direction 4 (-135°), B: Direction 6 ($+135^\circ$)



A



B

Figure 8.17: Correlation analysis for directions #7 and #8. Averaged between all subjects, Gray - standard deviation. A: Direction 7 (+90°), B: Direction 8 (+45°)

8.4.2 Discussion

Past work, [ScDM01], suggested that when exposed to a force field with uncertain magnitude values, subjects adapted to the approximate mean of the force field. To determine if this was the case in the present work, the after-effect values from Group #3 (Table 8.4) were compared with the after-effects from Group #1 (Table 8.2). This was done because the mean of the force field for this experiment was selected to be equal to the force field magnitude from Group #1. Comparison of the values in these two tables shows that the average after-effect values for each direction from Group #3 are of the same sign and approximate magnitude as those from Group #1. This suggests that the adaptation to the random force field was to the approximate mean force field. This is also supported by examining Figure 8.10 and noting that the force field gain with the least movement error is 10-12 Nsec/m. These results show that the adaptation to the approximate mean of the random field applies even when considering multiple directions, rather than just the single direction reported in [ScDM01].

In a similar manner to [ScDM01], examination of a typical movement error series, as shown in Figure 8.12, reveals a sharp decrease in movement error with the first 100 trials before stabilizing around some steady-state mean value. By fitting an exponential curve to this data one can estimate the 5τ time to steady state of approximately 70 trials, which gives a time constant of 14 trials. This time constant was consistent across all five subjects in this experiment. Interestingly, this time constant is six times larger than the 2.4 trial time constant reported for a Gaussian random gain in [ScDM01]. To explain this, consider the corresponding direction-based error plots (Figure 8.13), in which one can see that each direction's error series has an initial decrease in error with a time constant in the range of 2-3 trials. These results show that when multiple directions are considered, each individual direction adapts at the same rate as it would if there were only one direction, and that the overall adaptation rate (over all directions) is simply the sum of the time constants for each direction. This suggests that, even though there might be some generalization between directions as seen in Group #2, each direction in the random force field appears to adapt primarily based on the errors and gains in that direction alone.

With this result in mind, it remains to determine the relationship between current movement error and past movement errors and field gains. As seen in Figure 2.1, one would expect the current movement error to depend on the most recent movement errors and force field gains. According to the time constant analysis above, there should be significant correlations (similar to Figure 2.1) in each individual direction, but not for the overall error series. The correlations for the overall error series in Figure 8.14 show only slightly significant error/gain correlations at a lag of around 4-6 trials. These correlations are beyond the 1-2 lag results reported in [ScDM01] and may simply be a manifestation of the correlations for individual directions discussed in the next paragraph. The error auto-correlation sequences in Figure 8.14 show no clear trends and do not display the correlation pattern found in [ScDM01]. There are a few positive correlation points that may be the result of direction-based correlations due to the performance limitations of the manipulator (Chapter 6).

The average error auto-correlation results for the individual directions (Figures 8.15-8.17 – Upper images) do not show any significant correlations. It is possible that any error-based dependence is either offset by the positive correlations from the manipulator dynamics or affected by the additional uncertainty added to the system by the gearhead backlash.

The error/gain correlation analysis for each direction supports the expected gain dependence of the movement error. The negative correlation with the most recently experienced gains (1-2 trials) can be seen in all directions shown in Figures 8.15-17. While these negative correlations are not outside the 95% confidence interval limits, they are significant when compared to the correlations for other lag values. The less significant correlation is most likely the result of having subsequent trials in one direction separated by at least one trial in another direction. Depending on the number of intermediate trials, the subject will 'unlearn' the results for the most recent trial in a given direction and the resulting correlation will be reduced accordingly. This would apply to both the error autocorrelation and error-gain cross-correlation results. However, despite this unlearning, adaptation still occurs for each direction and, based on the results in Figures 8.15-8.17, it is most dependent on the previous gains experienced in that direction.

It is also interesting to note that the lowest zero-lag error/gain correlation values occur in Directions #4 and #8. This result supports the conclusion that, for these particular directions, the manipulator dynamics were significantly affecting the error values. This is evident because the error was not as dependent on past or current errors or gains and must have been the result of the manipulator performance limitations.

Overall, the results of this experiment show that in the presence of an unpredictable force field, subjects adapt to the approximate mean of the field gain and do so based on recently experienced force values. The recent experiences that most affect the performance in a given direction are the gains previously encountered in that particular direction. Also, it appears that the performance limitations of the manipulandum contribute to the inconsistencies in the data analysis for this particular experiment, whereas they did not significantly affect the results for Group #1. Based on this result, it is reasonable to state that, as one would expect, the lower cost manipulator option allows for a reasonable determination of general learning trends while limiting some finer detailed analysis.

8.5 Parkinson Disease Case Study

8.5.1 Motivation

With the experimental apparatus verified and the experimental results discussed, a practical application of the work at hand was started. In particular, a preliminary pilot study of the effects of Parkinson disease (PD) on motor learning was performed. While studies have been performed to study the effects of impairments such as Huntington's disease on motor learning [SmBS00], no similar studies have focused on Parkinson disease. Since the work in [SmBS00] showed that reaching task experiments involving viscous force fields may be used as a mechanism for early detection of Huntington's Disease, there is a motivation to determine whether similar claims can be made regarding PD. Also, once the motor learning aspects of PD are understood, there is the potential to develop motor learning-based rehabilitation exercises which patients can use to improve their motor control capabilities.

8.5.2 Background

First described in 1817, by Dr. James Parkinson, Parkinson disease is a brain disorder characterized by four main symptoms:

- Tremor of a limb at rest
- Slowness of movement
- Stiffness of limbs
- Poor balance

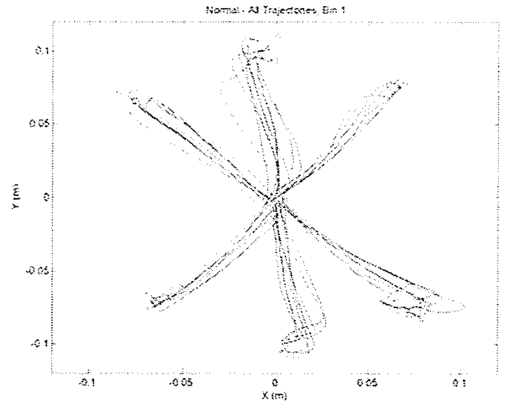
Parkinson disease occurs when dopamine-producing cells in the brain are damaged or die. Dopamine is a vital chemical, which allows smooth and coordinated functioning of the body's muscles [PaLK03]. There are approximately 1.5 million Americans with PD and an estimated 60,000 newly diagnosed cases each year. Diagnosis of PD is difficult since there are no current blood tests or imaging scans (X-ray, MRI) that can confirm PD. These tests can, however, be used to rule out other possible diagnoses. From a treatment perspective, there are many medications available to alleviate the symptoms of PD. These treatments mainly focus on the replacement or imitation of dopamine. There is currently no cure for PD, so these treatment options are intended to improve the quality of life for the patient by treating their most bothersome symptoms. Both the difficulty in diagnosis and lack of a cure are major motivating factors in studying the motor control aspects of PD.

8.5.3 Experimental Procedure

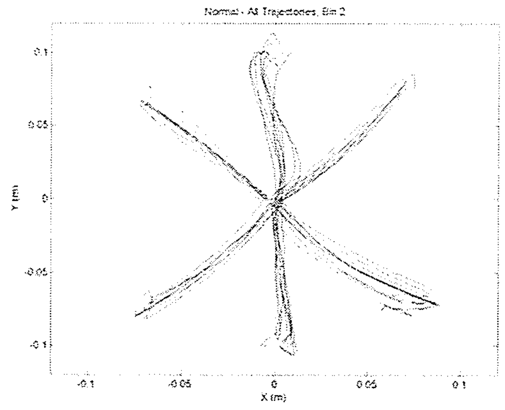
A single PD patient was asked to perform the reaching tasks described in Group #1. The only difference was that the force field baseline trial set was removed to reduce the amount of time required for the experiment. The results of these trials were compared with those of a control subject with no motor impairments (Subject #1 – Group #1). The subject agreed to not take his medication before the experiment in order to allow for a more accurate characterization of the effects of the disease.

8.5.4 Results

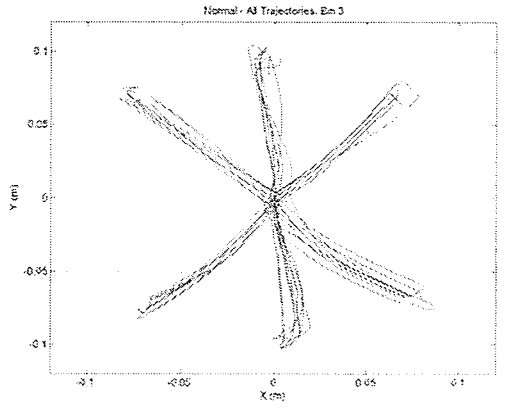
Since PD causes a tremor in patients, it was decided to compare the actual trajectories of the subjects as opposed to the smoothed average performance. Figure 8.18 shows the trajectories for a typical control subject, with each plot representing a bin of ten movements in each direction. The corresponding plots of the patient trajectories are presented in Figure 8.19. Figure 8.20 summarizes the average movement error performance of the control subjects from Group #1. Figure 8.21 shows the average movement error for each of the six bins for the patient. Finally, the average after-effect performance for each direction is shown in Table 8.5 for both the control subject and the patient.



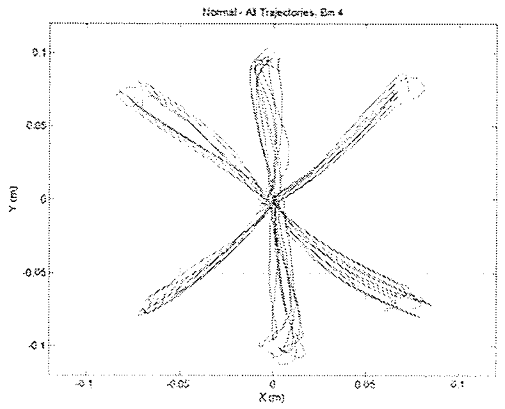
A



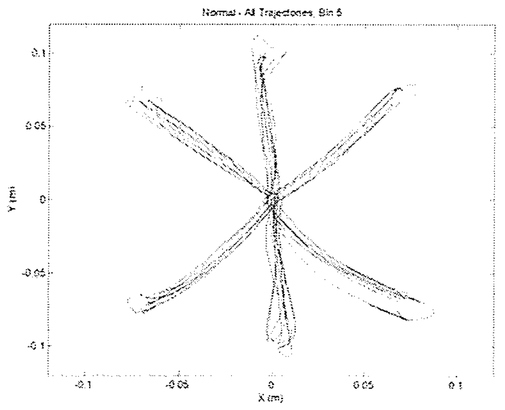
B



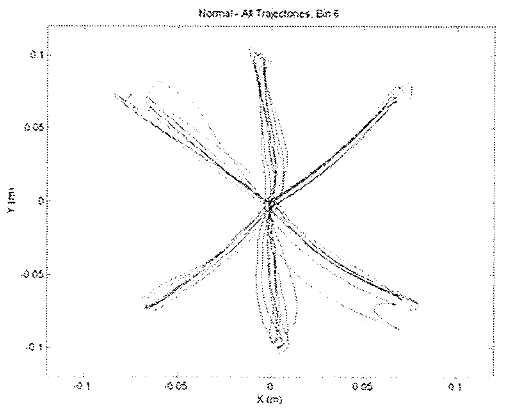
C



D

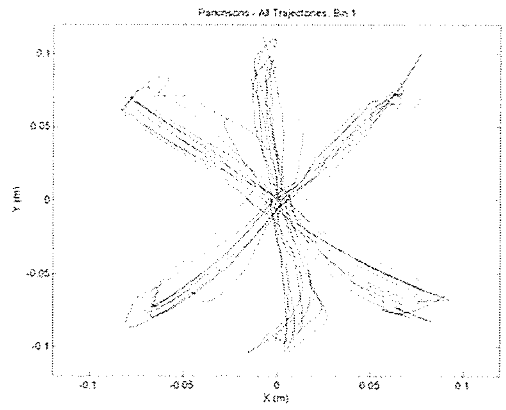


E

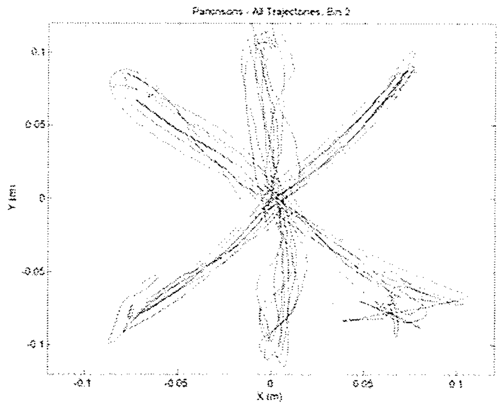


F

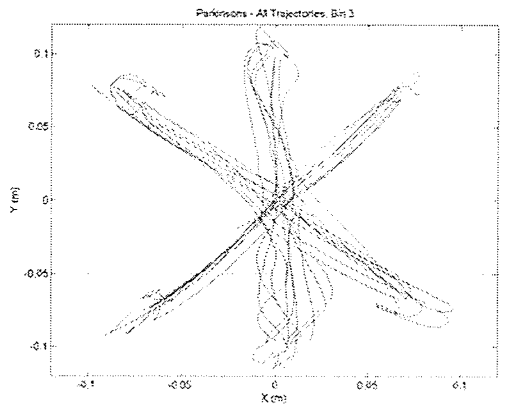
Figure 8.18: Control subject movement trajectories during adaptation.
A-F: Subsequent 10 move/direction bins



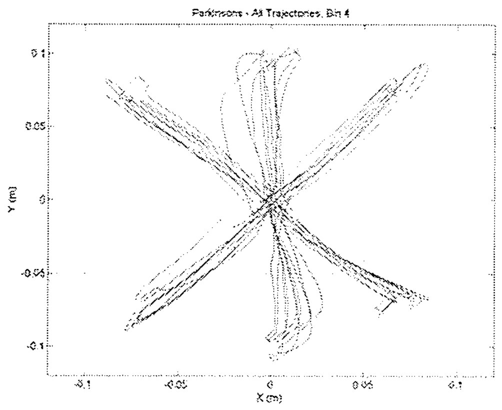
A



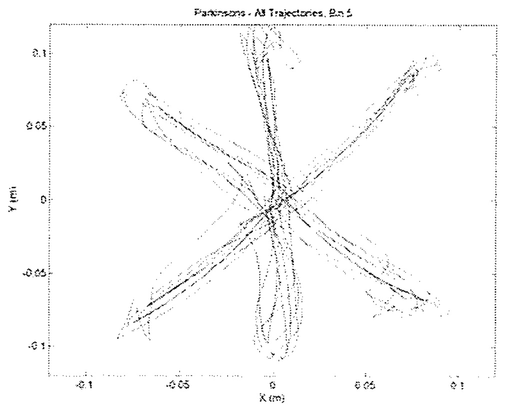
B



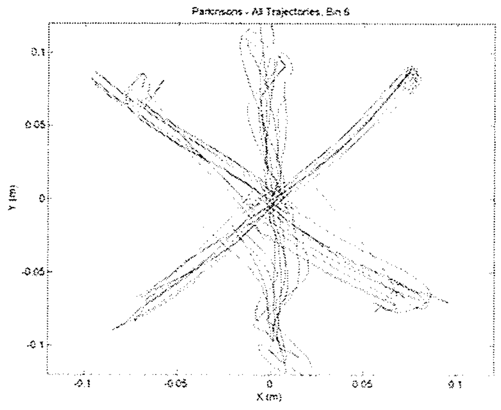
C



D



E



F

Figure 8.19: Patient movement trajectories during adaptation.
A-F: Subsequent 10 move/direction bins

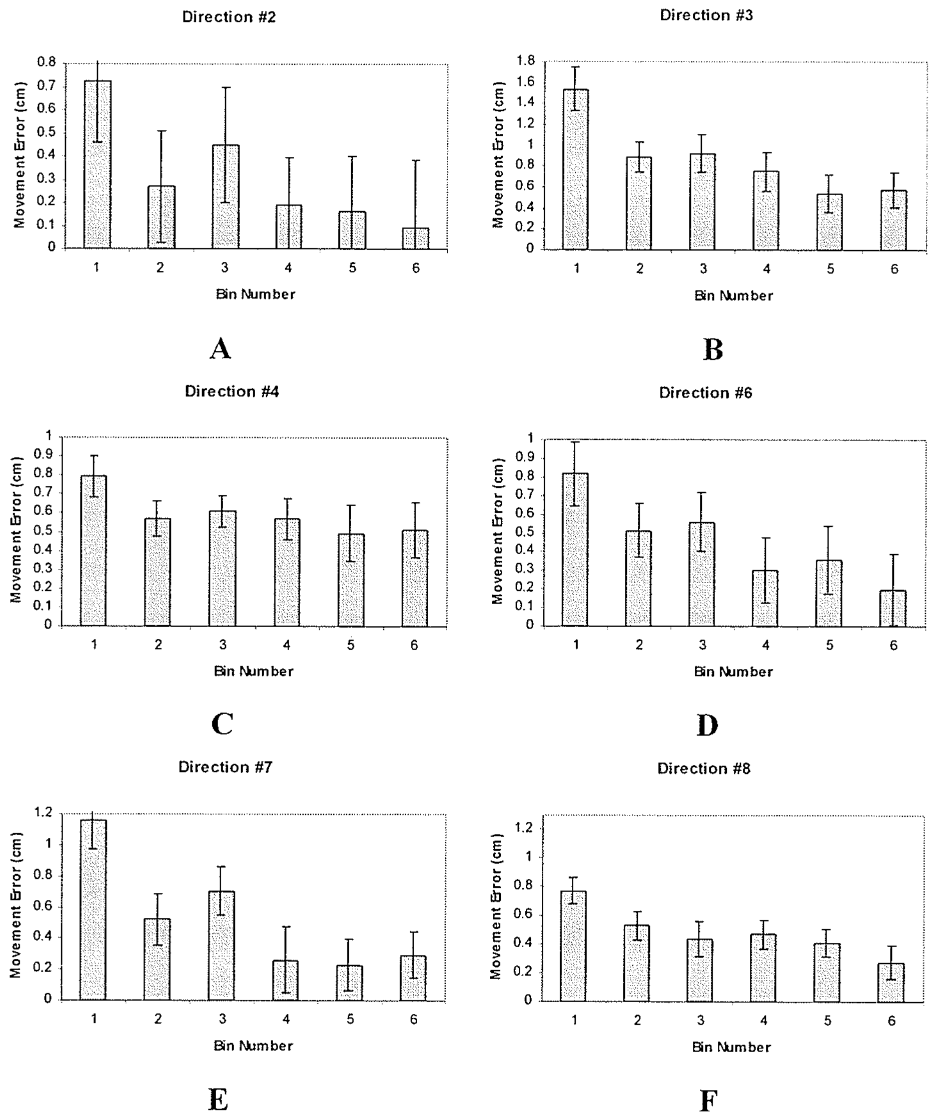


Figure 8.20: Control subject movement errors. Each bin represents the average for 10 trials, with bars showing standard deviation, A: Direction 2 (-45°), B: Direction 3 (-90°), C: Direction 4 (-135°), D: Direction 6 ($+135^\circ$), E: Direction 7 ($+90^\circ$), F: Direction 8 ($+45^\circ$)

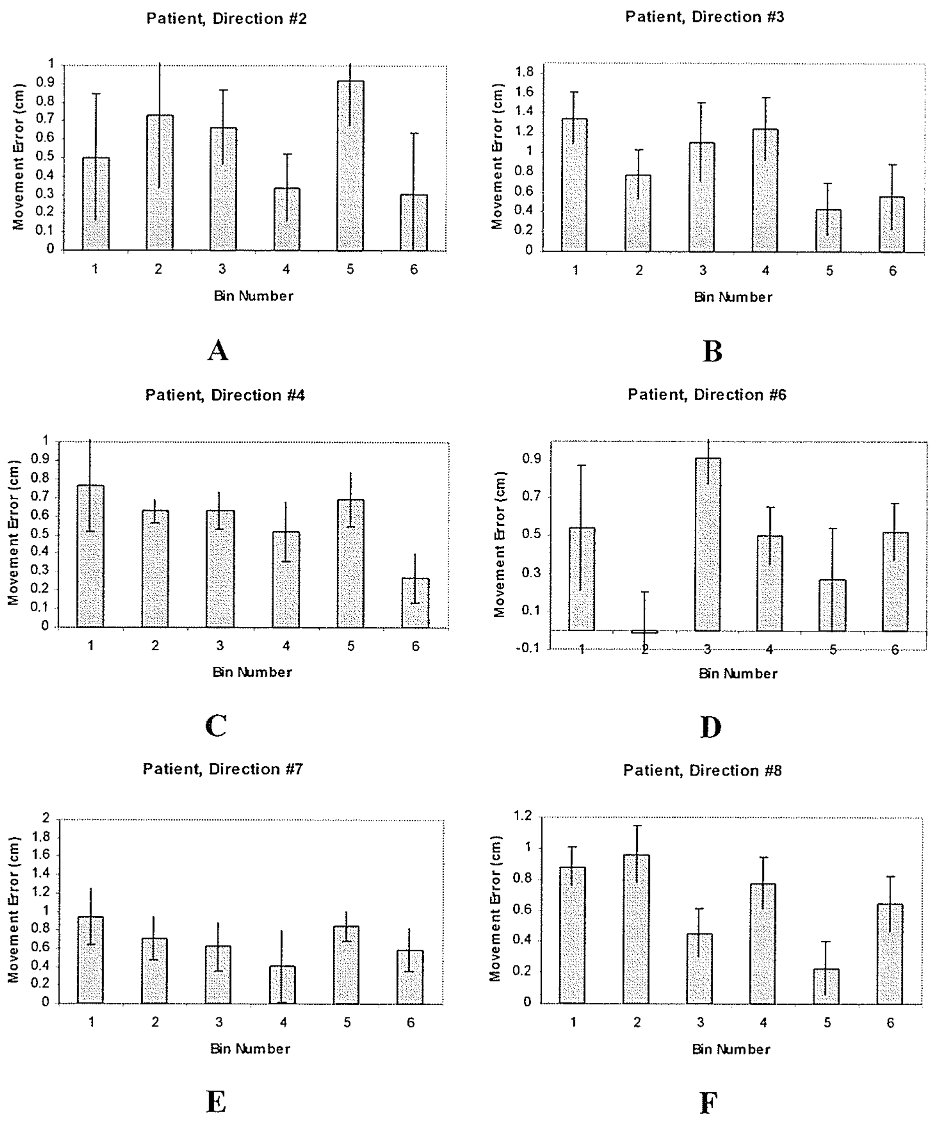


Figure 8.21: Patient movement errors.
Each bin represents the average for 10 trials, with bars showing standard deviation A: Direction 2 (-45°), B: Direction 3 (-90°), C: Direction 4 (-135°), D: Direction 6 ($+135^\circ$), E: Direction 7 ($+90^\circ$), F: Direction 8 ($+45^\circ$)

Table 8.5: After-effect performance comparison for each direction (Control vs. Patient)

	Control	Patient
Direction #2	-2.48	-0.87
Direction #3	-1.94	-0.92
Direction #4	-1.03	-0.43
Direction #6	-1.90	-0.96
Direction #7	-2.52	-2.51
Direction #8	-1.14	-0.62

8.5.5 Discussion

As was noted previously, the adaptation of the control subject to the force field resulted in a general decrease in movement error. This is shown directly in Figure 8.20 and is supported by the straight path trajectories presented in Figure 8.18. Note also that, as a result of the rest period between the third and fourth bins (Figures 8.18C and 8.18D respectively), the subject appears to have unlearned the force field to a small degree. This is evident in the slightly more erratic trajectories seen in Figure 8.18D.

The movement trajectories for the patient exhibit tremor and vibration as a result of their motor impairment. This is particularly evident in Directions #3 (-90°) and #7 $(+90^\circ)$, since the manipulator dynamics influenced the performance more in the diagonal directions. For this reason, this discussion will focus primarily on Directions #3 (-90°) and #7 $(+90^\circ)$. Despite the tremor, examination of Figures 8.19A and 8.19B shows that the patient did reduce the amount of movement error resulting from the force field. This is evidenced by the elimination of the hook shape that can be seen in Figure 8.19A (Direction #7 $(+90^\circ)$ especially). The reduction in movement error suggests that the patient exhibited adaptation and learned to compensate for the expected force field. The corresponding average movement error plots in Figure 8.21 also show this performance improvement for the first two bins of Directions #3 (-90°) and #7 ($+90^\circ$). Unfortunately, for many of the movement error calculations, the error resulting from tremor can skew the error averages, making the results in Figure 8.21 less reliable.

It should be noted that the patient fatigued quite easily and along with the fatigue came a loss of control over his hand tremor. This is seen in Figure 8.19C, where the movements become much more erratic than in the previous bin plot. After the rest between bins 3 and 4, the subject was able to regain control of their movements and regain their desired straight path movements. It can be seen in Figure 8.19D that the subject's performance still appears improved over that in the first bin, despite the rest. This shows that the patient's motor learning from the previous set of trials is persisting across the break. This is also the case for the control subject in Figure 8.18D. This result provides further support that this patient was able to adapt to the viscous force field.

Beyond the fourth bin, it was evident during the experiment that the subject was quite fatigued and that the lack of medication was resulting in even larger degrees of tremor. For this reason, the movement errors and trajectories beyond this point are not considered in this discussion.

With evidence supporting the adaptation to the force field, the properties of the resulting internal model were analyzed next using an after-effects analysis. After a longer rest period following the second set of adaptation targets (to allow the patient to regain control of their tremor), the after-effect targets were presented to the subject. The resulting after-effect movement errors are shown in Table 8.5. As with the control subject, all of the patient's after-effect errors were in the opposite direction of the applied force field. This result suggests that the disease did not impair the patient's ability to form an internal model of expected arm dynamics. This, along with the adaptation towards a straight path seen in Figure 8.19, shows that the impairment may not lie with the higher level planning structures of the brain. Once again, it is worth noting that the patient's internal model provided a sufficient predictive output force to permit negative error values despite fairly significant tremor.

Overall, the results of this pilot study support the idea that PD patients are still able to adapt to novel forces in their environment through the formation of an internal model. Such adaptation may be particularly useful in the development of rehabilitative tools, since this ability to adapt is essential to learning-based rehabilitation exercises.

CHAPTER IX

CONCLUSIONS AND RECOMMENDATIONS

9.1 Conclusions

In this report, the development and application of a 2-DOF robotic manipulandum was presented. This development focused on a lower-cost alternative to the traditionally more expensive systems. The experimental aspects of this work were meant to demonstrate motor learning in a more natural arm orientation and to also verify the performance of the low-cost system with respect to previously reported experimental research. Finally, with a verified system, a small pilot study was performed to see the potential relationships between PD and motor learning.

From a robotic development perspective, this work showed that a low cost geared DC servo system was capable of generating the required force fields for studying human motor control. In particular, an impedance control strategy was employed to allow the generation of endpoint-viscous forces based on the subject's hand velocity. Naturally, with the development of a low-cost system there was a trade off in performance. The use of planetary gearheads introduced backlash into the drive system, which resulted in increased standard error during movements. Also, this backlash hindered a test subject's ability to smoothly start and stop their motion. The gearheads also introduced extra friction components that affected the dynamics of the manipulator.

General motor learning was examined with the arm in a more natural non-abducted orientation. Test subjects exhibited a clear adaptation, through reduced movement error, to a perpendicular viscous force field designed to perturb their movements. The after-effects resulting from this adaptation showed that the adaptation was based on the formation of an internal model to predict the expected dynamics of the environment, and to automatically generate compensatory forces to regain a desired null field performance. These learning results also demonstrated that the low-cost manipulandum was able to provide insight into higher-level motor adaptation strategies.

Learning generalization was examined to determine if training with movements in one direction affected the performance in other directions. It was found that such a generalization occurred for directions rotated 45° from the training directions. The significance of this generalization was affected by how the dynamics of the robot (friction and backlash) altered the performance in the test directions. These results agree with past research and support the idea that the motor control system's internal model is composed of tunable basis functions with large receptive fields.

The final experimental area studied was learning within an uncertain force field. While previous work focused on such a field in a single direction, this work examined reaching in multiple directions. It was found that test subjects adapted to the approximate mean of the random force field quite quickly. Adaptation in a specific direction was seen to depend primarily on previous force field gains encountered in that direction and the rate of adaptation was found to be comparable to the previously reported single-direction results. The significance of this dependence was found to be lower than that in the past research since it was affected by the presence of intermediate trials in other directions. This work provided support for the idea that adaptation is based solely on recent memories.

Once the performance of the system was verified and the experimental results were found to be supported by previous research in the literature, the system was used to perform a pilot study on PD and motor learning. It was found that, for the single patient studied, PD does not impair the subject's ability to adapt to novel force fields and form an internal model of their expected arm dynamics. This result is promising since it shows that learning-based tools and activities could potentially be developed to aid in the rehabilitative exercises of PD sufferers.

9.2 Recommendations and Future Work

This section outlines the recommendations for future work related to this thesis. In particular it focuses on work to improve the mechanical performance of the manipulandum and the efficiency of the control software. It also addresses some specific ideas on where the PD related work should be headed, based on the results of the pilot study.

In order to solve the performance limitations of the manipulator, the gearhead backlash and friction must be corrected. The simplest method for doing this would be to replace the current motors with more expensive direct-drive DC torque motors. This would solve all of the drive-train issues discussed in Chapter 6. This motor replacement strategy would potentially involve a redesign of the support base as well, since the torque motors will be much larger in diameter. If the direct-drive option is not available, a more accurate friction model must be applied to completely compensate for the drive-train friction [Arms88]. Also, rather than replacing the motors, it could be possible to replace the gearheads with backlash-free gears, or at least with a gear system with reduced backlash.

From a control software perspective, the current implementation does not provide a lot of extra PC processing time for higher-level applications (such as rehabilitative games etc). In order to solve this issue, the impedance controller itself can be implemented on the PCI-7344 using its onboard programming capabilities. This would offload much of the work from the PC and permit more advanced user interface and application development.

In order to permit a wider array of potential experiments, two challenges must be met. First, a force transducer can be placed on the handle of the manipulator to measure interaction forces between the robot and test subject. This can either be purchased as a unit, or a 2D transducer can be constructed from strain gauges in a Wheatstone-bridge configuration, with their outputs being sent to the analog inputs of the PCI-7344. Second, in order to study the muscle forces involved in a particular reaching movement, the test subject's joint angles must be accurately estimated. For an abducted arm orientation, this is a simple procedure. However, for the more natural arm position studied in this work, the task is much more complicated. It was observed that test subjects controlled their movements using one rotational DOF at the elbow and three rotational DOF at the shoulder. Since, the current system is only capable of measuring the X, Y and Z coordinates of the subject's hand, it can only estimate three joint angles. To solve this problem, either the subjects must be restricted to use two DOF at the shoulder, or an accurate method of determining their elbow location must be developed. With this additional information all of the joint angles can then be found.

From an application perspective, further study is required to fully understand the effects of PD on the motor learning process. Many more patients are required to gain an appreciation of the full range of PD impairments. If the disease is found to have an effect on some aspect of motor learning, the progression of this effect must be determined by studying gene carriers, without any manifest PD, and patients with varying degrees of impairments. All of these subjects must be compared to control subjects with no motor impairments [SmBS00]. The results of these types of experiments will provide insight into whether rehabilitative tools and applications based on motor learning can be developed. The goal of these applications would be to provide exercises that the subjects can perform to appreciably improve their motor control, while also making the rehabilitation process more enjoyable. This can be achieved by adding a motivational game interface on the front-end of the impedance-controlled system.

REFERENCES

[Arms88] B. Armstrong, "Friction: Experimental determination, modeling and compensation," *IEEE Robotics and Automation Conference*, vol. 3, pp. 1422-1427, 1988.

[BaDe85] J.V. Basmajian and C.J. De Luca, *Muscles Alive*, Williams and Wilkins, Baltimore, MD, 4th edition, 1985.

[Bala03] S. Balakrishnan, *25.355: Robotics and Computer Numerical Control – Course Notes*, University of Manitoba, Department of Mechanical and Manufacturing Engineering, Winnipeg, MB, September 2003.

[Bala04] S. Balakrishnan, *25.774: Selected Topics in Robotics – Course Notes*, University of Manitoba, Department of Mechanical and Manufacturing Engineering, Winnipeg, MB, January 2004.

[BhSh99] N. Bhushan and R. Shadmehr, "Computational nature of human adaptive control during learning of reaching movements in force fields," *Biological Cybernetics*, vol. 81. pp 39-60, 1999.

[CoGM97] M.A. Conditt, F. Gandolfo and F.A. Mussa-Ivaldi, "The motor system does not learn the dynamics of the arm by rote memorization of past experience," *Journal of Neurophysiology*, vol. 78, pp. 554-560, 1997.

[CDGS02] S.E. Criscimagna-Hemminger, O. Donchin, M.S. Gazzaniga, and R. Shadmehr, "Learning dynamics of reaching movements generalizes from dominant to non-dominant arm," *Journal of Neurophysiology*, vol. 89, pp. 168-176, 2002.

[DoFS03] O. Donchin, J.T. Francis, and R. Shadmehr, "Quantifying generalization from trial-by-trial behavior of adaptive systems that learn with basis functions: Theory and experiments in human motor control," *Journal of Neuroscience*, vol. 23, no. 27, pp. 9032-9045, October 2003.

[DoSh02] O. Donchin and R. Shadmehr, "Linking motor learning to function approximation: Learning in an unlearnable field," In: *Advances in Neural Information Processing Systems* (T.G. Dietrich, S. Becker and Z. Ghahramani, eds.), vol. 14, pp. 197-203, Cambridge, MA: MIT, 2002.

[Faye86] I.C. Faye, "An impedance controlled manipulandum for human movement studies," S.M. Thesis, MIT, Department of Mechanical Engineering, Cambridge, MA, 1986.

[FlGu92] T. Flash and I. Gurevich, "Arm stiffness and movement adaptation in external loads," *Proceedings of the Annual Conference on Engineering in Medicine and Biology*, vol. 13, pp. 885-886, 1992.

[Hodg96] A.J. Hodgson, "Optimal design of a two degree of freedom manipulandum for human motor control studies," *IEEE Engineering in Medicine and Biology Society Conference*, pp. 615-616, 1996.

[Hoga85a] N. Hogan, "Impedance control: An approach to manipulation: Part 1 - Theory," *ASME Journal of Dynamic Systems, Measurement, and Control*, vol. 107, no. 1, pp. 1-7, March 1985.

[Hoga85b] N. Hogan, "Impedance control: An approach to manipulation: Part 2 - Implementation," *ASME Journal of Dynamic Systems, Measurement, and Control*, vol. 107, no. 1, pp. 8-16, March 1985.

[Hoga85c] N. Hogan, "Impedance control: An approach to manipulation: Part 3 - Applications," *ASME Journal of Dynamic Systems, Measurement, and Control*, vol. 107, no. 1, pp. 16-24, March 1985.

[KaWo98] M. Kawato and D. Wolpert, "Internal models for motor control," *Novartis Foundation Symposium*, vol. 218, pp. 291-304, 1998.

[KhDo02] W. Khalil and E. Dombre, *Modeling, identification & control of robots*, Taylor and Francis, New York, NY, pp. 480, 2002.

[MoRH94] D.C. Montgomery, G.C. Runger, and N.F. Hubele, *Engineering Statistics*, John Wiley & Sons, New York, NY, pp. 471, 1994.

[MuPa00] F.A. Mussa-Ivaldi and J.L. Patton, "Robots can teach people how to move their arm," *IEEE International Conference on Robotics and Automation*, April 2000, pp. 300-305.

[Muss97] F.A. Mussa-Ivaldi, "Nonlinear force fields: a distributed system of control primitives for representing and learning movements," *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics Automation*, pp. 84-90, 1997.

[Muss99] F.A. Mussa-Ivaldi, "Modular features of motor control and learning," *Current Opinion in Neurobiology*, vol. 9, pp. 713-717, 1999.

[Nati99] National Instruments, *Universal Motion Interface Accessory User Guide*, 321941B-01, August 1999.

[Nati01a] National Instruments, *Motion Control: FlexMotion Software 5.1.1 Reference Manual*, 370172C-01, April 2001.

[Nati01b] National Instruments, *Motion Control: 7344/7334 Hardware User Manual*, 322504C-01, August 2001.

[Ogat01] K. Ogata, *Modern Control Engineering*, Fourth Edition, Prentice Hall, Upper Saddle River, NJ, pp.964, 2001.

[PaLK03] R. Pahwa, K.E. Lyons, and W.C. Koller, *Handbook of Parkinson's disease*, Third Edition, Electronic Reproduction, NetLibrary, New York, NY, pp. 597, 2003.

[Reli92] Reliance Electric Company, *MAX-100 PWM Servo Drive Instruction Manual*, Document 0013-1015-001, Revision A, 1992.

[ScAt98] S. Schaal and C. Atkeson, "Constructive incremental learning from only local information," *Neural Computation*, vol. 10, pp. 2047-2084, 1998.

[ScDM01] R.A. Scheidt, J.B. Dingwell and F.A. Mussa-Ivaldi, "Learning to move amid uncertainty," *Journal of Neurophysiology,* vol. 86, no. 2, pp. 971-985, August 2001.

[ShMo00] R. Shadmehr and Z.M.K. Moussavi, "Spatial generalization from learning dynamics of reaching movements," *Journal of Neuroscience*, vol. 20, no. 20, pp. 7807-7815, October 2000.

[ShMu94] R. Shadmehr and F.A. Mussa-Ivaldi, "Adaptive representation of dynamics during learning of a motor task," *Journal of Neuroscience*, vol. 14, no. 5, pp. 3208-3224, May 1994.

[SmBS00] M.A. Smith, J. Brandt and R. Shadmehr, "Motor disorder in Huntington's disease begins as a dysfunction in error feedback control," *Nature*, vol. 403, pp. 544-549, February 2000.

[ThSh00] K.A. Thoroughman and R. Shadmehr, "Learning of action through adaptive combination of motor primitives," *Nature*, vol. 407, pp. 742-747, October 2000.

[Tsai99] L.Tsai, *Robot Analysis: The Mechanics of Serial and Parallel Manipulators*, John Wiley & Sons, New York, NY, pp. 505, 1999.

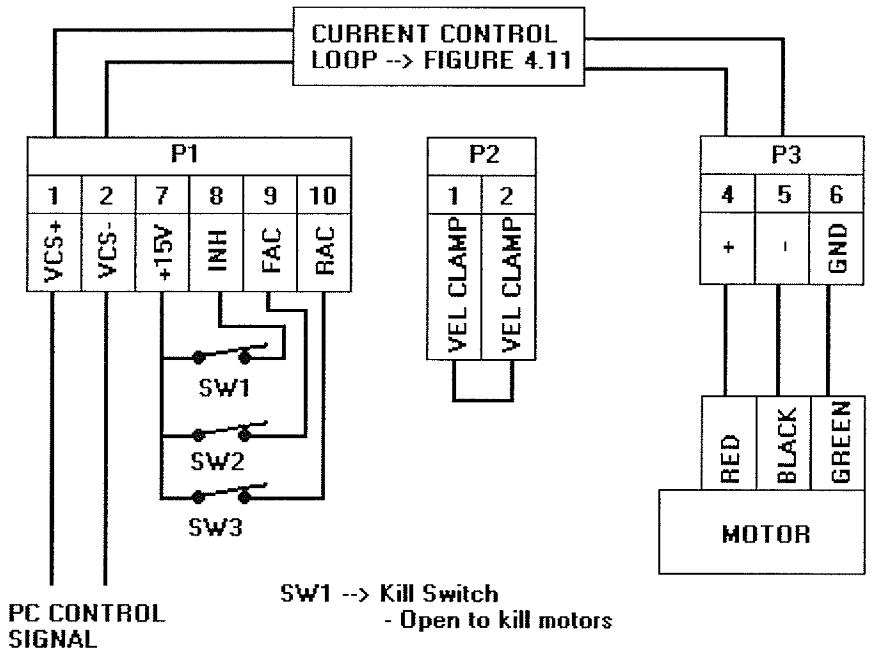
[Wint00] J.M. Winters, "Terminology and foundations of movement science," *Biomechanics and Neural Control of Posture and Movement*, Chapter 1, pp. 1-35, Springer-Verlag, New York, NY, 2000.

[WiSh02] S.P. Wise and R. Shadmehr, "Motor Control," *Encyclopedia of the Human Brain*, vol. 3, pp. 1-21, Elsevier Science, 2002.

APPENDIX A

DETAILED CIRCUIT DIAGRAM

→ Note: The following represents the cirsuit for a single motor
→ For full detailed information refer to [Reli92]



SW1 --> Kill Switch
- Open to kill motors

SW2 --> Forward Limit Switch
- Normally closed

SW3 --> Reverse Limit Switch
- Normally closed

APPENDIX B

TORQUE REQUIREMENT EXPERIMENT

For several locations in throughout the workspace, the torques required for generating a 1N force in each of four directions were found using *torqueTest.m*. This function makes use of the inverse kinematic solution implemented in *inverse.m*. Assuming a maximum torque output of 10.7 Nm (from 70:1 gear ratio), the maximum possible force in each direction was calculated. The minimum of these possible torques gives the force value that this system should be able to produce throughout the workspace in any direction. The results are summarized in the following table, and the required Matlab code is presented after the table.

Position		1 N Force		Required Torque		Max Force = Max Torque / Required Torque	
X	Y	Fx	Fy	T1	T2		
0.3	0	1	0	0	0.3162	#DIV/0!	33.83934
		0	1	0.3	0.15	35.66667	71.33333
		0.7071	0.7071	0.2121	0.3297	50.4479	32.45375
		0.7071	-0.7071	-0.2121	0.1175	50.4479	91.06383
0.4	0	1	0	0	0.2872	#DIV/0!	37.25627
		0	1	0.4	0.2	26.75	53.5
		0.7071	0.7071	0.2828	0.3445	37.83593	31.05951
		0.7071	-0.7071	-0.2828	0.0617	37.83593	173.4198
0.5	0	1	0	0	0.2449	#DIV/0!	43.6913
		0	1	0.5	0.25	21.4	42.8
		0.7071	0.7071	0.3536	0.35	30.26018	30.57143
		0.7071	-0.7071	-0.3536	-0.0036	30.26018	2972.222
0.3	0.1	1	0	-0.1	0.2462	107	43.4606
		0	1	0.3	0.0513	35.66667	208.577
		0.7071	0.7071	0.1414	0.2104	75.67185	50.85551
		0.7071	-0.7071	-0.2828	0.1379	37.83593	77.59246
0.4	0.1	1	0	-0.1	0.2244	107	47.68271
		0	1	0.4	0.1314	26.75	81.43075
		0.7071	0.7071	0.2121	0.2516	50.4479	42.52782
		0.7071	-0.7071	-0.3536	0.0658	30.26018	162.614
0.5	0.1	1	0	-0.1	0.1851	107	57.80659
		0	1	0.5	0.203	21.4	52.70936
		0.7071	0.7071	0.2828	0.2744	37.83593	38.99417
		0.7071	-0.7071	-0.4243	-0.0126	25.21801	849.2063
0.3	-0.1	1	0	0.1	0.3462	107	30.90699
		0	1	0.3	0.2487	35.66667	43.02372
		0.7071	0.7071	0.2828	0.4207	37.83593	25.4338
		0.7071	-0.7071	-0.1414	0.0689	75.67185	155.2975
0.4	-0.1	1	0	0.1	0.3244	107	32.98397
		0	1	0.4	0.2686	26.75	39.83619
		0.7071	0.7071	0.3536	0.4193	30.26018	25.51872
		0.7071	-0.7071	-0.2121	0.0395	50.4479	270.8861
0.5	-0.1	1	0	0.1	0.2851	107	37.53069
		0	1	0.5	0.297	21.4	36.02694
		0.7071	0.7071	0.4243	0.4116	25.21801	25.99611
		0.7071	-0.7071	-0.2828	-0.0084	37.83593	1273.81

Minimum Value: **21.4 N**

torqueTest.m

```
function [TOR] = torqueTest(Fx,Fy,pX,pY)

%find the joint angles for the current position
[theta1,theta2] = inverse(pX/100,pY/100);

%link lengths
a1 = 0.35;
a2 = 0.25;

%trigonometric functions
c1 = cos(theta1);
s1 = sin(theta1);
c12 = cos(theta1+theta2);
s12 = sin(theta1+theta2);

%jacobian
J = [-(a1*s1+a2*s12) -a2*s12;(a1*c1+a2*c12) a2*c12];

%find torques using irtual work
TOR = J'*[Fx;Fy];
```

inverse.m

```
function [T1,T2] = inverse(X,Y)

a1 = 0.35;
a2 = 0.25;
%solve for theta 2
T2 = acos((X^2+Y^2-a1^2-a2^2)/(2*a1*a2));

K1 = X;
K2 = Y;
K3 = a1+a2*cos(T2);
K4 = a2*sin(T2);

s1 = K2*K3-K1*K4;
c1 = K1*K3+K2*K4;

%solve for theta 1
T1 = atan2(s1,c1);
```

APPENDIX C

MANIPULATOR CODE

Manipulator.hpp:

```
#ifndef _MANIPULATOR
#define _MANIPULATOR

#include "FlexMotn.h"
#include <math.h>

#define INPUTVECTOR 0xFF
#define MASK1 0x02
#define MASK2 0x04

#define A1 0.35 //link lengths
#define A2 0.25
#define M1 0 //mass in kg of link 1
#define M2 0 //mass in kg of link 2
#define G 0 //9.81 //acceleration due to gravity (m/s^2)

#define KT 0.054232718  //motor torque constant (N-m/A) - [7.68 oz-in/A]
#define MAX_CURRENT 2.82  //maximum current (amps) - corresponds to +10V
#define MAX_VOLT 10
#define STEP 0.000305180437933928435187304493781 95  //DAC quantization step size
#define ENCODERCOUNT 4000  //number of encoder pulses in one revolution
#define N 70 //gear ratio

//NOTE: for directions, friction and offset, 3 removed
//      1 becomes old 2, 2 becomes old 3
#define DIR1 -1
#define DIR2 1

#define BPOS1 0.86
#define BNEG1 -0.86
#define SLOPE1 0.09
#define SLOPEMID1 3.8
#define BPOS2 1.02
#define BNEG2 -1.02
#define SLOPE2 0.09
#define SLOPEMID2 3.5

#define OFFSET1 0.095
#define OFFSET2 0.035

const float RADIANCONST = 2*M_PI/(ENCODERCOUNT*N);
const float TORQUECONST = MAX_VOLT/(MAX_CURRENT*KT*N);

class Manipulator {
 public:
   __fastcall Manipulator(u8 iboardID);
   __fastcall ~Manipulator();
   void __fastcall getPosition(float& posX, float& posY);
   void __fastcall getAngles(float& angle1,float& angle2);
   void __fastcall getVelocity(float& velX,float& velY);
   void __fastcall getTrajectory(float& posX, float& posY,float& velX,float& velY);
   void __fastcall setTorques(const float& tor1, const float& tor2);
   void __fastcall getTorques(float& tor1, float& tor2);
   void __fastcall jacobian(const float& in1,const float& in2,float& out1,float& out2);
   void __fastcall jacobianTrans(const float& in1,const float& in2,float& out1,float& out2);
   void __fastcall getGravityTorques(float& tor1, float& tor2);
   void __fastcall updateMotionInfo();
 private:
  u8 boardID;          // The identification number of the controller board

  //private utility variables
  i32 status;
  i32 temp;
  i16 outVal1;
  i16 outVal2;

  float torque1;    // last commanded torque value
  float torque2;

  //values for friction compensation
  float interPos1;
  float interNeg1;
  float velPos1;
  float velNeg1;
  float interPos2;
  float interNeg2;
  float velPos2;
  float velNeg2;

  //last updated joint angles (radians)
  float theta1, theta2, thetaTemp;
  //last updated joint velocities (rad/sec)
  float delta1, delta2, deltaTemp;
  //required sines and cosines of joint angles
  float c1, s1; //theta1
  float c2, s2; //theta2
  float c12, s12; //theta1+theta2
  float cSum, sSum;
  //last updated global cartesian position
  float positionX, positionY;
  //last updated global cartesian velocity
  float velocityX, velocityY;

  //private function to find friction compensation torque
  float __fastcall getFrictionTorque(const u8& axisNum);
};

#endif
```

Manipulator.cpp:

```
//TO DO:
// - setting of initial positions in constructor- use home and limit switches

#include "Manipulator.hpp"

//==========================================================================
// Default Constructor
//==========================================================================
__fastcall Manipulator::Manipulator(u8 iboardID)
{
 boardID = iboardID;

 Lock = new TCriticalSection();

 //initialize the flex motion board
 status = flex_initialize_controller(boardID,NULL);

 //must kill and disable all of the axes to create Axis instances
 //must unmap all of the assigned axis resources as well
 status = flex_stop_motion(boardID,0,NIMC_KILL_STOP,0x000F);
 status = flex_enable_axes(boardID,0,3,0x00);
```

```
  status = flex_config_axis(boardID,NIMC_AXIS1,0,0,0,0);
  status = flex_config_axis(boardID,NIMC_AXIS2,0,0,0,0);
  status = flex_config_axis(boardID,NIMC_AXIS3,0,0,0,0);
  status = flex_config_axis(boardID,NIMC_AXIS4,0,0,0,0);

  status = flex_config_axis(boardID,NIMC_AXIS1,NIMC_ENCODER1,0,0,0);
  status = flex_config_axis(boardID,NIMC_AXIS2,NIMC_ENCODER2,0,0,0);

  //enable the created axes
  u8 mask = MASK1|MASK2;
  status = flex_enable_axes(boardID,0,3,mask);

  //set friction related values
  velPos1 = BPOS1/SLOPEMID1;
  velNeg1 = BNEG1/SLOPEMID1;
  interPos1 = BPOS1-SLOPE1*velPos1;
  interNeg1 = BNEG1-SLOPE1*velNeg1;
  velPos2 = BPOS2/SLOPEMID2;
  velNeg2 = BNEG2/SLOPEMID2;
  interPos2 = BPOS2-SLOPE2*velPos2;
  interNeg2 = BNEG2-SLOPE2*velNeg2;

  flex_reset_pos(boardID, NIMC_AXIS1, 0, 0 ,INPUTVECTOR);
  flex_reset_pos(boardID, NIMC_AXIS2, -N*ENCODERCOUNT/4, 0 ,INPUTVECTOR);

  //update all of the stored motion variables
  updateMotionInfo();
  //initialize with zero torque commanded to motors
  setTorques(0,0);
} // Manipulator::Manipulator(u8 iboardID)

//============================================================================
// Manipulator::~Manipulator()
// - destructor
//============================================================================
__fastcall Manipulator::~Manipulator()
{
  setTorques(0,0);
  status = flex_stop_motion(boardID,0,NIMC_KILL_STOP,0x000F);
  status = flex_enable_axes(boardID,0,3,0x00);
} // Manipulator::~Manipulator()

//============================================================================
// Manipulator::getPosition()
// - return the most recent global cartesian coordinates of the endpoint
//============================================================================
void __fastcall Manipulator::getPosition(float& posX, float& posY)
{
  posX = positionX;
  posY = positionY;
} // Manipulator::getPosition(float& posX, float& posY, float& posZ)

//============================================================================
// Manipulator::getAngles()
// - return the most recently read joint angles
//============================================================================
void __fastcall Manipulator::getAngles(float& angle1,float& angle2)
{
  angle1 = theta1;
  angle2 = theta2;
} // Manipulator::getAnglesRad(float& angle1,float& angle2, float& angle3)

//============================================================================
// Manipulator::getVelocity()
// - return the most recent global cartesian velocity of the endpoint
//============================================================================
void __fastcall Manipulator::getVelocity(float& velX,float& velY)
{
  velX = velocityX;
  velY = velocityY;
```

```
} // Manipulator::getVelocity(float& velX,float& velY, float& velZ)

void __fastcall Manipulator::getTrajectory(float& posX, float& posY,float& velX,float& velY)
{
 posX = positionX;
 posY = positionY;

 velX = velocityX;
 velY = velocityY;
}

//===========================================================================
// Manipulator::setTorques()
// - set the motor torques to the desired values
// - these torque values will come from an external controller
//   and will depend on the control strategy being used
//===========================================================================
void __fastcall Manipulator::setTorques(const float& tor1, const float& tor2)
{
 torque1 = DIR1*TORQUECONST*(tor1+getFrictionTorque(NIMC_AXIS1)) + OFFSET1;
 torque2 = DIR2*TORQUECONST*(tor2+getFrictionTorque(NIMC_AXIS2)) + OFFSET2;

 //saturate
 torque1 = ((torque1>MAX_VOLT) ? MAX_VOLT : ((torque1<-MAX_VOLT) ? -MAX_VOLT : torque1));
 torque2 = ((torque2>MAX_VOLT) ? MAX_VOLT : ((torque2<-MAX_VOLT) ? -MAX_VOLT : torque2));

 status = flex_load_dac(boardID,NIMC_DAC1,(i16)floor(torque1/STEP),INPUTVECTOR);
 status = flex_load_dac(boardID,NIMC_DAC2,(i16)floor(torque2/STEP),INPUTVECTOR);

} // Manipulator::setTorques(float tor1, float tor2, float tor3)

//===========================================================================
// Manipulator::getTorques()
// - return the current commanded torque values for the motors
//===========================================================================
void __fastcall Manipulator::getTorques(float& tor1, float& tor2)
{
 tor1 = torque1;
 tor2 = torque2;
} // Manipulator::getTorques(float& tor1, float& tor2, float& tor3)

//===========================================================================
// Manipulator::jacobian()
// - multiply the three inputs by the jacobian of the manipulator and
//   return the result
// - based on most recently read joint angles
//===========================================================================
void __fastcall Manipulator::jacobian(const float& in1,const float& in2,float& out1,float& out2)
{
 out1 = ((-A1*s1-A2*s12)*in1-A2*s12*in2);
 out2 = ((A1*c1+A2*c12)*in1+A2*c12*in2);
} // Manipulator::jacobian(float in1,float in2,float in3,float& out1,float& out2,float& out3)

//===========================================================================
// Manipulator::jacobianTrans()
// - multiply the three inputs by the transpose of the jacobian of the manipulator and
//   return the result
// - based on most recently read joint angles
// - this is used for torque calculations of an impedance controller
//===========================================================================
void __fastcall Manipulator::jacobianTrans(const float& in1,const float& in2,float& out1,float& out2)
{
 out1 = ((-A1*s1-A2*s12)*in1+(A1*c1+A2*c12)*in2);
 out2 = (-A2*s12*in1+A2*c12*in2);
} // Manipulator::jacobianTrans(float in1,float in2,float in3,float& out1,float& out2,float& out3)

//===========================================================================
// Manipulator::getGravityTorques()
// - return the torques on the system joints due to gravity
//===========================================================================
void __fastcall Manipulator::getGravityTorques(float& tor1, float& tor2)
{
 tor1 = 0;
 tor2 = 0;
} // Manipulator::getGravityTorques(float& tor1, float& tor2, float& tor3)

//=============================================================================
// Manipulator::updateMotionInfo()
// - update the stored motion variables with new readings from the axes
// - simply a collection of function calls grouped together
//=============================================================================
void __fastcall Manipulator::updateMotionInfo()
{

 status = flex_read_pos_rtn(boardID,NIMC_AXIS1,&temp);
 theta1 = ((float)temp)*DIR1*RADIANCONST;
 status = flex_read_pos_rtn(boardID,NIMC_AXIS2,&temp);
 thetaTemp = ((float)temp)*DIR2*RADIANCONST;
 theta2 = thetaTemp-theta1+M_PI;

 status = flex_read_velocity_rtn(boardID,NIMC_AXIS1,&temp);
 delta1 = ((float)temp)*DIR1*RADIANCONST;
 status = flex_read_velocity_rtn(boardID,NIMC_AXIS2,&temp);
 deltaTemp = ((float)temp)*DIR2*RADIANCONST;
 delta2 = deltaTemp-delta1;

 c1 = cos(theta1);
 s1 = sin(theta1);
 c12 = cos(theta1+theta2);
 s12 = sin(theta1+theta2);

 positionX = A1*c1+A2*c12;
 positionY = A1*s1+A2*s12;

 jacobian(delta1,delta2,velocityX,velocityY);

} // Manipulator::updateMotionInfo()

//=============================================================================
//=============================================================================
// PRIVATE MEMBER FUNCTIONS
//=============================================================================
//=============================================================================

float __fastcall Manipulator::getFrictionTorque(const u8& axisNum)
{
 if (axisNum == NIMC_AXIS1)
 {
   if (delta1 >= velPos1)
   {
    return BPOS1;
   }
   else if (delta1 <= velNeg1)
   {
    return BNEG1;
   }
   else
   {
    return SLOPEMID1*delta1;
   }
 }
 else if (axisNum == NIMC_AXIS2)
 {
   if (delta2 >= velPos2)
   {
    return BPOS2;
   }
   else if (delta2 <= velNeg2)
   {
    return BNEG2;
    }
    else
    {
     return SLOPEMID2*delta2;
    }
  }
  else
  {
    return 0;
  }
}
```

APPENDIX D

CONTROLLER CODE

ControllerThread.h

```
//---------------------------------------------------------------------------

#ifndef ControllerThreadH
#define ControllerThreadH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include "Manipulator.hpp"
#include <vcl.h>

#define BOARD_ID 1 //flex motion software board ID
//---------------------------------------------------------------------------
class ControllerThread : public TThread
{
private:
  Manipulator* system;

  //last updated global cartesian position
  float currPosX, currPosY;
  //last updated global cartesian velocity
  float currVelX, currVelY;

  //utiliy variables for force calculation
  float posDiffX, posDiffY;
  float velDiffX, velDiffY;
  float FintX, FintY;

  float x0_X, x0_Y;
  float v0_X, v0_Y;

  //stiffness and viscosity matrix values
  float K11,K12,K21,K22;
  float B11,B12,B21,B22;
  float KGain;
  float BGain;

  float torque1, torque2;
protected:
  void __fastcall Execute();
public:
  __fastcall ControllerThread(bool CreateSuspended, Manipulator* manip);

  void __fastcall setK(float K11,float K12,float K21,float K22);
  void __fastcall setB(float B11,float B12,float B21,float B22);
  void __fastcall setKGain(float value);
  void __fastcall setBGain(float value);
  void __fastcall setVirtualPosition(const float& posX,const float& posY);
  void __fastcall setVirtualVelocity(const float& velX,const float& velY);

  void __fastcall getK(float& K11,float& K12,float& K21,float& K22);
  void __fastcall getB(float& B11,float& B12,float& B21,float& B22);
  void __fastcall getVirtualPosition(float& posX,float& posY);
  void __fastcall getVirtualVelocity(float& velX,float& velY);

  void __fastcall findTorques();
  void __fastcall setTorques();
  void __fastcall getTorques(float& tor1, float& tor2);
};
//---------------------------------------------------------------------------
```

```
#endif
```

ControllerThread.cpp:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "ControllerThread.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------

//   Important: Methods and properties of objects in VCL can only be
//   used in a method called using Synchronize, for example:
//
//      Synchronize(UpdateCaption);
//
//   where UpdateCaption could look like:
//
//      void __fastcall Unit2::UpdateCaption()
//      {
//        Form1->Caption = "Updated in a thread";
//      }
//---------------------------------------------------------------------------

//===========================================================================
// Constructor
// - create instance of ControllerThread class
// - set all parameters to default values
//===========================================================================
__fastcall ControllerThread::ControllerThread(bool CreateSuspended, Manipulator* manip)
        : TThread(CreateSuspended)
{
  system = manip;
  K11=K12=K21=K22 = 0;
  B11=B12=B21=B22 = 0;
  KGain = 1;
  BGain = 1;

  x0_X = 0;
  x0_Y = 0;
  v0_X = v0_Y = 0;

  system -> getPosition(currPosX,currPosY);
  system -> getVelocity(currVelX,currVelY);
}
//---------------------------------------------------------------------------
void __fastcall ControllerThread::Execute()
{
  while (!Terminated)
  {
    setTorques();
    findTorques();
    Suspend();
  }
}
//---------------------------------------------------------------------------

//===========================================================================
// Controller3D::setK()
//===========================================================================
void __fastcall ControllerThread::setK(float K11,float K12,float K21,float K22)
{
  this -> K11 = K11;
  this -> K12 = K12;
  this -> K21 = K21;
```

```
 this -> K22 = K22;
}

//===========================================================================
// Controller3D::setB()
//===========================================================================
void __fastcall ControllerThread::setB(float B11,float B12,float B21,float B22)
{
 this -> B11 = B11;
 this -> B12 = B12;
 this -> B21 = B21;
 this -> B22 = B22;
}

void __fastcall ControllerThread::setBGain(float value)
{
 BGain = value;
}
void __fastcall ControllerThread::setKGain(float value)
{
 KGain = value;
}

//===========================================================================
// Controller3D::setVirtualPosition()
//===========================================================================
void __fastcall ControllerThread::setVirtualPosition(const float& posX,const float& posY)
{
 x0_X = posX;
 x0_Y = posY;
}

//===========================================================================
// Controller3D::setVirtualVelocity()
//===========================================================================
void __fastcall ControllerThread::setVirtualVelocity(const float& velX,const float& velY)
{
 v0_X = velX;
 v0_Y = velY;
}

//===========================================================================
// Controller3D::getK()
//===========================================================================
void __fastcall ControllerThread::getK(float& K11,float& K12,float& K21,float& K22)
{
 K11 = this -> K11;
 K12 = this -> K12;
 K21 = this -> K21;
 K22 = this -> K22;
}

//===========================================================================
// Controller3D::getB()
//===========================================================================
void __fastcall ControllerThread::getB(float& B11,float& B12,float& B21,float& B22)
{
 B11 = this -> B11;
 B12 = this -> B12;
 B21 = this -> B21;
 B22 = this -> B22;
}

//===========================================================================
// Controller3D::getVirtualPosition()
//===========================================================================
void __fastcall ControllerThread::getVirtualPosition(float& posX,float& posY)
{
 posX = x0_X;
 posY = x0_Y;
```

```
}

//============================================================================
// Controller3D::getVirtualVelocity()
//============================================================================
void __fastcall ControllerThread::getVirtualVelocity(float& velX,float& velY)
{
 velX = v0_X;
 velY = v0_Y;
}

//============================================================================
// Controller3D::findTorques()
// - update the motion information (angles, velocities) of the manipulator
//   system
// - find desired interface force and calculate the required torque values
// - store these torque values for update on the next control loop
// - this is the general force calculation - currently the execute() function
//   is using a simplified method based on just velocity values
//============================================================================
void __fastcall ControllerThread::findTorques()
{
 system -> getTrajectory(currPosX,currPosY,currVelX,currVelY);

 //calculate difference from virtual
 posDiffX = x0_X - currPosX;
 posDiffY = x0_Y - currPosY;
 velDiffX = v0_X - currVelX;
 velDiffY = v0_Y - currVelY;

 //calculate desired force
 FintX = KGain*(K11*posDiffX+K12*posDiffY)+BGain*(B11*velDiffX+B12*velDiffY);
 FintY = KGain*(K21*posDiffX+K22*posDiffY)+BGain*(B21*velDiffX+B22*velDiffY);

 system -> jacobianTrans(FintX,FintY,torque1,torque2);
}

//============================================================================
// Controller3D::setTorques() - set motor torques and update manipulator motion info
//============================================================================
void __fastcall ControllerThread::setTorques()
{
 system -> updateMotionInfo();
 system -> setTorques(torque1,torque2);
}

//============================================================================
// Controller3D::getTorques()
//============================================================================
void __fastcall ControllerThread::getTorques(float& tor1, float& tor2)
{
 tor1 = torque1;
 tor2 = torque2;
} // Manipulator::getTorques(float& tor1, float& tor2, float& tor3)
```

APPENDIX E

EXPERIMENT CODE

ExperimentThread.h

```
//---------------------------------------------------------------------------

#ifndef ExperimentThreadH
#define ExperimentThreadH

#define ARRAY_LENGTH 1000

//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <stdio.h>
#include <fstream.h>
#include <vcl.h>
#include "Manipulator.hpp"
#include "ControllerThread.h"
//---------------------------------------------------------------------------
class ExperimentThread : public TThread
{
private:
  Manipulator* system;
  ControllerThread* controller;

  //position information
  float xPos, yPos;
  //velocity information
  float xVel, yVel;

  float velAbs;
  float velThresh;
  bool motionStarted;

  //output stream to data file
  ofstream fileOut;
  //input file stream for target data
  ifstream targetFile;

  //experiment parameters for controlling flow of runs
  int clockCount; //current clock counter value
  bool onRun; //currently performing reaching task
  bool experimentRunning; //currently running a set of trials
  int waitTime; //clock counts to wait before next target
  int maxRunLength; //top of target run time range
  int minRunLength; //bottom of target run time range
  int targetCount; //number of targets in current set of trials

  // target point
  float xTarget, yTarget;
  bool targetHit;
  int targetLengthVal;
  int cursorLengthVal;
  int targetHitVal;
  TColor targetColour;

  //controller update parameters
  float gain;
  int dir;

  //arrays for experiment parameters
  //set length of arrays to value larger than plan on using
```

```
    float xArray[ARRAY_LENGTH];
    float yArray[ARRAY_LENGTH];
    float gainArray[ARRAY_LENGTH];
    int dirArray[ARRAY_LENGTH];
    int runIndex;
    int numRuns;

    //visual interface control parameters
    bool visUpdate;
    TImage* xyImage;
    TCanvas* xyCanvas;
    int xRange;
    int yRange;
    int xCentre;
    int yCentre;
    double maxval;
    int xPixel,yPixel;
    int xPix,yPix;

    //private utility functions
    void __fastcall updateVisual();
    void __fastcall updateFile();
    void __fastcall updateTarget();
    void __fastcall updateController();
    int __fastcall roundValue(const float& value);
protected:
    void __fastcall Execute();
public:
    __fastcall ExperimentThread(bool CreateSuspended,Manipulator* manip,ControllerThread* cntrl,TImage* xyIm,int
xRangeVal, int yRangeVal, double maxVal);
    void __fastcall setWaitTime(int value);
    void __fastcall setRunTimes(int value, int tolerance);
    void __fastcall setTargetSize(int value);
    void __fastcall setCursorSize(int value);
    void __fastcall setVelocityThreshold(float value);
    void __fastcall openDataFile(char* fileName);
    void __fastcall closeDataFile();
    void __fastcall openTargetFile(char* fileName);
    void __fastcall closeTargetFile();
    void __fastcall begin();
    void __fastcall stop();
};
//---------------------------------------------------------------------------
#endif
```

ExperimentThread.cpp

```
//---------------------------------------------------------------------------
#include <stdio.h>
#include <fstream.h>
#include <vcl.h>

#pragma hdrstop

#include "abfSysUtils.hpp"

#include "ExperimentThread.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------

//   Important: Methods and properties of objects in VCL can only be
//   used in a method called using Synchronize, for example:
//
//      Synchronize(UpdateCaption);
//
//   where UpdateCaption could look like:
```

```
//
//    void __fastcall Unit2::UpdateCaption()
//    {
//      Form1->Caption = "Updated in a thread";
//    }
//---------------------------------------------------------------------------

__fastcall ExperimentThread::ExperimentThread(bool CreateSuspended,Manipulator* manip,ControllerThread*
cntrl,TImage* xyIm,int xRangeVal, int yRangeVal, double maxVal)
        : TThread(CreateSuspended)
{
 system = manip;
 controller = cntrl;
 clockCount = 0;

 visUpdate = true;
 experimentRunning = false;
 targetHit = false;
 onRun = false;
 motionStarted = false;

 targetLengthVal = 7;
 cursorLengthVal = 2;
 targetHitVal = targetLengthVal-cursorLengthVal;

 targetColour = clBlue;
 targetCount = 0;

 velThresh = 0.01;
 velAbs = 0;

 maxRunLength = 60;
 minRunLength = 40;
 waitTime = 100;

 runIndex = 0;
 numRuns = 0;

 //visual interface components
 xPos=yPos=0;
 xTarget=yTarget=0;
 xTarget = 0.4;
 xyImage = xyIm;
 xyCanvas = xyImage->Canvas;
 xRange=xRangeVal;
 yRange=yRangeVal;
 maxval = maxVal;
 xCentre=1+xRange/2;
 yCentre=1+yRange/2;
 xPixel=xPix=xCentre;
 yPixel=yPix=yCentre;
 xyCanvas->Brush->Style = bsSolid;
 updateVisual();
}

//---------------------------------------------------------------------------
void __fastcall ExperimentThread::updateVisual()
{
 //clear screen
 xyCanvas->Pen->Color = clBlack;
 xyCanvas->Brush->Color = clBlack;
 xyCanvas->Rectangle(0,0,xRange,yRange);

 //draw current target marker if on a set of trials
 if (experimentRunning)
 {
  xyCanvas->Brush->Color = targetColour;
  xyCanvas->Rectangle(xPix-targetLengthVal,yPix-targetLengthVal,xPix+targetLengthVal,yPix+targetLengthVal);
 }
```

```
  //draw marker for current position
  xyCanvas->Pen->Color = clWhite;
  xyCanvas->Brush->Color = clWhite;
  xyCanvas->Rectangle(xPixel-cursorLengthVal,yPixel-cursorLengthVal,xPixel+cursorLengthVal,yPixel+cursorLengthVal);

  //write debug info to screen
  xyCanvas->TextOutA(7*xRange/8,7*yRange/8,FloatToStr(runIndex));
  xyCanvas->TextOutA(xRange/4,7*yRange/8,IntToStr(motionStarted));

  //xyCanvas->TextOutA(7*xRange/8,1*yRange/8,FloatToStr(xVel));
  //xyCanvas->TextOutA(7*xRange/8,2*yRange/8,FloatToStr(yVel));

  xyImage->Repaint();
}
//---------------------------------------------------------------------------

void __fastcall ExperimentThread::updateFile()
{
  if (fileOut.is_open())
  {
    fileOut<<clockCount<<"\t"<<xPos<<"\t"<<yPos<<"\t"<<"\t"<<xVel<<"\t"<<yVel<<"\t"<<endl;
  }
}

void __fastcall ExperimentThread::updateTarget()
{
  //read next target point from array
  xTarget = xArray[runIndex];
  yTarget = yArray[runIndex];
  xPix = xCentre+roundValue(xRange*yTarget/(2*maxval));
  yPix = yCentre+roundValue(yRange*(xTarget-0.4)/(2*maxval));
}

void __fastcall ExperimentThread::updateController()
{
  //set gain
  gain = gainArray[runIndex];
  //set viscosity direction matrix
  dir = dirArray[runIndex];
  switch (dir)
  {
    case 1:
    case 5:
      controller->setB(0,1,0,0);
      break;
    case 2:
    case 6:
      controller->setB(0.5,0.5,-0.5,-0.5);
      break;
    case 3:
    case 7:
      controller->setB(0,0,-1,0);
      break;
    case 4:
    case 8:
      controller->setB(-0.5,0.5,-0.5,0.5);
      break;
    default:
      //ignore direction and only update gain value
      //do nothing, gain updated after switch statement
      break;
  }
  controller->setBGain(gain);

}

int __fastcall ExperimentThread::roundValue(const float& value)
{
  int floorVal = floor(value);
  if(value-floorVal < 0.5)
  {
   return floorVal;
  }
  else
  {
   return ceil(value);
  }
 }
//---------------------------------------------------------------------------

void __fastcall ExperimentThread::setWaitTime(int value)
{
  waitTime = value;
}

void __fastcall ExperimentThread::setRunTimes(int value, int tolerance)
{
  maxRunLength = value+tolerance;
  minRunLength = value-tolerance;
}

void __fastcall ExperimentThread::setTargetSize(int value)
{
  targetLengthVal = (value-1)/2;
  targetHitVal = targetLengthVal-cursorLengthVal;
}

void __fastcall ExperimentThread::setCursorSize(int value)
{
  cursorLengthVal = (value-1)/2;
  targetHitVal = targetLengthVal-cursorLengthVal;
}

void __fastcall ExperimentThread::setVelocityThreshold(float value)
{
  velThresh = value;
}

void __fastcall ExperimentThread::openDataFile(char* fileName)
{
  closeDataFile();
  fileOut.open(fileName);
}

void __fastcall ExperimentThread::closeDataFile()
{
  if (fileOut.is_open())
  {
   fileOut.close();
  }
}

void __fastcall ExperimentThread::openTargetFile(char* fileName)
{
  int tempDir;
  float tempX, tempY, tempGain;
  int index;

  closeTargetFile();
  targetFile.open(fileName);
  if (targetFile.is_open())
  {
   index = 0;
   //process the data file and place values in the appropriate arrays
   targetFile >> tempDir >> tempX >> tempY >> tempGain;
   while(!targetFile.eof())
   {
    dirArray[index] = tempDir;
    xArray[index] = tempX;
    yArray[index] = tempY;
```

```
      gainArray[index] = tempGain;

      index++;
      targetFile >> tempDir >> tempX >> tempY >> tempGain;
    }

    numRuns = index;
  }
}

void __fastcall ExperimentThread::closeTargetFile()
{
  if (targetFile.is_open())
  {
    targetFile.close();
  }
}

void __fastcall ExperimentThread::begin()
{
  experimentRunning = true;
  clockCount = 0;
  runIndex = -1;
}

void __fastcall ExperimentThread::stop()
{
  experimentRunning = false;
  clockCount = 0;
}

//---------------------------------------------------------------------------
void __fastcall ExperimentThread::Execute()
{
  while(!Terminated)
  {
    clockCount++;
    system -> getTrajectory(xPos,yPos,xVel,yVel);
    xPixel = xCentre+roundValue(xRange*yPos/(2*maxval));
    yPixel = yCentre+roundValue(yRange*(xPos-0.4)/(2*maxval));

    if (onRun)
    {
      velAbs = sqrt(pow(xVel,2)+pow(yVel,2));
      if (motionStarted == false && velAbs < velThresh)
      {
        clockCount = 0;
      }
      else
      {
        motionStarted = true;
      }

      updateFile();

      targetHit = (xPixel>=xPix-targetHitVal)&&(xPixel<=xPix+targetHitVal)&&(yPixel>=yPix-
targetHitVal)&&(yPixel<=yPix+targetHitVal);

      if (targetHit)
      {
        if (clockCount > maxRunLength)
        {
          targetColour = clYellow;
          abfSpeakerBeep(200,15);
        }
        else if(clockCount < minRunLength)
        {
          targetColour = clRed;
          abfSpeakerBeep(600,15);
        }
        else
        {
          targetColour = clGreen;
          abfSpeakerBeep(400,15);
        }
        onRun = false;
        clockCount = 0;
        //should field be set to zero between runs
        controller->setB(0,0,0,0);
      }
    }
    else //(not on run)
    {
      if (experimentRunning)
      {
        if (clockCount == waitTime)
        {
          //start next run
          onRun = true;
          motionStarted = false;
          clockCount = 0;
          targetColour = clBlue;
          abfSpeakerBeep(400,15);
          runIndex++;
          if (runIndex<numRuns)
          {
            //update the target marker
            updateTarget();
            //update the controller viscosity matrix
            updateController();
          }
          else
          {
            //finished set of trials for experiment
            experimentRunning = false;
            onRun = false;
          }
        }
      }
    }

    if (visUpdate)
    {
      //update the visual interface
      updateVisual();
    }
    visUpdate = !visUpdate;
    Suspend();
  }//while
}
//------------------------------------------------------------------------------
```

APPENDIX F

GUI & MAIN APPLICATION CODE

Manipulandum.cpp – MainApp:

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
USERES("Manipulandum.res");
USEFORM("Unit1.cpp", Form1);
USELIB("..\mar4\Manipulandum\FlexBC32.lib");
USEUNIT("ControllerThread.cpp");
USEUNIT("Manipulator.cpp");
USEUNIT("ExperimentThread.cpp");
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    try
    {
        Application->Initialize();
        Application->CreateForm(__classid(TForm1), &Form1);
        Application->Run();
    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
    return 0;
}
//---------------------------------------------------------------------------
```

Unit1.h:

```
//---------------------------------------------------------------------------

#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <ExtCtrls.hpp>
#include "abfComponents.hpp"
//---------------------------------------------------------------------------

class TForm1 : public TForm
{
__published:        // IDE-managed Components
    TPageControl *TabControl;
    TTabSheet *ControllerSettings;
    TTabSheet *ExperimentSettings;
    TGroupBox *ImpedanceGroup;
    TLabel *Label2;
    TLabel *Label3;
    TLabel *Label4;
    TLabel *Label5;
TLabel *Label6;
TLabel *Label7;
TLabel *Label9;
TLabel *Label10;
TEdit *X0_X;
TEdit *X0_Y;
TEdit *V0_X;
TEdit *V0_Y;
TEdit *K11;
TEdit *K12;
TEdit *K21;
TEdit *K22;
TEdit *B11;
TEdit *B12;
TEdit *B21;
TEdit *B22;
TButton *ImpedanceUpdate;
TGroupBox *JointGroup;
TLabel *Label12;
TLabel *Label14;
TLabel *Label13;
TLabel *Label15;
TLabel *Label16;
TLabel *Label17;
TLabel *Label18;
TEdit *Edit1;
TEdit *Edit2;
TEdit *Edit3;
TEdit *Edit4;
TEdit *Edit5;
TButton *JointUpdate;
TLabel *Label1;
TComboBox *ControllerSelect;
TButton *StartController;
TButton *StopController;
TLabel *Label21;
TLabel *Label22;
TLabel *Label24;
TLabel *Label25;
TLabel *Label26;
TLabel *Label27;
TLabel *Label29;
TLabel *Label30;
TLabel *Label31;
TImage *xyImage;
TabfThreadTimer *Timer1;
TLabel *clockLabel;
TLabel *Label19;
TEdit *outputFileBox;
TLabel *Label20;
TEdit *targetFileBox;
TLabel *Label32;
TLabel *Label33;
TLabel *Label34;
TLabel *Label35;
TEdit *waitTimeBox;
TEdit *runToleranceBox;
TEdit *runTimeBox;
TEdit *markerWidthBox;
TLabel *Label36;
TLabel *Label37;
TLabel *Label38;
TLabel *Label39;
TButton *initializeExperiment;
TButton *startExperiment;
TButton *stopExperiment;
TLabel *Label8;
TLabel *Label11;
TEdit *cursorWidthBox;
TEdit *BGainBox;
```

```
        TEdit *KGainBox;
        TLabel *Label23;
        TLabel *Label28;
        TLabel *Label40;
        TEdit *velThreshBox;
        TLabel *Label41;
        void __fastcall Timer1Timer(TObject *Sender);
        void __fastcall StartControllerClick(TObject *Sender);
        void __fastcall StopControllerClick(TObject *Sender);
        void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
        void __fastcall VisualInterfaceShow(TObject *Sender);
        void __fastcall ImpedanceUpdateClick(TObject *Sender);
        void __fastcall initializeExperimentClick(TObject *Sender);
        void __fastcall startExperimentClick(TObject *Sender);
        void __fastcall stopExperimentClick(TObject *Sender);
private:    // User declarations
public:                 // User declarations
        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

Unit1.cpp:

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include "ControllerThread.h"
#include "Manipulator.hpp"
#include "ExperimentThread.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "abfComponents"
#pragma resource "*.dfm"
TForm1 *Form1;

bool visual;

//variables related to the actual control thread
ControllerThread* controller;
ExperimentThread* experiment;
Manipulator* manip;

int clockCount;

//variables for controlling the subject visual interface
double maxval;

bool experBool;

//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
  experBool = true;

  clockCount = 0;

  //initialization of visual interface components
  visual=true;
  maxval = 0.105;
  //initialization of controller components
  manip = new Manipulator(BOARD_ID);
  controller = new ControllerThread(true,manip);
  experiment = new ExperimentThread(true,manip,controller,xyImage,xyImage->Width,xyImage->Height,maxval);
  experiment->Priority = tpTimeCritical;
  controller->Priority = tpTimeCritical;
}
//---------------------------------------------------------------------------


void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  //run the controller
  clockLabel->Caption = ++clockCount;
  controller -> Resume();
  //if (experBool == true)
  //{
   experiment -> Resume();
  //}
  //experBool = !experBool;
}
//---------------------------------------------------------------------------


void __fastcall TForm1::StartControllerClick(TObject *Sender)
{
 Timer1->Enabled = true;
 controller -> Resume();
 //experiment->Resume();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::StopControllerClick(TObject *Sender)
{
 Timer1->Enabled = false;
 controller -> Suspend();
 //experiment -> Suspend();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
  controller->Terminate();
  experiment->Terminate();
}


void __fastcall TForm1::ImpedanceUpdateClick(TObject *Sender)
{
 controller -> setKGain(StrToFloat(KGainBox->Text));
 controller -> setBGain(StrToFloat(BGainBox->Text));
 controller -> setK(StrToFloat(K11->Text),StrToFloat(K12->Text),StrToFloat(K21->Text),StrToFloat(K22->Text));
 controller -> setB(StrToFloat(B11->Text),StrToFloat(B12->Text),StrToFloat(B21->Text),StrToFloat(B22->Text));
 controller -> setVirtualPosition(StrToFloat(X0_X->Text),StrToFloat(X0_Y->Text));
 controller -> setVirtualVelocity(StrToFloat(V0_X->Text),StrToFloat(V0_Y->Text));
}
//---------------------------------------------------------------------------

void __fastcall TForm1::VisualInterfaceShow(TObject *Sender)
{
 xyImage->Repaint();
}


void __fastcall TForm1::initializeExperimentClick(TObject *Sender)
{
 experiment->setWaitTime(StrToInt(waitTimeBox->Text));
 experiment->setRunTimes(StrToInt(runTimeBox->Text),StrToInt(runToleranceBox->Text));
 experiment->setTargetSize(StrToInt(markerWidthBox->Text));
```

```
  experiment->setCursorSize(StrToInt(cursorWidthBox->Text));
  experiment->setVelocityThreshold(StrToFloat(velThreshBox->Text));
  experiment->openDataFile(outputFileBox->Text.c_str());
  experiment->openTargetFile(targetFileBox->Text.c_str());
}
//---------------------------------------------------------------------------

void __fastcall TForm1::startExperimentClick(TObject *Sender)
{
  experiment->begin();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::stopExperimentClick(TObject *Sender)
{
  experiment->closeDataFile();
  experiment->closeTargetFile();
  experiment->stop();
}
//---------------------------------------------------------------------------
```

APPENDIX G

DATA TRANSFORMATION PROGRAM

transformFile.m:

```
function dummy = transformFile(targetFile, oldFile, newFile)

dummy = 0;

%define position transformation matrices
%translate all so that (0.38,2) becomes (0,0)
originTrans = [1 0 -0.38;0 1 0.02;0 0 1];

%translate trajectories that don't start at the origin so that they do start at the origin
trans5 = [1 0 0;0 1 -0.1;0 0 1];
trans6 = [1 0 -.07071;0 1 -.07071;0 0 1];
trans7 = [1 0 -0.1;0 1 0;0 0 1];
trans8 = [1 0 -.07071;0 1 .07071;0 0 1];

%rotate everything by pi/2 to make directions coincide with what users saw on the screen
rot = [cos(-pi/2) -sin(-pi/2) 0;sin(-pi/2) cos(-pi/2) 0;0 0 1];

%velocity transformations will be such that:
%yVel = -xVel
%xVel = yVel

%read in target directions from target file
values = load(targetFile);
directions = values(:,1);
gains = values(:,4);

%read in recorded time, position and velocity information
values = load(oldFile);
times = values(:,1);
xPosition = values(:,2);
yPosition = values(:,3);
xVelocity = values(:,4);
yVelocity = values(:,5);

sum(times==10)

%open file to write transformed data to
FILE = fopen(newFile,'w');

%loop through movements, reformat and write to destination file
count = 2;
curr = 1;

%ignore the first movement in the file
while (curr <= length(times) && (times(curr) ~= 1))
  curr = curr+1;
end
%find end of current trajectory
next = curr+1;
while (next <= length(times) && times(next) ~= 0 && times(next) ~= 1)
  next = next+1;
end
curr = next+1;

while (count <= length(directions))
  %find start of next trajectory
  while (curr <= length(times) && (times(curr) ~= 1))
    curr = curr+1;
  end
  %find end of current trajectory
  next = curr+1;
  while (next <= length(times) && times(next) ~= 0 && times(next) ~= 1)
    next = next+1;
  end
  next = next-1;
  if (curr <= length(times) && next <= length(times))
    positions = [xPosition(curr:next) yPosition(curr:next) ones(size(xPosition(curr:next)))]';
    velocities = [xVelocity(curr:next) yVelocity(curr:next)]';
    timeVals = times(curr:next);
    for i=1:length(xPosition(curr:next))
      %transform each data point using the defined transformations
      pos = originTrans*positions(:,i);
      if (directions(count)==5)
        pos = trans5*pos;
      elseif (directions(count)==6)
        pos = trans6*pos;
      elseif (directions(count)==7)
        pos = trans7*pos;
      elseif (directions(count)==8)
        pos = trans8*pos;
      end
      pos = rot*pos;

      xVel = velocities(2,i);
      yVel = -velocities(1,i);

      %write following lines to file for current point
      %  TIME XPOS YPOS XVEL YVEL DIR GAIN
      fprintf(FILE,'%d\t%d\t%d\t%d\t%d\t%d\t%d\n',timeVals(i),pos(1),pos(2),xVel,yVel,directions(count),gains(count));
    end
  end

  curr = next+1;
  count = count+1;
end

%close the new data file
fclose(FILE);
```

APPENDIX H

TRAJECTORY TRIMMING FUNCTION

trimTrajectories.m:

```
function dummy = trimTrajectories(inputFile,outputFile)

%trims each trajectory so that it only includes the points up to the first
%transient minimum in the velocity profile

dummy = 0;

%open file to write transformed data to
FILE = fopen(outputFile,'w');

%load transformed file in following format:
%  TIME XPOS YPOS XVEL YVEL DIR GAIN
values = load(inputFile);

T = values(:,1);
sum((T==1));

xPos = values(:,2);
yPos = values(:,3);
xVel = values(:,4);
yVel = values(:,5);
DIR = values(:,6);
GAIN = values(:,7);

curr = 1;

plotBool = 1;
while (curr <= length(T))
   %find end of current trajectory
   next = curr+1;
   while (next <= length(T) & T(next) ~= 1)
      next = next+1;
   end
   next = next-1;

   positions = [xPos(curr:next) yPos(curr:next)];
   velocities = [xVel(curr:next) yVel(curr:next)];
   direction = DIR(curr);
   gainVal = GAIN(curr);

   %find the absolute velocities for each point in the trajectory
   absVel = sqrt(velocities(:,1).^2+velocities(:,2).^2);
            %find maximum velocity point
            [maxVel,maxVelIndex] = max(absVel);
            %find the first transient minimum in the velocity profile
            minVelIndex = maxVelIndex+20;
   length(absVel);
   if (minVelIndex > length(absVel))
      minVelIndex = length(absVel);
   else
                        while (minVelIndex <= length(absVel)-1 & absVel(minVelIndex) < absVel(minVelIndex-1) &
absVel(minVelIndex) > absVel(minVelIndex+1))
                           minVelIndex = minVelIndex+1;
                        end
   end

   newPos = positions(1:minVelIndex,:);
   newVel = velocities(1:minVelIndex,:);
```

```
    for i=1:minVelIndex
        %write following lines to file for current point
        %  TIME XPOS YPOS XVEL YVEL DIR GAIN
        fprintf(FILE,'%d\t%d\t%d\t%d\t%d\t%d\t%d\n',i,newPos(i,1),newPos(i,2),newVel(i,1),newVel(i,2),direction,gainVal);
    end

    curr = next+1;
end

%close the new data file
fclose(FILE);
```

APPENDIX I

MOVEMENT ERROR FUNCTIONS

findMovementErrors.m:

```
function dummy = findMovementErrors(dataFile1,dataFile2,dataFile3,desiredGain,nullMean)

dummy = zeros(1,8);

%load transformed file in following format:
% TIME XPOS YPOS XVEL YVEL DIR GAIN
values1 = load(dataFile1);
T = values1(:,1);
xPos = values1(:,2);
yPos = values1(:,3);
xVel = values1(:,4);
yVel = values1(:,5);
DIR = values1(:,6);
GAIN = values1(:,7);

if (dataFile2 ~= 0)
  values2 = load(dataFile2);
  T = [T;values2(:,1)];
        xPos = [xPos;values2(:,2)];
        yPos = [yPos;values2(:,3)];
        xVel = [xVel;values2(:,4)];
        yVel = [yVel;values2(:,5)];
        DIR = [DIR;values2(:,6)];
        GAIN = [GAIN;values2(:,7)];
end

if (dataFile3 ~= 0)
  values3 = load(dataFile3);
  T = [T;values3(:,1)];
        xPos = [xPos;values3(:,2)];
        yPos = [yPos;values3(:,3)];
        xVel = [xVel;values3(:,4)];
        yVel = [yVel;values3(:,5)];
        DIR = [DIR;values3(:,6)];
        GAIN = [GAIN;values3(:,7)];
end

errors = [];
times = [];
directions = [];
peakVels = [];

curr = 1;
moveCount = 1;

while (curr <= length(T))
  %find end of current trajectory
  next = curr+1;
  while (next <= length(T) & T(next) ~= 1)
    next = next+1;
  end
  next = next-1;
  if (GAIN(curr)==desiredGain)
    positions = [xPos(curr:next) yPos(curr:next)];
    velocities = [xVel(curr:next) yVel(curr:next)];
    [dispVal,timeVal] = findDisp(DIR(curr),positions,velocities);
    errors = [errors dispVal];
    times = [times timeVal];
```

```
        peakVels = [peakVels max(sqrt(velocities(:,1).^2+velocities(:,2).^2))];
        directions = [directions DIR(curr)];
    end
    curr = next+1;
    moveCount = moveCount+1;
end

%convert errors values to centimetres
errors = errors*100;

%extract errors series for each individual direction
errors2 = [];
errors3 = [];
errors4 = [];
errors6 = [];
errors7 = [];
errors8 = [];

times2 = [];
times3 = [];
times4 = [];
times6 = [];
times7 = [];
times8 = [];

peakVels2 = [];
peakVels3 = [];
peakVels4 = [];
peakVels6 = [];
peakVels7 = [];
peakVels8 = [];

for i=1:length(directions)
    if (directions(i) == 2)
        errors2 = [errors2 errors(i)];
        times2 = [times2 times(i)];
        peakVels2 = [peakVels2 peakVels(i)];
    elseif (directions(i) == 3)
        errors3 = [errors3 errors(i)];
        times3 = [times3 times(i)];
        peakVels3 = [peakVels3 peakVels(i)];
    elseif (directions(i) == 4)
        errors4 = [errors4 errors(i)];
        times4 = [times4 times(i)];
        peakVels4 = [peakVels4 peakVels(i)];
    elseif (directions(i) == 6)
        errors6 = [errors6 errors(i)];
        times6 = [times6 times(i)];
        peakVels6 = [peakVels6 peakVels(i)];
    elseif (directions(i) == 7)
        errors7 = [errors7 errors(i)];
        times7 = [times7 times(i)];
        peakVels7 = [peakVels7 peakVels(i)];
    elseif (directions(i) == 8)
        errors8 = [errors8 errors(i)];
        times8 = [times8 times(i)];
        peakVels8 = [peakVels8 peakVels(i)];
    end
end

errors2 = errors2-nullMean(2)
errors3 = errors3-nullMean(3)
errors4 = errors4-nullMean(4)
errors6 = errors6-nullMean(6)
errors7 = errors7-nullMean(7)
errors8 = errors8-nullMean(8)

figure
subplot(3,2,1)
plot([1:1:length(errors2)],errors2)
```

```
title('Movement Errors - direction 2')
grid
subplot(3,2,2)
plot([1:1:length(errors3)],errors3)
title('Movement Errors - direction 3')
grid
subplot(3,2,3)
plot([1:1:length(errors4)],errors4)
title('Movement Errors - direction 4')
grid
subplot(3,2,4)
plot([1:1:length(errors6)],errors6)
title('Movement Errors - direction 6')
grid
subplot(3,2,5)
plot([1:1:length(errors7)],errors7)
title('Movement Errors - direction 7')
grid
subplot(3,2,6)
plot([1:1:length(errors8)],errors8)
title('Movement Errors - direction 8')
grid


binSize = 60;
index1=1;
index2 = index1+binSize-1;
meanErrors = [];
stdErrors = [];
while (index1 <= length(errors))
   if (index2 > length(errors));
      index2 = length(errors);
   end
   values = errors(index1:index2);
   meanValue = mean(values);
   meanErrors = [meanErrors meanValue];
   stdValue = std(values);
   stdErrors = [stdErrors stdValue];
   index1 = index2+1;
   index2 = index1+binSize-1;
end
figure
bar(meanErrors,0.5);
title('Maximum Perpendicular Displacement - All Directions, red - std error')
hold
plot([1:1:length(stdErrors)],stdErrors,'rd-')
grid
dummy(1) = meanErrors(length(meanErrors)-1);

binSize = binSize/6;

index1=1;
index2 = index1+binSize-1;
meanErrors = [];
stdErrors = [];
while (index1 <= length(errors2))
   if (index2 > length(errors2));
      index2 = length(errors2);
   end
   values = errors2(index1:index2);
   meanValue = mean(values);
   meanErrors = [meanErrors meanValue];
   stdValue = std(values);
   stdErrors = [stdErrors stdValue];
   index1 = index2+1;
   index2 = index1+binSize-1;
end
meanErrors2 = meanErrors;
stdErrors2 = stdErrors;
figure
```

```
bar(meanErrors,0.5);
title('Maximum Perpendicular Displacement - Direction 2, red - std error')
hold
plot([1:1:length(stdErrors)],stdErrors,'rd-')
grid
dummy(2) = meanErrors(length(meanErrors)-1);

index1=1;
index2 = index1+binSize-1;
meanErrors = [];
stdErrors = [];
while (index1 <= length(errors3))
   if (index2 > length(errors3));
      index2 = length(errors3);
   end
   values = errors3(index1:index2);
   meanValue = mean(values);
   meanErrors = [meanErrors meanValue];
   stdValue = std(values);
   stdErrors = [stdErrors stdValue];
   index1 = index2+1;
   index2 = index1+binSize-1;
end
meanErrors3 = meanErrors;
stdErrors3 = stdErrors;
figure
bar(meanErrors,0.5);
title('Maximum Perpendicular Displacement - Direction 3, red - std error')
hold
plot([1:1:length(stdErrors)],stdErrors,'rd-')
grid
dummy(3) = meanErrors(length(meanErrors)-1);

index1=1;
index2 = index1+binSize-1;
meanErrors = [];
stdErrors = [];
while (index1 <= length(errors4))
   if (index2 > length(errors4));
      index2 = length(errors4);
   end
   values = errors4(index1:index2);
   meanValue = mean(values);
   meanErrors = [meanErrors meanValue];
   stdValue = std(values);
   stdErrors = [stdErrors stdValue];
   index1 = index2+1;
   index2 = index1+binSize-1;
end
meanErrors4 = meanErrors;
stdErrors4 = stdErrors;
figure
bar(meanErrors,0.5);
title('Maximum Perpendicular Displacement - Direction 4, red - std error')
hold
plot([1:1:length(stdErrors)],stdErrors,'rd-')
grid
dummy(4) = meanErrors(length(meanErrors)-1);

index1=1;
index2 = index1+binSize-1;
meanErrors = [];
stdErrors = [];
while (index1 <= length(errors6))
   if (index2 > length(errors6));
      index2 = length(errors6);
   end
   values = errors6(index1:index2);
   meanValue = mean(values);
   meanErrors = [meanErrors meanValue];
```

```
    stdValue = std(values);
    stdErrors = [stdErrors stdValue];
    index1 = index2+1;
    index2 = index1+binSize-1;
end
meanErrors6 = meanErrors;
stdErrors6 = stdErrors;
figure
bar(meanErrors,0.5);
title('Maximum Perpendicular Displacement - Direction 6, red - std error')
hold
plot([1:1:length(stdErrors)],stdErrors,'rd-')
grid
dummy(6) = meanErrors(length(meanErrors)-1);

index1=1;
index2 = index1+binSize-1;
meanErrors = [];
stdErrors = [];
while (index1 <= length(errors7))
    if (index2 > length(errors7));
        index2 = length(errors7);
    end
    values = errors7(index1:index2);
    meanValue = mean(values);
    meanErrors = [meanErrors meanValue];
    stdValue = std(values);
    stdErrors = [stdErrors stdValue];
    index1 = index2+1;
    index2 = index1+binSize-1;
end
meanErrors7 = meanErrors;
stdErrors7 = stdErrors;
figure
bar(meanErrors,0.5);
title('Maximum Perpendicular Displacement - Direction 7, red - std error')
hold
plot([1:1:length(stdErrors)],stdErrors,'rd-')
grid
dummy(7) = meanErrors(length(meanErrors)-1);

index1=1;
index2 = index1+binSize-1;
meanErrors = [];
stdErrors = [];
while (index1 <= length(errors8))
    if (index2 > length(errors8));
        index2 = length(errors8);
    end
    values = errors8(index1:index2);
    meanValue = mean(values);
    meanErrors = [meanErrors meanValue];
    stdValue = std(values);
    stdErrors = [stdErrors stdValue];
    index1 = index2+1;
    index2 = index1+binSize-1;
end
meanErrors8 = meanErrors;
stdErrors8 = stdErrors;
figure
bar(meanErrors,0.5);
title('Maximum Perpendicular Displacement - Direction 8, red - std error')
hold
plot([1:1:length(stdErrors)],stdErrors,'rd-')
grid
dummy(8) = meanErrors(length(meanErrors)-1);


OUT = [meanErrors2;meanErrors3;meanErrors4;meanErrors6;meanErrors7;meanErrors8];
save meanError.txt OUT -ascii
```

```
OUT = [stdErrors2;stdErrors3;stdErrors4;stdErrors6;stdErrors7;stdErrors8];
save stdError.txt OUT -ascii
```

findDisp.m:

```
function [disp,curr] = findDisp(dir,positions,velocities)

%for a given reaching trajectory calculate the maximum perpendicular
%displacement from the desired path.

%this uses input trajectories as from files converted using transformFile.m

%based on direction of the reaching movement, select the desired
%trajectory end point and use the first position as the starting point
Xstart = 0;
Ystart = 0;
Xend = 0;
Yend = 0;

T = [1:size(positions,1)];

if (dir == 1)
  Xend = 0.1;
  Yend = 0;
elseif (dir == 2)
  Xend = 0.07071;
  Yend = -0.07071;
elseif (dir == 3)
  Xend = 0;
  Yend = -0.1;
elseif (dir == 4)
  Xend = -0.07071;
  Yend = -0.07071;
elseif (dir == 5)
  Xend = -0.1;
  Yend = 0;
elseif (dir == 6)
  Xend = -0.07071;
  Yend = 0.07071;
elseif (dir == 7)
  Xend = 0;
  Yend = 0.1;
elseif (dir == 8)
  Xend = 0.07071;
  Yend = 0.07071;
else
  Xend = 0;
  Yend = 0;
end

%extract staring point from trajectory
Xstart = positions(1,1);
Ystart = positions(1,2);

%note: positive distance corresponds to the displacing force field
%direction
distances = [];
for i=1:size(positions,1)
    %find values for the equation of the line
    %Ax+By+C=0
    A = Yend-Ystart;
    B = Xstart-Xend;
    C = (Ystart-Yend)*Xstart+(Xend-Xstart)*Ystart;

    value = -(A*positions(i,1)+B*positions(i,2)+C)/sqrt(A^2+B^2);
```

```
   distances = [distances value];
end

curr = size(positions,1);

[dummy maxIndex] = max(abs(distances(1:curr)));

disp = distances(maxIndex);
```

APPENDIX J

CORRELATION ANALYSIS FUNCTION

```
function dummy = corrAnalysis(gains,directions,errors)

%extract errors series for each individual direction
errors2 = [];
errors3 = [];
errors4 = [];
errors6 = [];
errors7 = [];
errors8 = [];

gains2 = [];
gains3 = [];
gains4 = [];
gains6 = [];
gains7 = [];
gains8 = [];

for i=1:length(directions)-1
   if (directions(i) == 2)
      errors2 = [errors2 errors(i)];
      gains2 = [gains2 gains(i)];
   elseif (directions(i) == 3)
      errors3 = [errors3 errors(i)];
      gains3 = [gains3 gains(i)];
   elseif (directions(i) == 4)
      errors4 = [errors4 errors(i)];
      gains4 = [gains4 gains(i)];
   elseif (directions(i) == 6)
      errors6 = [errors6 errors(i)];
      gains6 = [gains6 gains(i)];
   elseif (directions(i) == 7)
      errors7 = [errors7 errors(i)];
      gains7 = [gains7 gains(i)];
   elseif (directions(i) == 8)
      errors8 = [errors8 errors(i)];
      gains8 = [gains8 gains(i)];
   end
end

% figure
% subplot(3,2,1)
% plot([1:1:length(errors2)],errors2)
% title('Movement Errors - direction 2')
% grid
% subplot(3,2,2)
% plot([1:1:length(errors3)],errors3)
% title('Movement Errors - direction 3')
% grid
% subplot(3,2,3)
% plot([1:1:length(errors4)],errors4)
% title('Movement Errors - direction 4')
% grid
% subplot(3,2,4)
% plot([1:1:length(errors6)],errors6)
% title('Movement Errors - direction 6')
% grid
% subplot(3,2,5)
% plot([1:1:length(errors7)],errors7)
% title('Movement Errors - direction 7')
% grid
% subplot(3,2,6)
% plot([1:1:length(errors8)],errors8)
% title('Movement Errors - direction 8')
% grid

mean2 = mean(errors2);
mean3 = mean(errors3);
mean4 = mean(errors4);
mean6 = mean(errors6);
mean7 = mean(errors7);
mean8 = mean(errors8);

errorVector = errors-mean(errors);
gainVector = gains-mean(gains);
CI = (2/sqrt(length(errors)))*ones(1,81);
[C,LAGS] = xcorr(gainVector,gainVector,40);
G0 = max(C);
C = C/G0;
% figure
% plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
% title('Overall Gain/Gain Correlation')
% grid
[C,LAGS] = xcorr(errorVector,errorVector,40);
E0 = max(C);
C = C/E0;
figure;subplot(2,1,1)
plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
title('Overall Error/Error Correlation')
grid
[C,LAGS] = xcorr(errorVector,gainVector,40);
C = C/(sqrt(G0*E0));
subplot(2,1,2)
plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
title('Overall Error/Gain Correlation')
grid


errorVector = errors2-mean(errors2);
gainVector = gains2-mean(gains2);
CI = (2/sqrt(length(errors2)))*ones(1,81);
[C,LAGS] = xcorr(gainVector,gainVector,40);
G0 = max(C);
C = C/G0;
% figure
% plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
% title('Gain/Gain Correlation 2')
% grid
[C,LAGS] = xcorr(errorVector,errorVector,40);
E0 = max(C);
C = C/E0;
figure;subplot(2,1,1)
plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
title('Error/Error Correlation 2')
grid
[C,LAGS] = xcorr(errorVector,gainVector,40);
C = C/(sqrt(G0*E0));
subplot(2,1,2)
plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
title('Error/Gain Correlation 2')
grid

errorVector = errors3-mean(errors3);
gainVector = gains3-mean(gains3);
CI = (2/sqrt(length(errors3)))*ones(1,81);
[C,LAGS] = xcorr(gainVector,gainVector,40);
G0 = max(C);
C = C/G0;
% figure
% plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
% title('Gain/Gain Correlation 3')
% grid
```

```
[C,LAGS] = xcorr(errorVector,errorVector,40);
E0 = max(C);
C = C/E0;
figure;subplot(2,1,1)
plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
title('Error/Error Correlation 3')
grid
[C,LAGS] = xcorr(errorVector,gainVector,40);
C = C/(sqrt(G0*E0));
subplot(2,1,2)
plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
title('Error/Gain Correlation 3')
grid

errorVector = errors4-mean(errors4);
gainVector = gains4-mean(gains4);
CI = (2/sqrt(length(errors4)))*ones(1,81);
[C,LAGS] = xcorr(gainVector,gainVector,40);
G0 = max(C);
C = C/G0;
% figure
% plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
% title('Gain/Gain Correlation 4')
% grid
[C,LAGS] = xcorr(errorVector,errorVector,40);
E0 = max(C);
C = C/E0;
figure;subplot(2,1,1)
plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
title('Error/Error Correlation 4')
grid
[C,LAGS] = xcorr(errorVector,gainVector,40);
C = C/(sqrt(G0*E0));
subplot(2,1,2)
plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
title('Error/Gain Correlation 4')
grid

errorVector = errors6-mean(errors6);
gainVector = gains6-mean(gains6);
CI = (2/sqrt(length(errors6)))*ones(1,81);
[C,LAGS] = xcorr(gainVector,gainVector,40);
G0 = max(C);
C = C/G0;
% figure
% plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
% title('Gain/Gain Correlation 6')
% grid
[C,LAGS] = xcorr(errorVector,errorVector,40);
E0 = max(C);
C = C/E0;
figure;subplot(2,1,1)
plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
title('Error/Error Correlation 6')
grid
[C,LAGS] = xcorr(errorVector,gainVector,40);
C = C/(sqrt(G0*E0));
subplot(2,1,2)
plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
title('Error/Gain Correlation 6')
grid

errorVector = errors7-mean(errors7);
gainVector = gains7-mean(gains7);
CI = (2/sqrt(length(errors7)))*ones(1,81);
[C,LAGS] = xcorr(gainVector,gainVector,40);
G0 = max(C);
C = C/G0;
% figure
% plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
% title('Gain/Gain Correlation 7')
% grid
[C,LAGS] = xcorr(errorVector,errorVector,40);
E0 = max(C);
C = C/E0;
figure;subplot(2,1,1)
plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
title('Error/Error Correlation 7')
grid
[C,LAGS] = xcorr(errorVector,gainVector,40);
C = C/(sqrt(G0*E0));
subplot(2,1,2)
plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
title('Error/Gain Correlation 7')
grid

errorVector = errors8-mean(errors8);
gainVector = gains8-mean(gains8);
CI = (2/sqrt(length(errors8)))*ones(1,81);
[C,LAGS] = xcorr(gainVector,gainVector,40);
G0 = max(C);
C = C/G0;
% figure
% plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
% title('Gain/Gain Correlation 8')
% grid
[C,LAGS] = xcorr(errorVector,errorVector,40);
E0 = max(C);
C = C/E0;
figure;subplot(2,1,1)
plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
title('Error/Error Correlation 8')
grid
[C,LAGS] = xcorr(errorVector,gainVector,40);
C = C/(sqrt(G0*E0));
subplot(2,1,2)
plot(LAGS,C,'b-',LAGS,CI,'r-',LAGS,-CI,'r-')
title('Error/Gain Correlation 8')
grid
```