CONTROL EXPERIMENTS IN PLANAR MANIPULATION AND GRASPING

by

Richard M. Murray

Memorandum No. UCB/ERL M89/3

17 January 1989
(Revised 22 August 1990)

Cover Page

CONTROL EXPERIMENTS IN PLANAR MANIPULATION AND GRASPING

by

Richard M. Murray

Memorandum No. UCB/ERL M89/3

17 January 1989
(Revised 22 August 1990)

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

CONTROL EXPERIMENTS IN PLANAR MANIPULATION AND GRASPING

by

Richard M. Murray

Memorandum No. UCB/ERL M89/3

17 January 1989
(Revised 22 August 1990)

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

Many algorithms have been proposed in the literature for control of multi-fingered robot hands. This paper compares the performance of several of these algorithms, as well as some extensions of more conventional manipulator control laws, in the case of planar grasping. A brief introduction to the subject of robot hands and the notation used in this paper is included.

Research supported in part by the Defense Advanced Research Projects Agency under grant N00039-88-C-0292 and the National Science Foundation under grant DMC84-51129.

Contents

Notation

θ	vector of joint positions of the fingers
X_o	position and orientation of an object in a grasp
X_c	vector of positions of the fingers (contacts) of a hand
f_c	combined forces of all fingers
f_e	external forces applied to an object
f_I	internal force applied to an object by the environment
f_N	internal (null) force applied to an object by the fingers grasping it
F_o	force and torque applied to an object by the fingers grasping it
τ	vector of joint torques applied to the fingers
FC	set of forces that make up a friction cone
$\overset{\circ}{FC}$	interior of friction cone
G	grasp map for a hand
$\mathcal{N}(G)$	null space of the grasp matrix
K_i	forward kinematic map for the i^{th} finger
J_h	combined Jacobian for all the fingers in a hand
$M(\theta)$	combined inertia matrix for all the fingers in a hand
M_o	inertia matrix of an object
$N(\theta,\dot{\theta})$	friction and gravity forces
$C(\theta,\dot{\theta})\dot{\theta}$	Coriolis and centrifugal forces

Chapter 1

Introduction

Traditionally robot manipulators interact with their workspace by picking up objects in a gripper attached to the end of the manipulator. Fine positioning of the object is achieved by using the entire manipulator to orient the object held in the gripper. Multi-fingered robot hands can be used to increase the fine motion capabilities of a robot manipulator—instead of using the large motors of the robot manipulator for all positioning tasks, a hand is mounted at the end of the manipulator and the task is divided into the gross motion of the manipulator and the fine motion of the hand (and the object contained in its grasp). The multiple degrees-of-freedom of the hand allow grasps to be selected that are appropriate for the type of task to be performed, and, like a human hand, a robot hand can accurately perform small manipulations on a wide variety of objects.

A great deal of research has focused on the kinematic analysis of hands and the generation of stable grasps. Much of the early work dealt with specific examples of grasping (for example, [Oka82]). Salisbury presented a more generalized approach to grasping which has been widely accepted for the analysis of grasps in which slipping does not occur. He described a grasp in terms of the forces that can be exerted on a object by a finger, or equivalently, the directions of motions constrained by the finger. An important extension to that work was performed by Kerr [Ker84], who considered grasps in which the fingers were allowed to roll along the object. The problem of choosing stable grasps given the object and finger descriptions has also been extensively studied. A good treatment of this subject is given by Nguyen [Ngu86].

More recently the problem of generating feedback control laws for coordinated manipulation of an object being grasped by a robot hand has been investigated. Li, Hsu

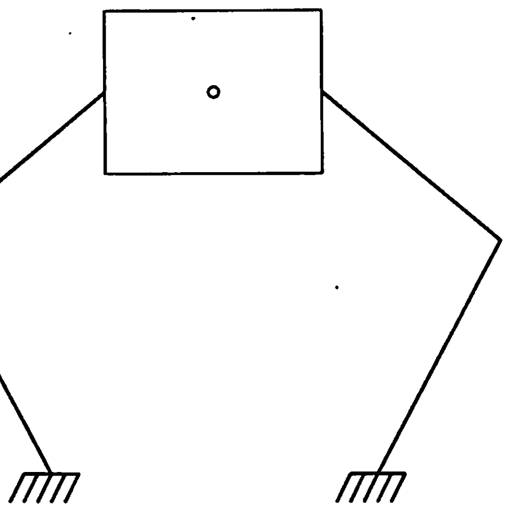

Figure 1.1: A two-fingered hand grasping a box

and Sastry [LHS88] have derived a generalization of the computed torque control law [Bej74] for multi-fingered hands with rigid contacts. This has been further extended by Cole, Hauser and Sastry [CHS88] to consider rolling contact types (as defined by Kerr). A related area of interest is the control of multiple robots performing a single coordinated task ([ZL85], [Hay86], [TBY86], [NNY87]). Generalizations of multi-manipulator robot systems which consider hands as a special case have also been recently formulated ([Hsu88], [MHS89]).

There are several articulated hands that have been developed to study problems in grasping and manipulation. Many of these hands, such as the Utah/MIT hand [JWBI86], and the Stanford/JPL hand [Sal82], are quite complex and require sophisticated computer architectures to control them ([VD87], [SNH*86], [CFAB86]). Unfortunately, this combination of complex kinematics and control architecture makes the implementation of proposed control algorithms correspondingly complex. Operating environments with multiple processors capable of interprocess communication must be developed. Other robot hands, such as the NYU hand [DLSS88], use stepper motors as joint actuators which makes some control schemes more difficult to implement. As a result of these obstacles, few experimental results of the algorithms proposed by researchers can be found in the literature. In particular, comparisons between different hand control algorithms are not currently available.

To provide a facility for experimental verification of control algorithms, we have built a very simple two-fingered planar hand. A diagram of this hand is shown in figure 1.1. The hand uses direct drive DC motors, which eliminates potential problems with gear backlash and friction, and is interfaced to an IBM PC/AT, which can be programmed for real-time control with relatively little overhead. Due to the simplicity of its design, the implementation of a control algorithm is simplified and the performance of the hand can be studied more quickly and easily. Additionally, an intuitive understanding of the structure of a control law can often be reached.

This paper compares the performance of several control schemes implemented on this planar two-fingered hand. The paper is organized as follows: chapter 2 provides an introduction to hand kinematics and derives the dynamic equations which describe the motion of an object contained in a grasp. Using these dynamics, chapter 3 analyzes several existing hand control algorithms and presents extensions of some single robot control algorithms. In order to allow the control laws to be used in more complicated environments, we present the algorithms for the general case of a many-fingered hand operating in three dimensions and simplify to the planar case only when necessary. Proofs of stability using Lyapunov functions are given. Chapter 4 contains experimental comparisons of the algorithms on a two-fingered hand executing a simple positioning task. The control algorithm sources and the dynamic and kinematic equations for the hand are contained in the appendices.

Chapter 2

Grasping fundamentals

The following section provides a brief introduction to grasping and the notations used in this paper. For a more complete discussion of the kinematics of grasping see Kerr [Ker84]. The dynamics outlined here were developed by Li, Hsu and Sastry [LHS88] and Cole, Hauser and Sastry [CHS88] and can be found for more general constrained robot systems in a recent paper by Murray, Hsu and Sastry [MHS89].

2.1 Contact kinematics

A *contact* between a finger and an object is described by a mapping between forces exerted by the finger at the point of contact and the resultant forces at some reference point on the object (e.g., the center of mass). We represent the force exerted by the i^{th} finger as $f_{c_i} \in \mathbf{R}^{n_i}$ where n_i is the dimension of the range of forces that can be applied by the finger. The contact map is a function $G_i : \mathbf{R}^{n_i} \to \mathbf{R}^6$,

$$F_o = \begin{bmatrix} f_o \\ \tau_o \end{bmatrix} = G_i(f_{c_i}) \tag{2.1}$$

A contact can also be described as reducing the number of degrees of freedom of an object. Normally an object has 3 translational and 3 rotational degrees of freedom. A contact prevents motion in certain directions and thus reduces the degrees of freedom.

Several simple contact models are used to classify common fingertip configurations. A *point contact* is obtained when there is no friction between the fingertip and the object. In this case, forces can only be applied in the direction normal to the surface of the object

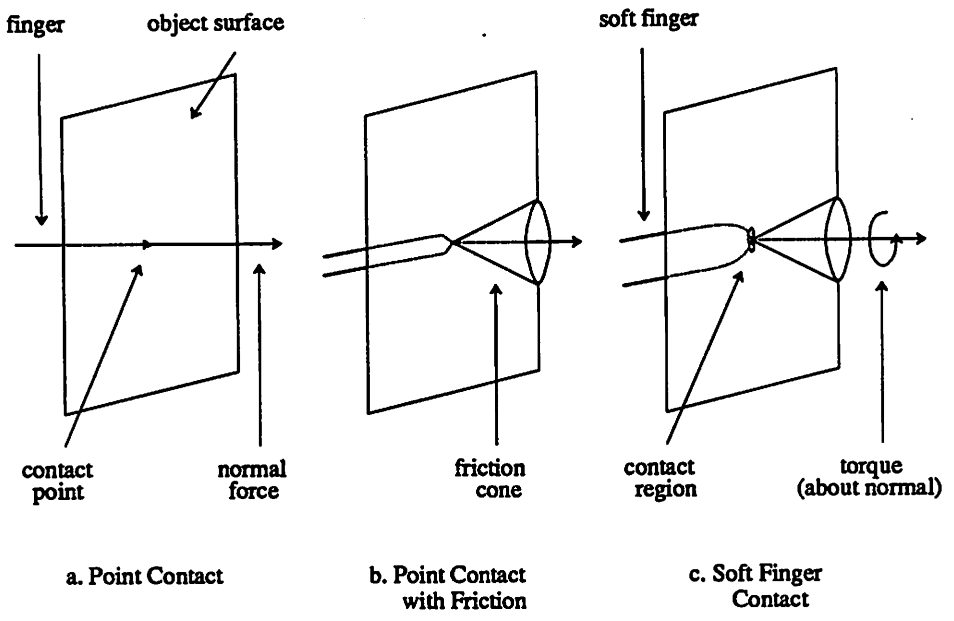

Figure 2.1: Contact types

and hence $n_i = 1$. Point contacts reduce the number of degrees of freedom of the object by 1.

A *point contact with friction* model assumes that friction exists between the fingertip and the object, in which case forces can be exerted in any direction that is within a cone of forces about the direction of the surface normal. This cone, called the *friction cone* is determined by the coefficient of friction (we shall define the friction cone more precisely in the next section). Figure 2.1b shows a point contact with friction and the resultant friction cone. This model assumes that rotational forces cannot be applied (i.e., there is no rotational friction about the surface normal). For planar grasping operations, a point contact with friction reduces the number of degrees of freedom of the object by 2.

A more realistic contact model is the *soft finger* contact. Here we assume that we can not only apply forces in a cone about the surface normal but that we can also apply torques about that normal (see figure 2.1c). These torques are limited by the torsional friction coefficient. Soft finger contacts are only useful in three dimensional grasping, where they reduce the number of degrees of freedom by 4 (one for each direction and rotation about the surface normal). In a planar grasp we do not allow rotation out of the plane, so the soft finger contact can be treated as if it were a point contact with friction.

Many other types of contacts are possible. An enumeration of contact types is given by Salisbury [Sal82], and Kerr [Ker84] extends this list to include rolling contacts.

If we consider only motions in which the finger always maintains contact with the object (a condition which we shall enforce in our control laws) then the contact map for each model can be represented as a simple linear map derived from the contact type and the location of the object reference point relative to the contact reference point. Thus we can represent the map as a matrix which is a function of the contact location and its orientation.

If we have several fingers contacting an object then the net force on the object is the sum of the forces due to each finger. The *grasp map*, G, is the map between finger forces and the resultant object force. Since each contact map is linear and forces can be superposed, we can compose the individual contact maps to form G:

$$f_o = \begin{bmatrix} G_1 & \cdots & G_k \end{bmatrix} \begin{bmatrix} f_{c_1} \\ \vdots \\ f_{c_k} \end{bmatrix} = G f_c, \qquad \begin{array}{l} f_o \in \mathbb{R}^6 \\ f_c \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \times \ldots \times \mathbb{R}^{n_k} \end{array} \tag{2.2}$$

The grasp map is a function of the position and orientation of the object as well as that of the fingertips. When we wish to make this dependence explicit we will write the grasp map as $G(X)$ where X is a vector consisting of the relevant position variables.

The null space of the grasp map corresponds to finger forces which cause no net force to be exerted on the object. We call the force on the object resulting from finger forces which lie in the null space of G, denoted $\mathcal{N}(G)$, *internal* or *null forces.* It is these internal forces which allow us to grip an object.

The velocity of the contact points can be related to the velocity of the object using the principle of virtual work. Since the work done by a hand must be independent of the coordinates in which we measure force and displacement, we can equate the virtual work done in the object frame of reference with that done in the finger frame. If we apply a force f_o and torque τ_o on the object then the virtual work done due to a virtual displacement ∂x_o is simply:

$$\begin{bmatrix} f_o \\ \tau_o \end{bmatrix} \partial x_o = f_c \partial x_c \tag{2.3}$$

where f_c and ∂x_c are the finger forces and virtual displacements, respectively. Using the definition of the grasp matrix we have

$$F_o{}^T \partial x_o = (G f_c)^T \partial x_o = f_c{}^T \partial x_c \tag{2.4}$$

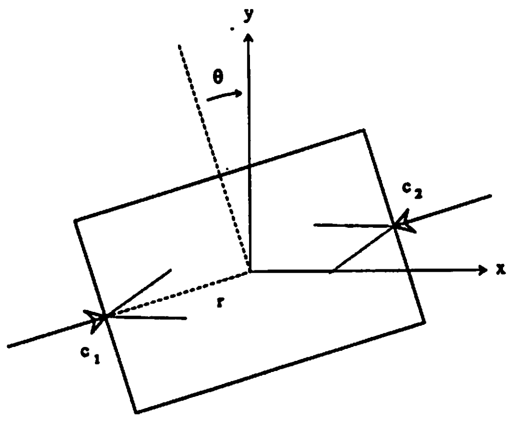

Figure 2.2: Planar two fingered grasp

This holds for arbitrary f_c. Therefore we may write

$$G^T \partial x_o = \partial x_c \tag{2.5}$$

and it follows that

$$\dot{x}_c = G^T \dot{x}_o. \tag{2.6}$$

Example

Consider a simple two-fingered planar hand as shown in figure 2.2. Since we are in the plane, the grasp matrix maps finger forces into x and y forces, and a torque perpendicular to the xy plane. If we assume that the contacts are point contacts with friction, then the grasp map for figure 2.2 is

$$G = \underbrace{\left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \\ -r\sin(\phi) & r\cos(\phi) \end{array}\right.}_{G_1}\underbrace{\left.\begin{array}{cc} 1 & 0 \\ 0 & 1 \\ r\sin(\phi) & -r\cos(\phi) \end{array}\right]}_{G_2} \tag{2.7}$$

where all forces are measured with respect to the xy coordinates shown in the figure.

Equation 2.7 shows that x and y forces from the fingers cause the same x and y forces to be exerted on the object as well as a torque that is dependent on the orientation of the object. The null space of this map is spanned by the vector

$$\begin{bmatrix} \cos\phi \\ \sin\phi \\ -\cos\phi \\ -\sin\phi \end{bmatrix} \tag{2.8}$$

which corresponds to forces applied along the line connecting the two fingertips. Finger forces applied along this line will cause no net force on the object.

2.2 Grasp stability

One measure of the stability of a grasp is the range of forces that can be exerted by the hand on the object. The fingers can only apply unidirectional forces and all forces must lie within the friction cone for the contact. We say a grasp on an object satisfies *force closure* if we can exert, through a set of contacts, arbitrary forces and torques on the object [Ngu86]. More formally, we define the set FC as

$$FC = \left\{ f_c \in \mathbb{R}^n : \|f^t_{c_{ij}}\| \le \mu_{ij} \|f^n_{c_i}\|, \quad i = 1, \ldots, k, \quad j = 1, \ldots, n_i \right\} \tag{2.9}$$

where $f^t_{c_{ij}}$ is the tangent component of the j^{th} element of f_{c_i}, $f^n_{c_i}$ is the normal force for the i^{th} contact and μ_{ij} is the coefficient of friction corresponding to $f_{c_{ij}}$. For soft finger and rolling contacts, the torques exerted by the fingers also satisfy equation 2.9 with $f^t_{c_{ij}}$ replaced by the torque (i.e., we do not want to apply a torque which is greater than the torsional friction coefficient multiplied by the magnitude of the normal force). The force closure condition can now be stated mathematically as

$$\text{force closure} \Leftrightarrow G(FC) = \mathbb{R}^6 \tag{2.10}$$

which says that the range of the grasp map over forces lying in the friction cone covers $\mathbb{R}^6$ (the space of forces and torques applied to the object). One property of a force closure grasp is that G must have full row rank. If this were not true then there would be some object force which could not be produced by the fingers (a contradiction).

It is also useful to define the concept of prehensility. A grasp is *prehensile* if there exists a force contained in the null space of the grasp map which also lies in the *interior* of the friction cone. More formally,

$$\text{prehensility} \Leftrightarrow \mathcal{N}(G) \cap \overset{\circ}{FC} \neq \emptyset \tag{2.11}$$

where $\overset{\circ}{FC}$ is the set of forces lying completely within the friction cone (i.e., $\|f^t_{c_{ij}}\| < \mu_{ij}\|f^n_{c_i}\|$). We shall require this property in order to insure that our controllers can maintain a grip on an object while manipulating it. For a more rigorous development of these concepts see [CHS88].

2.3 Finger Kinematics

Up to this point we have assumed that our fingers are point forces in space. In fact, we are more interested in considering fingers which are kinematic mechanisms. For each finger i we associate a forward kinematic map $K_i : \mathbb{R}^{m_i} \to \mathbb{R}^6$ which takes joint position to end effector position and orientation. The Jacobian of the forward kinematic map relates joint velocities to the end effector velocities,

$$\dot{x}_{c_i} = \frac{\partial K_i}{\partial \theta_i}\frac{d\theta}{dt} = J_i(\theta_i)\dot{\theta}_i, \qquad \theta_i \in \mathbb{R}^{m_i},\ x_i \in \mathbb{R}^6 \tag{2.12}$$

Since each finger can be controlled individually, we can stack these matrices to get the forward kinematic map for the hand,

$$\begin{bmatrix} x_{c_1} \\ \vdots \\ x_{c_k} \end{bmatrix} = \begin{bmatrix} K_1(\theta_1) \\ \vdots \\ K_k(\theta_k) \end{bmatrix} \tag{2.13}$$

Taking the derivative of this map about a point θ,

$$\dot{x}_c = \begin{bmatrix} \dot{x}_{c_1} \\ \vdots \\ \dot{x}_{c_k} \end{bmatrix} = \begin{bmatrix} \frac{\partial K_1(\theta_1)}{\partial \theta_1} & & 0 \\ & \ddots & \\ 0 & & \frac{\partial K_k(\theta_k)}{\partial \theta_1} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \vdots \\ \dot{\theta}_k \end{bmatrix} \tag{2.14}$$

We shall call the Jacobian of the forward kinematic map the *hand Jacobian*, J_h,

$$\dot{x}_c = J_h(\theta)\dot{\theta} \tag{2.15}$$

Again we can apply the principle of virtual work to relate the forces at the contact points, f_c, to the individual joint torques, $\tau \in \mathbb{R}^{m_1} \times \mathbb{R}^{m_2} \times \ldots \times \mathbb{R}^{m_k}$

$$\tau = J_h^T(\theta) f_c \tag{2.16}$$

The force and velocity transformations for the entire hand are summarized in figure 2.3 below.

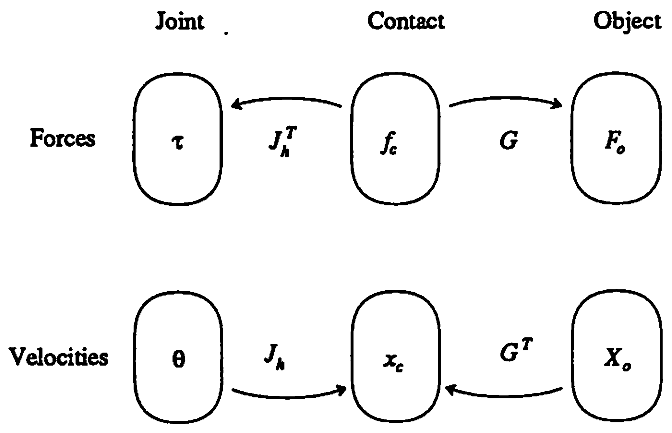

Figure 2.3: Grasping transformations

Example

Consider the two-fingered planar hand shown in figure 2.4. The kinematics for each finger are

$$\begin{bmatrix} x_{c_i} \\ y_{c_i} \end{bmatrix} = \begin{bmatrix} l_{i1}\sin\theta_{i1} + l_{i2}\sin(\theta_{i1}+\theta_{i2}) \pm b/2 \\ l_{i1}\cos\theta_{i1} + l_{i2}\cos(\theta_{i1}+\theta_{i2}) \end{bmatrix} \tag{2.17}$$

Stacking the kinematic maps and solving for the Jacobian we get

$$\begin{bmatrix} \dot{x}_{c_1} \\ \dot{y}_{c_1} \\ \dot{x}_{c_2} \\ \dot{y}_{c_2} \end{bmatrix} = \begin{bmatrix} l_{l1}c_{l1} + l_{l2}c_{l12} & l_{l2}c_{l12} & 0 & 0 \\ -l_{l1}s_{l1} - l_{l2}s_{l12} & -l_{l2}s_{l12} & 0 & 0 \\ 0 & 0 & l_{r1}c_{r1} + l_{r2}c_{r12} & l_{r2}c_{r12} \\ 0 & 0 & -l_{r1}s_{r2} - l_{r2}s_{r12} & -l_{r2}s_{r12} \end{bmatrix} \begin{bmatrix} \dot{\theta}_{11} \\ \dot{\theta}_{12} \\ \dot{\theta}_{21} \\ \dot{\theta}_{22} \end{bmatrix} \tag{2.18}$$

where c_{l1} denotes $\cos(\theta_{l1})$, c_{l12} denotes $\cos(\theta_{l1}+\theta_{l2})$ and the sine terms are represented in the same manner.

2.4 Hand Dynamics

The dynamics of a robot manipulator and, in particular, a single finger of a hand can be represented as a differential equation with respect to the joint angles, θ_i,

$$M_i(\theta_i)\ddot{\theta}_i + C_i(\theta_i,\dot{\theta}_i)\dot{\theta}_i + N_i(\theta_i,\dot{\theta}_i) = \tau_i - J_i^T f_{c_i} \tag{2.19}$$

where $M_i(\theta_i) \in \mathbb{R}^{n_i \times n_i}$ is the symmetric moment of inertia matrix for the i^{th} finger, $C_i(\theta_i,\dot{\theta}_i)\dot{\theta}_i \in \mathbb{R}^{n_i}$ is a vector of Coriolis and centrifugal terms, $N_i(\theta_i,\dot{\theta}_i) \in \mathbb{R}^{n_i}$ is a vector of gravity and friction forces and $\tau \in \mathbb{R}_{n_i}$ is the vector of applied joint torques. The

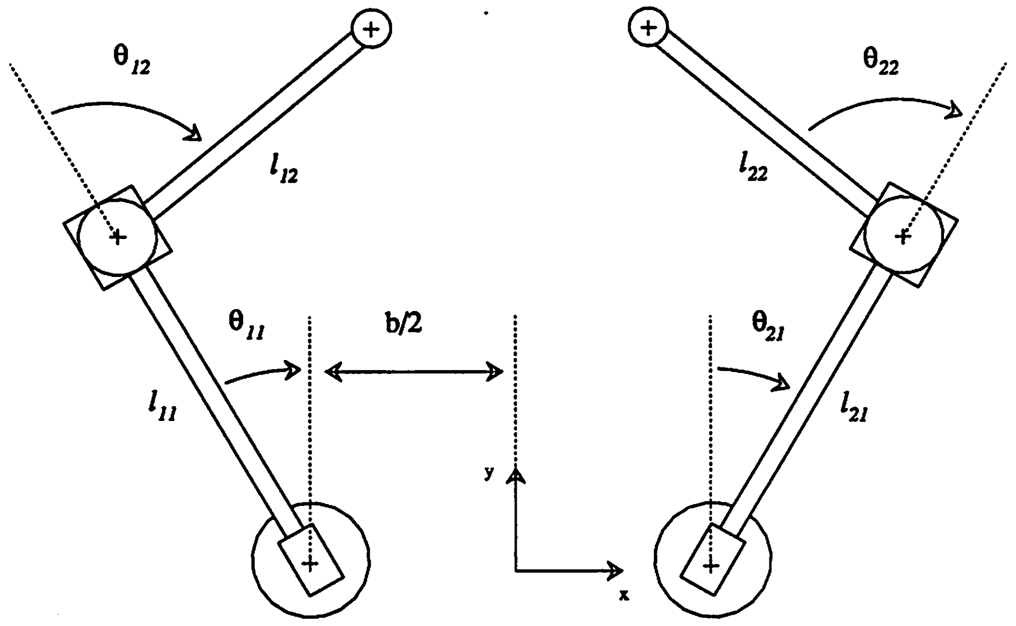

Figure 2.4: STYX - simple two-fingered hand

final term in equation 2.19 is the torque due to the force applied at the fingertip. It is the addition of this term that causes coupling between the fingers (due to the object being grasped). We also note that with proper definition of $C_i(\theta_i, \dot{\theta}_i)$ the matrix $\dot{M}_i - 2C_i$ is skew-symmetric (see [Hsu88] or [OS88]). This property is important in proving stability for some of the controllers presented in the next section.

Stacking the equations for all the fingers in the hand we can write the hand dynamics,

$$\begin{bmatrix} M_1(\theta_1) & & 0 \\ & \ddots & \\ 0 & & M_k(\theta_k) \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \vdots \\ \ddot{\theta}_k \end{bmatrix} + \begin{bmatrix} C_i(\theta_1, \dot{\theta}_1) & & 0 \\ & \ddots & \\ 0 & & C_k(\theta_k, \dot{\theta}_k) \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \vdots \\ \dot{\theta}_k \end{bmatrix} +$$
$$\begin{bmatrix} N_1(\theta_1, \dot{\theta}_1) \\ \vdots \\ N_k(\theta_k, \dot{\theta}_k) \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \vdots \\ \tau_k \end{bmatrix} - \begin{bmatrix} J_1^T(\theta_1) & & 0 \\ & \ddots & \\ 0 & & J_k^T(\theta_k) \end{bmatrix} \begin{bmatrix} f_{c_1} \\ \vdots \\ f_{c_k} \end{bmatrix} \tag{2.20}$$

or more simply

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + N(\theta, \dot{\theta}) = \tau - J_h^T f_c \tag{2.21}$$

and it follows directly that $\dot{M} - 2C$ is skew-symmetric.

The dynamics of the object are governed by the Newton-Euler equations. Expressed in the base (inertial) frame, these equations can be written in terms of the object position, x_o, and angular velocity, ω_o, with respect to the center of mass

$$\begin{bmatrix} m_o I & 0 \\ 0 & I_b \end{bmatrix} \begin{bmatrix} \ddot{x}_o \\ \dot{\omega}_o \end{bmatrix} + \begin{bmatrix} 0 \\ \omega_o \times I_b \omega_o \end{bmatrix} = \begin{bmatrix} f_o \\ \tau_o \end{bmatrix} \tag{2.22}$$

where $m_o I \in \mathbf{R}^{3\times 3}$ is the object mass matrix, and $I_b \in \mathbf{R}^{3\times 3}$ is the object inertia matrix. Note that the inertia matrix is a function of object orientation and can be written as $I_b = R_o I_o R_o^T$ where R_o is the rotation matrix between the base coordinate frame and a coordinate frame affixed to the object.

If we parameterize the orientation of the object by a vector $\phi \in \mathbf{R}^3$ (e.g., roll-pitch-yaw angles or Euler angles) then there exists a linear transformation $P(\phi)$ such that

$$\omega_o = P(\phi)\dot{\phi} \tag{2.23}$$

(see [CHS88]). We also note that we can represent the cross product operation as

$$\omega \times y = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} y \tag{2.24}$$

and hence $(\omega\times)$ can be represented as a 3×3 skew-symmetric matrix. We rewrite equation 2.22 as

$$M_o \ddot{X}_o + C_o(X_o, \dot{X}_o)\dot{X}_o = F_o = G f_c + f_e \tag{2.25}$$

where

$$\begin{aligned}
M_o &= \begin{bmatrix} m_o I & 0 \\ 0 & P^T R_o I_o R_o^T P \end{bmatrix} \\
C_o &= \begin{bmatrix} 0 & 0 \\ 0 & \begin{matrix} P^T R_o I_o R_o^T \dot{P} + \\ P^T (P\dot{\phi}\times)(R_o I_o R_o^T P) \end{matrix} \end{bmatrix} \\
X_o &= \begin{bmatrix} x_o \\ \phi \end{bmatrix} \\
F_o &= \begin{bmatrix} f_o \\ P^T \tau_o \end{bmatrix}
\end{aligned}$$

The vector $P^T\tau_o$ in F_o is the vector of torques expressed in the ϕ coordinate frame. The second equality in equation 2.25 follows from equation 2.2 (assuming the grasp uses the same parameterization for orientation) and f_e is due to external forces applied to the object.

With this definition of M_o and C_o the matrix $\dot{M}_o - 2C_o$ has the form:

$$\dot{M}_o - 2C_o = \begin{bmatrix} 0 & 0 \\ 0 & \begin{matrix} P^T\dot{R}_oI_oR_o^TP + P^TR_oI_o\dot{R}_o^TP + \dot{P}^TR_oI_oR_o^TP \\ -P^TR_oI_oR_o^T\dot{P} - 2P^T(P\dot{\phi}\times)(R_oI_oR_o^TP) \end{matrix} \end{bmatrix} \tag{2.26}$$

and if we use the identity $\dot{R}_o = \omega_o \times R_o = (P\dot{\phi} \times R_o)$ we see that

$$\dot{M}_o - 2C_o = \begin{bmatrix} 0 & 0 \\ 0 & \begin{matrix} -P^T\dot{R}_oI_oR_o^TP + P^TR_oI_o\dot{R}_o^TP \\ +\dot{P}^TR_oI_oR_o^TP - P^TR_oI_oR_o^T\dot{P} \end{matrix} \end{bmatrix} \tag{2.27}$$

is skew symmetric.

Equations 2.21 and 2.25 represent the dynamics of a hand grasping an object. The only additional equation needed is the specification of the contact force, f_c. In general this force depends upon the characteristics of the fingertip and the compliances of the hand and object. For our purposes, we can assume that all bodies are rigid and that the contacts are never broken. In this case the fingertips are constrained to move at the same speed as the contact points on the object. This constraint can be written as

$$J_h(\theta)\dot{\theta} = G^T(X_o, X_c)\dot{X}_o \tag{2.28}$$

where $J_h\dot{\theta}$ is the vector of fingertip velocities and $G^T\dot{X}_o$ is the velocity of the contact points on the object.

To actually solve for the contact force, we differentiate 2.28 to obtain

$$\dot{J}_h\dot{\theta} + J_h\ddot{\theta} = \dot{G}^T\dot{X}_o + G^T\ddot{X}_o \tag{2.29}$$

We can now use equation 2.29 to eliminate f_c from equations 2.21 and 2.25 and derive the complete hand dynamics. Solving equation 2.25 for f_c we have

$$f_c = G^+\Big(M_o\ddot{X}_o + C(X_o, \dot{X}_o)\dot{X}_o) + f_e\Big) + f_I \tag{2.30}$$

where $G^+ = G^T(GG^T)^{-1}$ is the least squares generalized right inverse of G and $f_I \in \mathcal{N}(G)$. Substituting this into equation 2.21 we get

$$M(\theta)\ddot{\theta} + C(\theta,\dot{\theta})\dot{\theta} + N(\theta,\dot{\theta}) = \tau - J_h^TG^+\Big(M_o\ddot{X}_o + C(X_o,\dot{X}_o)\dot{X}_o) + f_e\Big) - J_h^Tf_I \tag{2.31}$$

Now we can use the constraint relation (2.29) to eliminate $\ddot{\theta}$ and it follows that if the hand is not in a singular position then

$$\ddot{\theta} = J_h^{-1}\left(G^T\ddot{X}_o + \dot{G}^T\dot{X}_o - \dot{J}_h\dot{\theta}\right) \tag{2.32}$$

and

$$\begin{aligned}\left[M(\theta)J_h^{-1}G^T + J_h^T G^+ M_o\right]\ddot{X}_o - M(\theta)J_h^{-1}\left(\dot{J}_h\dot{\theta} - \dot{G}^T\dot{X}_o\right)&\\ +C(\theta,\dot{\theta})\dot{\theta} + N(\theta,\dot{\theta}) + J_h^T G^+\left(C_o(X_o,\dot{X}_o)\dot{X}_o + f_e\right) + J_h^T f_I &= \tau\end{aligned} \tag{2.33}$$

This can be rewritten as

$$M_h(X_o)\ddot{X}_o + C_h(X_o,\dot{X}_o)\dot{X}_o + N_h(X_o,\dot{X}_o) = GJ_h^{-T}\tau - f_e \tag{2.34}$$

where

$$\begin{aligned}M_h &= GJ_h^{-T}M(\theta)J_h^{-1}G^T + M_o\\ C_h &= C_o(X_o,\dot{X}_o) + GJ_h^{-T}\left(C(\theta,\dot{\theta})J_h^{-1}G^T + M(\theta)\frac{d}{dt}\left(J_h^{-1}G^T\right)\right)\\ N_h &= GJ_h^{-T}N(\theta,\dot{\theta})\end{aligned}$$

and $\dot{M}_h - 2C_h$ is skew-symmetric (the proof is algebraic and similar to that given for the object dynamics). This is the description of the hand dynamics in object coordinates. We have assumed here that the finger coordinates, θ, are derivable from the object position, X_o—this holds for point contact models, but can fail for rolling contacts in three dimensions. See [CHS88] for a more thorough discussion.

In the case of planar grasping, the object dynamics are somewhat simplified since the object is only allowed to rotate about the axis perpendicular to the plane of motion. If we represent the position and orientation of the object as (x, y, ϕ) and the inertia of the object as $I_o \in \mathbb{R}$, we have

$$\begin{bmatrix} m_o & 0 & 0\\ 0 & m_o & 0\\ 0 & 0 & I_o\end{bmatrix}\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{\phi}\end{bmatrix} = \begin{bmatrix} f_x\\ f_y\\ \tau_\phi\end{bmatrix} \tag{2.35}$$

and hence $C_o = 0$. Furthermore, in three dimensions a parameterization of $SO(3)$ (the space of rigid body rotations) must be selected and this adds some complexity to the dynamics. Since there is only one axis of rotation in the planar case, the representation of the orientation of the object is particularly simple.

Chapter 3

Control Algorithms for Grasping

3.1 The grasping control problem

The grasping control problem can be broken into two parts

1. Tracking – the center of mass of the object should follow a specified trajectory.

2. Holding – the finger forces should lie within the friction cone at all times.

Condition 2 is important not only because we do not wish to lose our grip on the object, but also because we assumed in our derivation of the grasp dynamics that contact was maintained. Without this constraint we would have to specify the dynamics of contact.

If a grasp is prehensile it can be shown that given an arbitrary set of finger forces, f_c, we can find an internal force, $f_N \in \mathcal{N}(G)$, such that the combined force $f_c + f_N$ is inside the friction cone (see section 3.6). Thus, given a force generated to solve the tracking problem, we can always add a force to this such that condition 2 is satisfied. Since internal forces cause no net motion of the hand or object, this additional force does not affect the net force exerted by the fingers on the object. We shall assume in the sequel that such an internal force is available at all times. Section 3.6 discusses the choice of this force in more detail.

To satisfy the tracking problem, we will examine several different algorithms. Each of the algorithms makes different assumptions about the grasp dynamics and all of them assume that the grasp is prehensile and satisfies force closure.

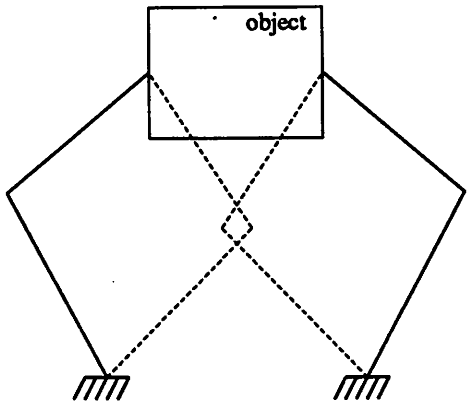

Figure 3.1: Non-unique solutions of the inverse kinematics for a planar two-fingered hand

3.2 Individual joint control

The first algorithm we will study is based on the idea that we can specify the trajectory of the object by transforming that trajectory into the space of the joint variables and then controlling the fingers individually. In using this approach we assume that the dynamics of the object can be neglected and we concern ourselves only with the finger dynamics. Thus, we model the system as

$$M(\theta) + C(\theta,\dot{\theta})\dot{\theta} + N(\theta,\dot{\theta}) = \tau \tag{3.1}$$

We are given the desired joint trajectory, θ_d, and its acceleration, $\ddot{\theta}_d = G^T J^{-1}\ddot{X}_d + \frac{d}{dt}(G^T J^{-1})\dot{X}_d$, which is calculated using the inverse kinematic map between the object location and the joint positions. In general this map is not unique, but often in practice it is not hard to choose between solutions. For example, of the four possible solutions shown for a two-fingered hand in figure 3.1, only one is desirable since the other solutions intersect the object.

To control each individual joint we use a computed torque control law ([Bej74], [LWP80]).

$$\tau = M(\theta)\Big(\ddot{\theta}_d + K_v\dot{e}_\theta + K_p e_\theta\Big) + C(\theta,\dot{\theta})\dot{\theta} + N(\theta,\dot{\theta}) \tag{3.2}$$

where K_v and K_p are positive definite matrices and $e_\theta = \theta_d - \theta$. Our error dynamics become

$$M(\theta)\Big(\ddot{e}_\theta + K_v\dot{e}_\theta + K_p e_\theta\Big) = 0 \tag{3.3}$$

We can now choose K_v and K_p so that the error dynamics are exponentially stable at the origin, $e_\theta = 0$.

We must also add a null force term to grip the object. Since the null force term does not cause any motion in the object, it does not affect the dynamics of the links and our tracking stability remains unchanged. If we are given a null force term f_N, then we can apply this force by adding a joint torque of $J_h^T f_N$ (the null force applied at the finger tips, reflected back to the joints). The final control law is then

$$\tau = M(\theta)\Big(\ddot{\theta}_d + K_v\dot{e}_\theta + K_p e_\theta\Big) + C(\theta,\dot{\theta})\dot{\theta} + N(\theta,\dot{\theta}) + J_h^T f_N \tag{3.4}$$

There are several potential advantages to this type of approach. Since the control law introduces no coupling between the fingers, we can use a separate processor for each finger. Furthermore, if the mass matrix is diagonal, then the control law in equation 3.4 is also diagonal and we can control each *joint* with its own controller. While it is rare that manipulator dynamics are truly diagonal, it is often a good approximation. This approximation is particularly useful since many commercial motor controllers are capable of performing PD control laws for a single joint. Even if the control equation has coupling terms, the amount of calculation required to compute the motor torque can be reduced to no more than $2\sum n_i^2$ multiplications (assuming the nonlinear terms can be ignored). This requires precomputing $M\ddot{\theta}_d$, MK_v and MK_p (remembering that M, K_v and K_p are block diagonal).

3.3 Force transformation

If we had only the object dynamics to consider (equation 2.25), we could use a computed torque control law in object coordinates having the form

$$F_o = M_o\Big(\ddot{X}_d + K_v\dot{e}_x + K_p e_x\Big) \tag{3.5}$$

This gives us the desired forces (and torque) to be applied to the center of mass of the object. We can find the forces that would have to be applied at the fingertips to get such an object force by premultiplying by $G^+ = G^T(GG^T)^{-1}$, a pseudo-inverse of G. This transforms object forces to finger forces having minimum norm (i.e., zero internal force component). Similarly, we can transform the finger forces to joint torques by premultiplying by J_h^T. Thus to generate an object force as in equation 3.5 we apply torque

$$\tau = J_h^T G^+ M_o\Big(\ddot{X}_d + K_v\dot{e}_x + K_p e_x\Big) \tag{3.6}$$

This algorithm is an example of *coordinated* control. The fingers are now coupled by the control law. In the case of a two-fingered planar hand this means that we control the center and orientation of the line connecting the two fingertips. One consequence of this coupling is that an easy multiprocessor solution is no longer available. We must now communicate more information than just the desired joint trajectories.

This algorithm takes more calculation than the joint control algorithm since typically we must calculate the position of the object given the joint angles before we can apply the control law. This calculation requires sine and cosine calculations (for the fingertip locations) and an arctangent operation (for the orientation of the object). Once the control law is calculated we must multiply by $M(\theta)J_h^{-1}G^T$ which requires at most n^2 multiplications ($M(\theta)J_h^{-1}G^T$ is *not* block diagonal).

To speed up the overall control sample rate we can break the calculation into two pieces. The update loop calculates $M(\theta)J_h^{-1}G^T$ and $J_h f_N$ while the control loop carries out the matrix multiplications and additions. If the trajectory that the object is following is changing slowly, then we find empirically that we can run the update loop more slowly and speed up the control loop.

3.4 Generalized Computed Torque

Since the hand dynamics in equation 2.34 have the same basic form as the dynamics of a simple manipulator, it is straightforward to extend manipulator control laws into hand control laws. One common manipulator control algorithm, which we have already seen for joint control, is the computed torque control law. Applying that formulation to 2.34 we get:

$$\tau = J_h^T G^+ \left[M_h \left(\ddot{X}_d + K_v \dot{e}_x + K_p e_x \right) + C_h(X, \dot{X})\dot{X} + N_h(X, \dot{X}) \right] \tag{3.7}$$

This gives an error equation in object coordinates of

$$M_h \left(\ddot{e}_x + K_v \dot{e}_x + K_p e_x \right) = 0 \tag{3.8}$$

and away from singularities of J_h, our dynamics are governed by

$$\ddot{e}_x + K_v \dot{e}_x + K_p e_x = 0 \tag{3.9}$$

The computational resources required to implement this algorithm are considerable. Not only must we calculate the inertia matrix, but we have to cancel all the nonlinear terms.

These nonlinear terms contain many trigonometric calculations but they can be minimized by the use of carefully constructed lookup tables. This algorithm was originally proposed for multi-fingered hands in [LHS88] and has been extended to the case of rolling contacts in [CHS88].

3.5 Natural and Stiffness Controllers

Natural control was proposed by Koditschek in 1984 [Kod84] as an alternative to computed torque which did not rely on exact cancellation of nonlinear dynamics. It relies on the skew-symmetric property of the robot dynamics, $\alpha^T\left(\dot{M} - 2C\right)\alpha = 0$ for all $\alpha \in R^n$. It is an extension of the simpler PD control law and it can be shown to be asymptotically stable for the tracking control problem. The natural control law has the form

$$\tau = J_h^T G^+ \left[M_h \ddot{X}_d + C_h(X, \dot{X})\dot{X}_d + K_v \dot{e}_x + K_p \dot{e}_x + N_h(X, \dot{X})\right] \tag{3.10}$$

where K_p and K_v are again positive definite gain matrices.

Stiffness control is a slight extension of natural control that has the additional property that the error can be shown to approach zero *exponentially*. Forms of this control law can be found in [Sad87] and [SL87]. The control law presented here is a slight generalization of the laws proposed by others, but similar forms can be found in the literature [WB88]. The generalization leads to a more complicated proof of stability but allows more control over the resulting stiffness. We use the control law

$$f = M_h\left(\ddot{X}_o + \lambda\dot{e}\right) + C_h(X_o, \dot{X}_o)\left(\dot{X}_o + \lambda e\right) + N_h(X_o, \dot{X}_o) + K_p e + K_v \dot{e} \tag{3.11}$$

where $\lambda > 0$ and K_p, K_v are positive definite. The closed loop system becomes

$$M_h(\ddot{e} + \lambda\dot{e}) + C_h(X_o, \dot{X}_o)\left(\dot{X}_o + \lambda e\right) + K_p e + K_v \dot{e} = 0 \tag{3.12}$$

To prove exponential stability of this nonlinear system we use the Lyapunov candidate

$$V = \frac{1}{2}(\dot{e} + \lambda e)^T M_h(\dot{e} + \lambda e) + \frac{1}{2}e^T K_p e + \frac{1}{2}\lambda_1 e^T K_v e \tag{3.13}$$

and it is easy to show that

$$\sigma_1 \|(e, \dot{e})\| \leq \|V(e, \dot{e})\| \leq \sigma_2 \|(e, \dot{e})\| \tag{3.14}$$

where σ_1 and σ_2 are positive constants related to the singular values of M_h, K_p and K_v. With this choice of V we can calculate $\dot{V}$ as

$$\begin{aligned}
\dot{V} &= (\dot{e}+\lambda e)^T M_h(\ddot{e}+\lambda\dot{e}) && (3.15)\\
&\quad +\frac{1}{2}(\dot{e}+\lambda e)^T\dot{M}_h(\dot{e}+\lambda e)+\dot{e}^T K_p e+\lambda e^T K_v\dot{e}\\
&= (\dot{e}+\lambda e)^T\Big(-C_h(X_o,\dot{X}_o)(\dot{e}+\lambda e)-K_v\dot{e}-K_p e\Big)\\
&\quad +\frac{1}{2}(\dot{e}+\lambda e)^T\dot{M}_h(\dot{e}+\lambda e)+\dot{e}^T K_p e+\lambda e^T K_v\dot{e} && (3.16)
\end{aligned}$$

Using the fact that the matrix $\dot{M}_h - 2C_h$ is skew-symmetric and canceling terms,

$$\dot{V} = -\dot{e}^T K_v\dot{e} - \lambda_1 e^T K_p e \tag{3.17}$$

Since $\dot{V}$ is quadratic in e and $\dot{e}$, it follows that $e \to 0$ exponentially.

The complexity of these algorithms is roughly the same as the computed torque control law.

3.6 Choosing the grasp force, f_N

All of the algorithms have relied on the choice of a grasping force, $f_N \in \mathcal{N}(G)$ which maintains contact between the fingertips and the object by insuring that the finger forces lie in the friction cone. There are several possible methods for calculating this term. Since ideally f_N does not affect the force applied to the center of mass of the object we should be free to choose f_N without worrying about its effect on the tracking control problem.

First, we prove that if a grasp is prehensile then we can always find a null force, f_N, such that the total finger force, $f_c + f_I + f_N$, is inside the friction cone. Since f_I is an internal force, it can be included in f_N and so we shall assume it is zero. By the definition of a prehensile grasp there exists $f_N \in \mathcal{N}(G) \cap \overset{\circ}{FC}$ such that

$$\|f^t_{N_{ij}}\| < \mu_i \|f^n_{N_i}\| \tag{3.18}$$

where $f^t_{N_{ij}}$ is the tangent component of f_N projected onto the j^{th} force direction of the i^{th} contact and f_{N_i} is the normal component of f_N at the i^{th} contact point. It is important to note that f_{N_i} is nonzero for each i and therefore by increasing f_N, we always increase the normal component of the force exerted at each contact with respect to the tangential forces. Since $\overset{\circ}{FC}$ is defined as the cartesian product of the n friction cones in equation 3.18,

$f_N \in \mathcal{N}(G) \cap \overset{\circ}{FC}$ implies $\alpha f_N \in \mathcal{N}(G) \cap \overset{\circ}{FC}$ for all $\alpha \in \mathbb{R}^+$. Now we can look at the unit vector in the $fc + \alpha f_N$ direction as $\alpha \to \infty$:

$$\lim_{\alpha \to \infty} \frac{f_c + \alpha f_N}{\|f_c + \alpha f_N\|} = f_N \tag{3.19}$$

Since $f_N \in \overset{\circ}{FC}$ it follows that for sufficiently high α, $\frac{f_c + \alpha f_N}{\|f_c + \alpha f_N\|}$ is also in $\overset{\circ}{FC}$ and hence $f_c + \alpha f_N$ is in the interior of the FC. Now from the definition of FC, the individual contact forces must all lie within their respective friction cones simultaneously.

The simplest f_N is a constant f_N. It must be large enough so that finger forces never leave the friction cone over the entire trajectory of the object. Generally this requires a knowledge of the bounds on the external forces that can be exerted on the object. The advantage of this approach is that $J_h^T f_N$ can be calculated at the same rate as J_h—saving computation time.

A more robust f_N could be calculated by looking at the finger forces (these can be derived from the joint torques, τ, using $f_c = (J_h^T)^{-1}\tau$) and finding a null force which causes $f_c + f_N$ to lie in FC. If the grasp map has a simple form, such as the one given in the example in section 2.1, a basis for the null space can be used to construct the set of all valid f_N. This calculation takes more time but may be necessary in the case of large uncertainties.

Other grasp force calculations are discussed in [LHS88] but all of these algorithms share some fundamental problems. One difficulty is that in a real-world hand the maximum motor torques that can be generated are finite. Thus, we are not guaranteed that we can apply an f_N which satisfies $f_c + f_N \in FC$ without saturating the motors. Another issue is the effect of the null force term in the presence of errors. If a large internal force term is used and, due to sensor or actuator errors, it does not actually lie in the null space of the grasp matrix, the resulting force can cause positioning errors and in the extreme case, instability. Chapter 5 discusses some research issues related to the solution of these problems.

Chapter 4

Experimental comparison of algorithms

The algorithms presented in the previous chapter have been implemented on a multi-fingered hand known as Styx. Styx is a planar two-fingered hand built at the University of California, Berkeley to test different multi-fingered hand control algorithms. A labeled diagram of Styx is shown in figure 4.1.

The motors used in Styx are direct drive DC motors mounted at the base of each joint and are driven with a pulse width modulated 20 kHz square wave. Since the second motor is mounted at the end of the first joint, the dynamics of the first joint are relatively independent of the configuration of the robot (i.e., $M(\theta)$ can be considered to constant). Each motor contains a quadrature encoder which is used to sense position. The resolution of

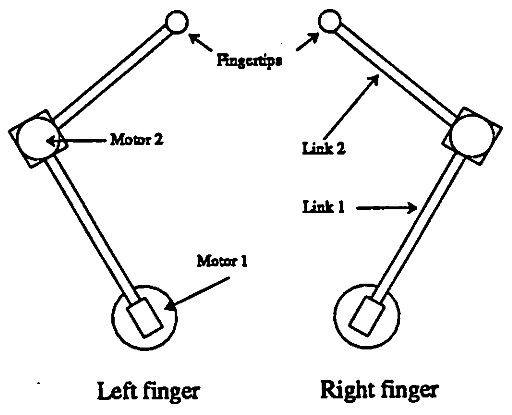

Figure 4.1: Top view of styx

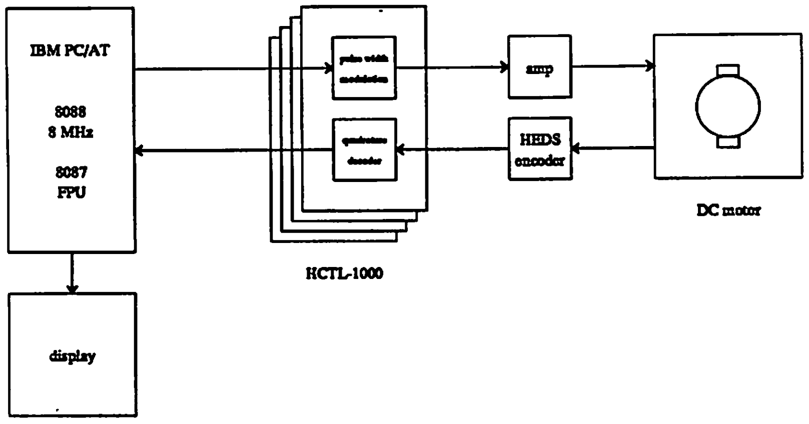

Figure 4.2: Styx control hardware

this encoder is 500 lines/revolution, which generates 2000 edges (or counts) per 360 degree rotation.

Styx is connected to an IBM PC/AT running at 8.0 MHz with an 8087 floating point coprocessor. The motors and encoders are interfaced to the AT using a set of four HP HCTL-1000 motion control chips interfaced to the AT bus. The HCTL chips generate a pulse width modulated signal which is fed to an amplifier. The quadrature signal from the position encoders is connected directly to the chip inputs. Figure 4.2 shows a block diagram of the Styx control hardware.

The software to drive Styx is composed of an assembly language scheduler which controls the sample rate of the inner (control) and outer (update) loops. The algorithms themselves are written in the C programming language and compiled using the Microsoft 5.1 Optimizing C compiler. Control rates of 120 Hz have been achieved for the more complicated controllers by careful use of table lookups and coding. More information on the technical details of Styx is available in Appendix A.

All of the algorithms implemented on Styx made some basic simplifying assumptions:

1. Motor dynamics can be ignored – for small velocities the torque generated by a motor is proportional to the input pulse width. Figure 4.3 shows the stall torque versus pulse width for a single motor on Styx. Additionally, friction terms were left out of the dynamic model (i.e., $N(\theta, \dot{\theta}) = 0$).

2. Jacobian and mass matrices change slowly – since the trajectories commanded were

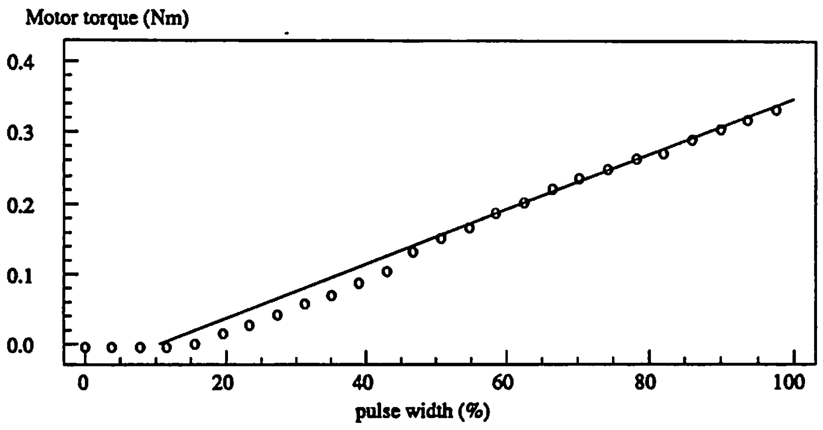

Figure 4.3: Static motor torque versus pulse width

slow relative to the control rate, the Jacobian and mass matrices did not need to be recalculated in the control loop. The update loop, which runs at a slower rate, was used instead. For Styx, the Jacobian requires a much more CPU intensive calculation than the basic control law (which is similar for all controllers) so that the update rate is the limiting factor in the speed of an algorithm.

3. Fingers can be modelled as point contacts – the actual finger tips used on Styx were rubber circles. To avoid the added computational complexity required to model the rolling contacts, the fingertips were modelled as simpler point contacts. For Styx this meant that the center of mass of the object shifted slightly as the object moved through its trajectory. For the trajectories given here, the orientation of the object, ϕ, was near zero so this effect was minimized.

Several different trajectories were traversed with each control law. A single circular trajectory is presented here to conserve space and to allow easy visual interpretation of the results. The trajectory shown in figure 4.4 is a circle with a diameter of 5 cm and a period of 0.5 Hz. This circle is too fast for most of the controllers to track accurately but was chosen to emphasize sources of error in the controllers.

Since we are comparing different control algorithms we must decide on what criterion to judge an algorithm. Although the final test of an algorithm should be how well it satisfies a set of design criteria, other comparisons can be useful to get a feel for the relative strengths of an algorithm. Three types of comparisons are presented here.

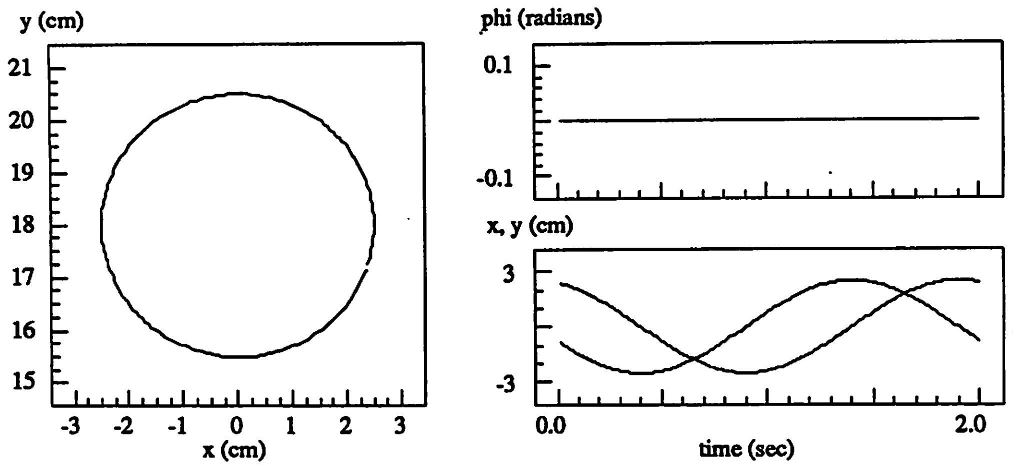

Figure 4.4: Desired trajectory

4.1 Fixed Gain comparisons

The *fixed gain* comparison uses a fixed K_p and K_v for each of the algorithms. K_p and K_v were chosen by selecting the cutoff frequency, ω_n, and the damping factor, ζ, for the closed loop system using any of the computed torque based schemes (which give a linear error equation). Given these two values, we set $K_p = \omega_n^2 I$ and $K_v = 2\zeta\omega_n I$. For all of the experiments presented in this section, ω_n was chosen as 2.5 Hz, or one tenth of the (Jacobian) update frequency. This value of ω_n was used so that noise introduced into the system by the update loop would be attenuated by 40 dB. ζ was chosen as 0.5 to provide fast transient response (using a critical damping factor, $\zeta = 1$, gave sluggish response). The fixed gain comparison is really a measure of how closely the system model matches the actual system—we would expect that the most complicated model would yield the best controller.

4.1.1 Joint control

The joint control algorithm proved to be very sensitive to the radius of the object being grasped. Since the inverse kinematic solution requires knowledge of the object radius, errors in this radius cause the desired joint position to be wrong. If the modeled radius is too small we get a constant joint error and this joint error results in additional internal force and object position errors (due to nonlinearity of the hand kinematic map). Likewise, if the modeled object radius is too large then the internal force term causes a constant error in

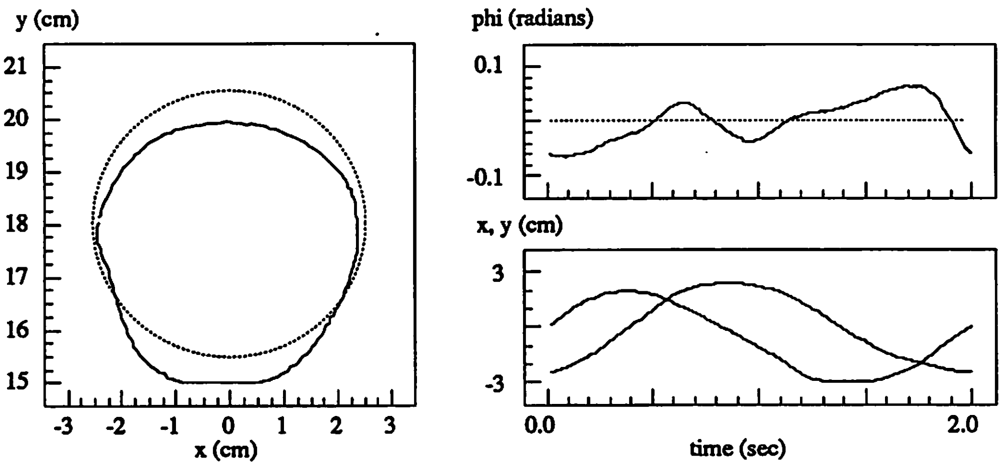

Figure 4.5: Joint control algorithm

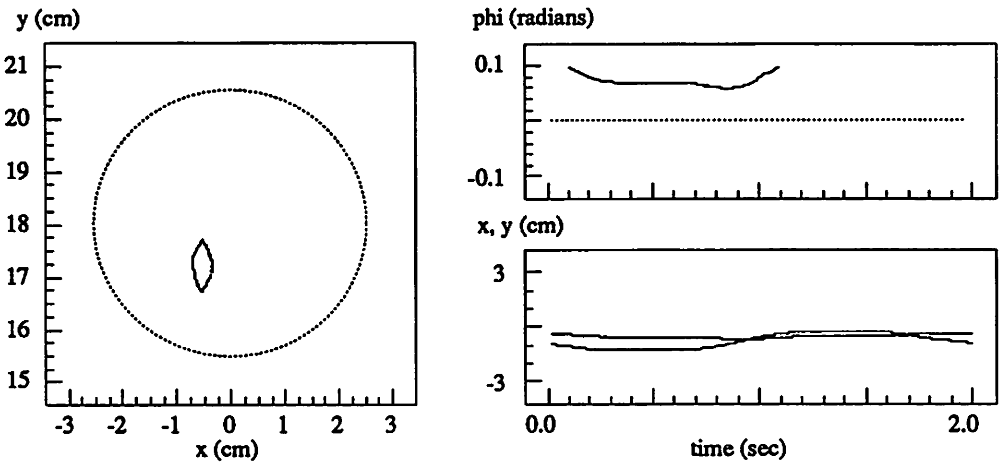

Figure 4.6: Force transformation algorithm

the joint positions. In fact, if the controller gains are large enough the PD control law can override the constant internal grasping force and cause contact to be broken. The constant displacement seen in figure 4.5 is due to unintentionally setting the object radius slightly too large (this same radius was used by all of the algorithms).

Figure 4.5 also shows that the orientation error for the joint control algorithm is very sensitive. Because a small change in orientation produces a very small change in the joint position (as compared to an error in object position) the joint control algorithm is not very effective at controlling the orientation. Increasing the gain in the joints (which requires an increase in controller rate) can help overcome this problem.

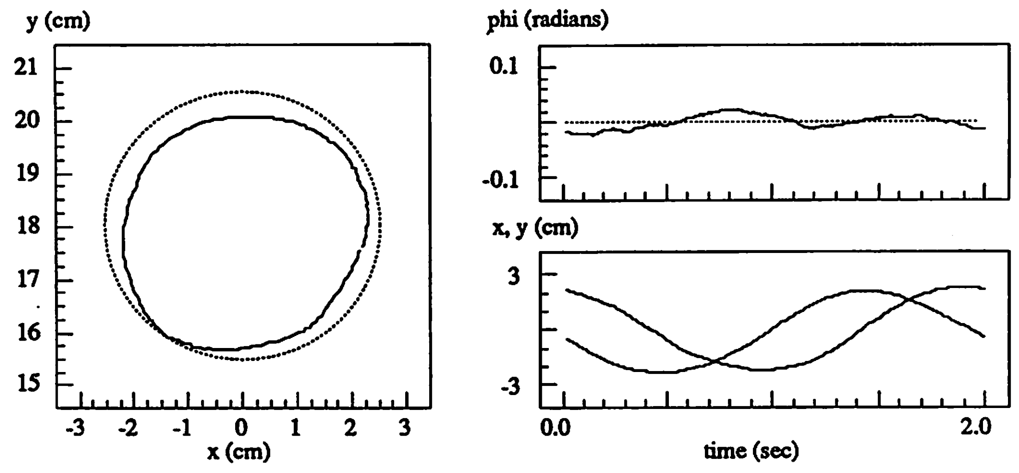

Figure 4.7: Computed torque algorithm

4.1.2 Force transformation

For Styx, the mass of the fingers is much heavier than the mass of the object and so we do not expect an algorithm which ignores the finger dynamics to perform well. Figure 4.6 confirms our intuition. Commanding torques which are sufficient to move only the object produces large errors due to the mass of the fingers.

4.1.3 Computed torque

The performance of the computed torque algorithm is the best of any of the algorithms presented in this section (see figure 4.7). The position error is comparable to that of the joint control algorithm but the orientation error is much lower. Also the computed torque algorithm is insensitive to errors in the object size—the controller is effectively controlling the position of the line connecting the fingertips and simultaneously pushing in along that line. For very small objects, the orientation becomes very sensitive to the fingertip positions and the performance degrades somewhat.

4.1.4 Natural and stiffness control

The natural and stiffness controllers do not give a linear error equation and hence our design criterion for choosing K_p and K_v does not apply. In fact this is one of the problems with these algorithms—there is no simple method for choosing the gains. The proof of stability gives general conditions for convergence ($K_p > 0$, $K_v > 0$) but does not

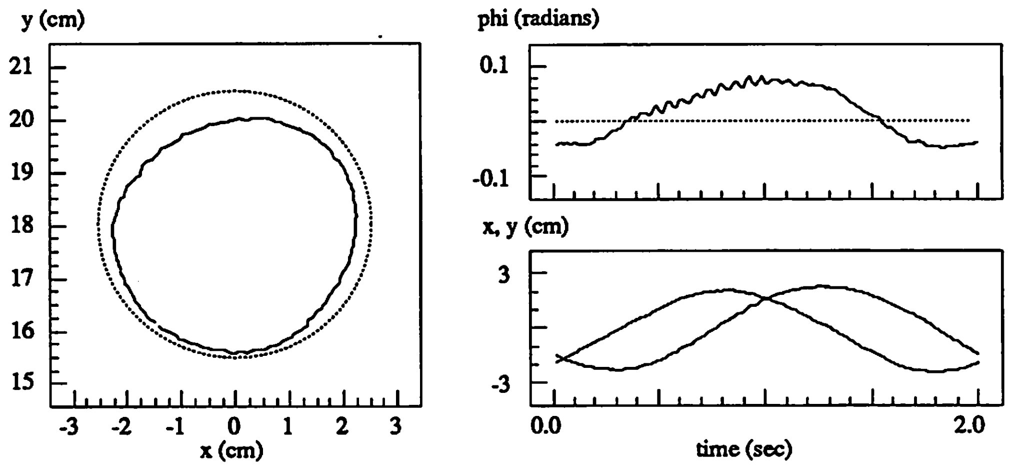

Figure 4.8: Natural control law – equivalent gains

provide a good indication of the performance to be expected. If we execute the algorithms with the gains used in this section we find that the grasped object moves very slightly (much less than with the force transformation controller shown in figure 4.6).

4.2 Equivalent gain comparisons

In order to compensate for the low gains of the natural and stiffness controllers one might consider using $M_h K_p$ and $M_h K_v$ as gains. This would then give a control law very similar to computed torque and we might expect equivalent performance. Unfortunately, the stability analysis for the natural and stiffness controllers requires that the gain matrices be constant and M_h is a function of finger and object position. Furthermore, $M_h > 0$ and $K_p > 0$ does not imply $M_h K_p > 0$. An approximate solution is available. Since K_p and K_v are diagonal (by choice) we can examine the diagonal entries of M_h at some nominal position and use these entries to scale K_p and K_v. In the case of a constant diagonal inertia matrix this would then give us a control law similar to computed torque. We term this type of comparison an *equivalent gain* comparison since it attempts to compensate for differences in the magnitudes of the overall gain matrices.

4.2.1 Natural control

The results of scaling the gain matrices for the natural controller are shown in figure 4.8. We see that the position error is greatly reduced (before scaling the hand barely

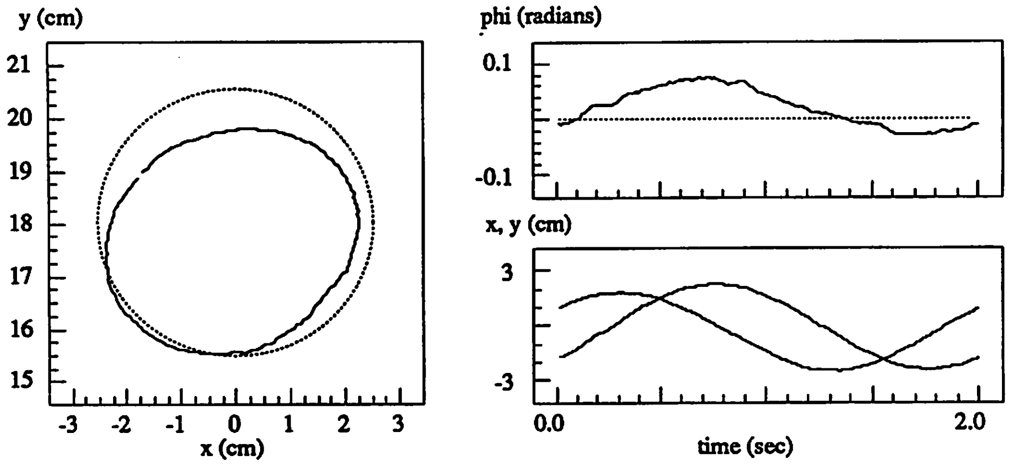

Figure 4.9: Stiffness control law – equivalent gains

moved), but there appear to be oscillations in the orientation (upper right graph). In fact the controller was very marginally stable and any slight disturbance would cause the hand to oscillate. Part of the reason that the controller is so nearly unstable may be the form of the inertia matrix for the hand. For the experiments performed here, the diagonal entries of this matrix were quite large but the inertial coupling between the x position and the orientation was of comparable magnitude.

4.2.2 Stiffness control

The stiffness control law appeared to be slightly more stable (see figure 4.9). The oscillations in orientation are no longer present although there was little improvement in trajectory error. Results of other tests indicate that the natural controller was in fact more stable but still quite underdamped.

4.3 Fixed Hardware comparisons

The final comparison is the *fixed hardware* comparison. Here we allow the gains and control rates to be maximized individually. This is the most realistic comparison since it ranks the controllers in overall effectiveness. It is reasonable to imagine that a simple controller might be able to perform better because it can run at a much higher servo rate—thus compensating for uncertainties more quickly. For each controller the following steps where performed:

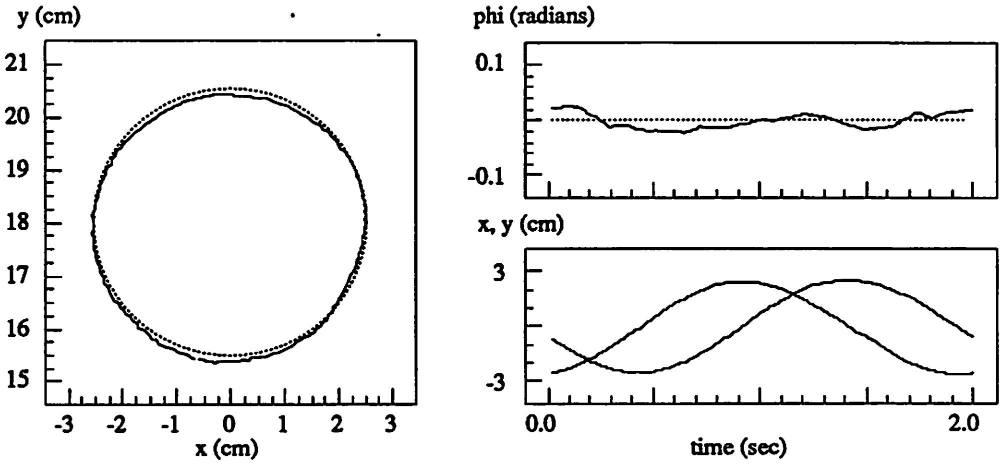

Figure 4.10: Joint control algorithm (update rate = 100 Hz, ω_n = 5 Hz)

1. Maximize controller speed – the update and control rates were chosen as large as possible such that both loops could still finish their calculations in the allotted time.

2. Maximize controller gain – the same method as previously outlined was used: ω_n was chosen as one tenth of the update frequency and ζ was set to 0.5.

Since the control law was linked with the frequency of the update loop and the control loop was running much faster than the dynamics of the system (effectively emulating a continuous time controller), only the update frequency was varied in the experiments presented here. Due to the structure of the control software, only integer multiples of the control period were used for the update period.

4.3.1 Joint control

The simplicity of the joint control algorithm allows a controller with sufficiently high gain and bandwidth to overcome unmodeled disturbances (see figure 4.10). Due to other sources of noise in the system the cutoff frequency for this algorithm was placed at 5 Hz instead of 10 Hz. Note the slight shift in the y position due to the aforementioned error in object radius.

4.3.2 Force transformation

We might expect the force transformation algorithm to experience similar improvements. Due to its moderate complexity, an update rate of 50 Hz was the maximum

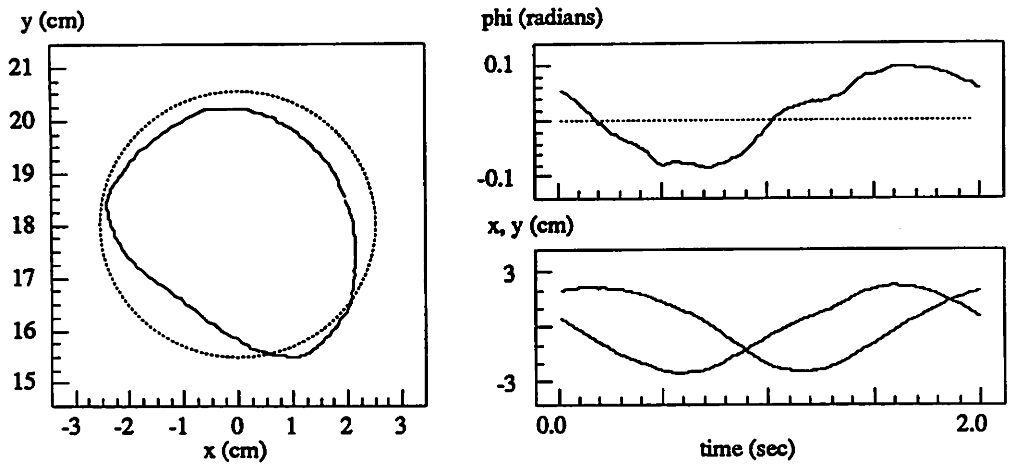

Figure 4.11: Force transformation algorithm without nonlinear terms (50 Hz)

achievable rate (up from 25 Hz). Figure 4.11 shows that the overall performance was still very poor. This is to be expected since the formulation completely ignored the finger dynamics, which were much larger than the object dynamics.

4.3.3 Computed torque

Computing the full set of nonlinear compensating terms allows the computed torque algorithm to be run with an update rate of only 25 Hz (shown in figure 4.7). This limits the frequency response of the controller as well as the DC gain (which is determined by $K_p = \omega_n^2$). A considerable savings in computational complexity can be gained by ignoring the nonlinear terms. Calculations indicate that these terms are small relative to the link/motor inertias. The magnitudes of the various terms for a sample trajectory are shown in figure 4.12. By ignoring these nonlinear terms we can increase ω_n to 3.3 Hz and we see that the controller is able to follow the desired trajectory much more closely (figure 4.13). Notice how small the orientation error is compared to other algorithms.

4.3.4 Natural and stiffness control

The natural and stiffness controllers were of sufficient complexity that it was not possible to increase the update rate past 25 Hz, even by ignoring the nonlinear terms. Therefore the equivalent gain comparisons shown in figures 4.8 and 4.9 where the best results obtained.

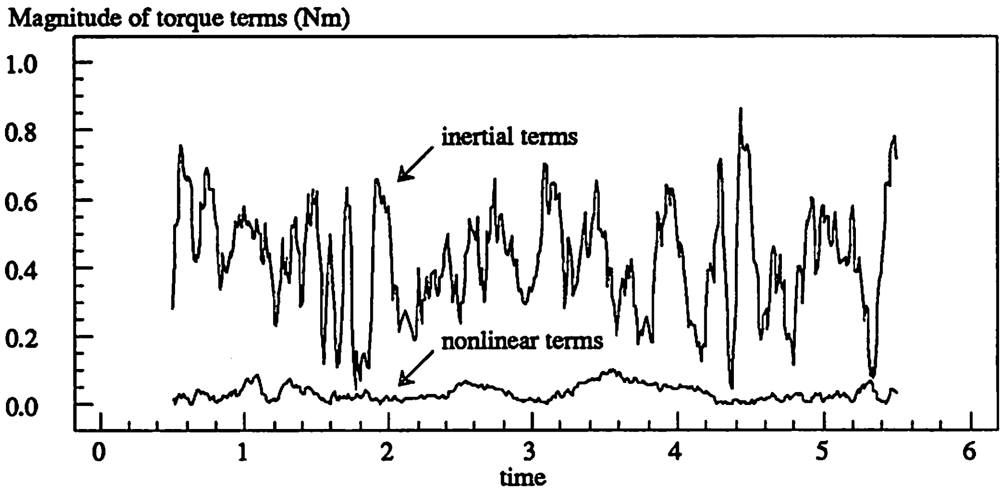

Figure 4.12: Magnitude of torque coefficients for a sample trajectory

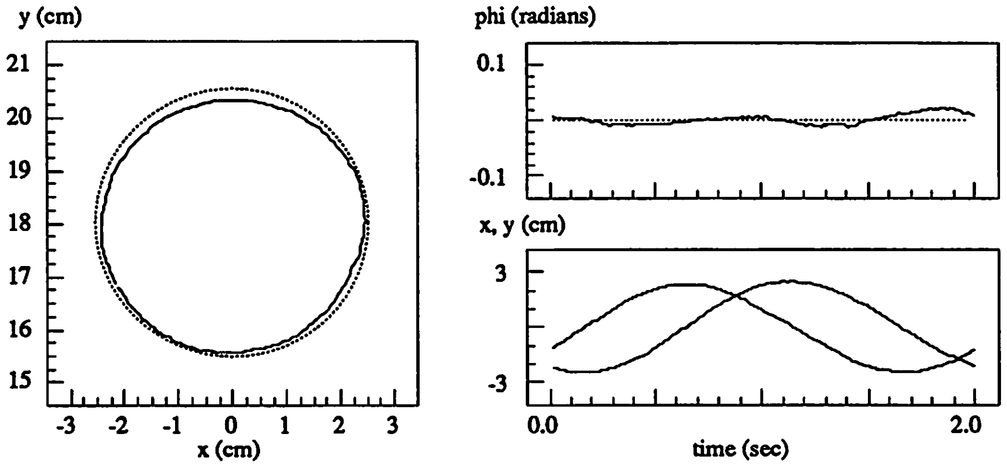

Figure 4.13: Computed torque algorithm without nonlinear terms (33 Hz)

4.4 Conclusions

Based on the experiments performed on Styx, the most effective control laws are the simple joint control law and the generalized computed torque control law. Although the joint control law ignores the interaction between the joints (caused by grasping the object), it can be run at sufficiently high rates (and hence gains) to overcome errors. It has the additional advantage that it is stable even when contact is broken, since the joint controllers are individually stable. The computed torque control law, being a coordinated control law, relies on the fact that contact is not broken. While intermittent breaks don't usually present a problem, the controller does exhibit strange behavior when the object is removed from the fingers' grasp—the fingers move in to the center of the line connecting the fingertips, where the orientation gain becomes very large. Undesirable oscillations result.

The disadvantage of the joint control law is that the error dynamics are not linear and therefore it is difficult to predict the results. If the object being grasped has a large inertia relative to the fingers, the joint controller will experience problems like those of the force transformation controller, resulting in large errors. The performance of the generalized computed torque algorithm will be basically unchanged, since all dynamics were considered in deriving the algorithm and the resulting error system is linear. This makes the computed torque control law an attractive alternative for position control of multi-fingered hands.

Chapter 5

Extensions and discussion

There are many areas in the analysis and control of multi-fingered hands that are not fully understood. Although some of the difficulties encountered with the control of multi-fingered hands can be solved by extending results available for classical robots, there are many problems for which this is not true. These problems stem from the constraints imposed by the grasp and the unidirectionality of these constraints due to the condition that the forces must lie in the friction cone at all times.

One of the areas that needs to be investigated more fully is the choice of the internal force used to grip an object. A common assumption is that the internal force can simply be chosen large enough to keep the net finger forces within the friction cone at each contact. While this is always theoretically possible if the grasp is prehensile, there are practical limits which discourage this simple solution. For example, the motors which are driving the fingers usually have a maximum torque that they can generate and hence the finger forces are limited to some range (which changes with configuration). This limit must be taken into account in designing the control law, the desired trajectory, or both.

Another problem that occurs when choosing internal forces is due to the potential instability of the grasp when measurement errors are present. If we apply a large internal force between the fingertips, and due to errors in measurement or system parameterization these forces do not lie exactly in the null space of the grasp matrix, then these forces cause motion in the object which induces position errors. If these forces are large enough they can even overwhelm the position control law and cause the system to go unstable.

Consider as an example the situation shown in figure 5.1. The position of the

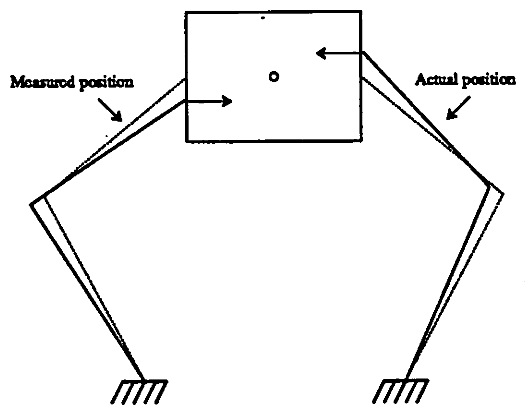

Figure 5.1: A two-fingered grasp with errors in position measurement

fingers as measured by the system is such that internal forces cause no motion of the object (internal forces in this example are applied along the horizontal line connecting the measured contact points). If the fingers do not actually lie on this horizontal line, then the internal force causes a torque to be exerted on the object. This torque will cause the box to rotate and force errors in the object orientation. If the errors are small the box will probably be stable with some fixed error. However, if the commanded internal force is large then the grasp may become unstable.

There are several interesting aspects to this problem. Besides the usual robustness analysis based on the control gains, one can also look at the effect of using different contact types. In this particular case, it can be shown that using disks as the fingers instead of point contacts increases the stability of the grasp. When errors in position measurement occur, the contacts roll in such a way as to counteract the effects of these errors (by bringing the contacts in line with each other). The larger the disk, the smaller the final error. A more thorough and general analysis of this situation would be useful both in the choice of the control gains as well as the selection of a grasp.

Another area of open research is that of a hand grasping an object which is interacting with the environment. The control laws presented here were analyzed as position control laws. In many robot tasks the hand must control both the position of the object and the forces it exerts on the environment. A standard example is a hand grasping a peg which is to be inserted in a hole. One method of performing such an insertion is to control the stiffness of the peg in some directions while controlling the position of the peg in other directions [Whi82]. Control laws which are capable of controlling both position and force (or stiffness) are available for simple robot manipulators (so-called hybrid control laws). Extensions to these methods for more general constrained multi-manipulator systems are needed. Some preliminary work in this area can be found in [MHS89].

Acknowledgements

I would like to thank the members of the robotics lab at U.C. Berkeley, and in particular Ping Hsu, John Hauser, Zexiang Li and Arlene Cole, for many useful suggestions and comments on this research. The construction of Styx and the associated electronics could not have been completed without the help of Kris Pister and Jeff Kerr. Finally, I wish to express my thanks to Shankar Sastry for providing motivation and encouragement throughout the course of this project.

Appendix A

STYX specifications

A.1 Kinematics

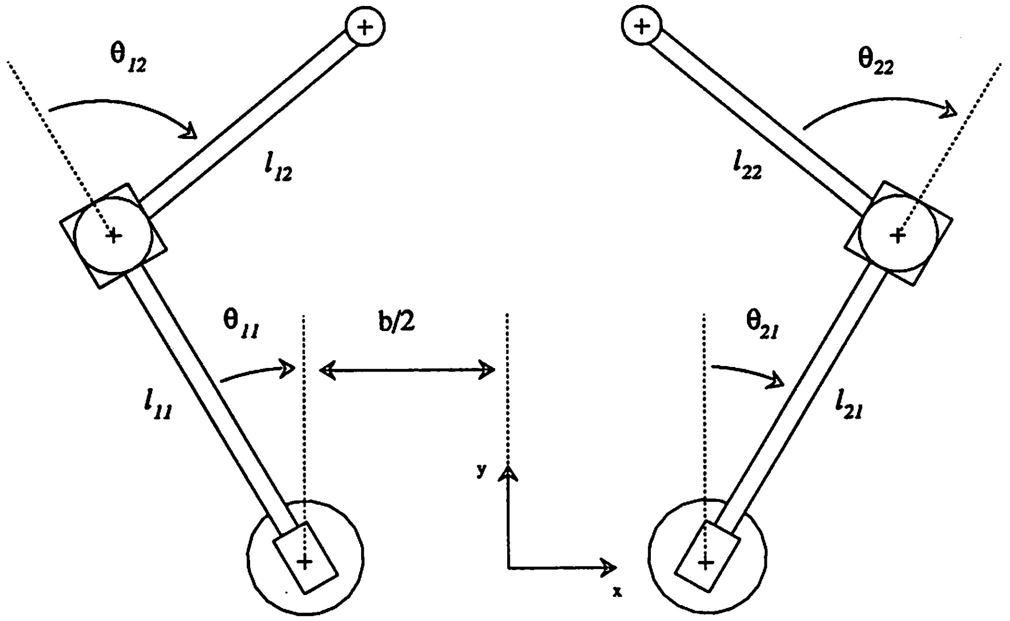

Figure A.1: Kinematic parameters for Styx

Forward kinematic map

$$\begin{bmatrix} x_{c_1} \\ y_{c_1} \\ x_{c_2} \\ y_{c_2} \end{bmatrix} = \begin{bmatrix} l_{l1}\sin\theta_{l1} + l_{l2}\sin(\theta_{l1}+\theta_{l2}) - b/2 \\ l_{l1}\cos\theta_{l1} + l_{l2}\cos(\theta_{l1}+\theta_{l2}) \\ l_{r1}\sin\theta_{r1} + l_{r2}\sin(\theta_{r1}+\theta_{r2}) + b/2 \\ l_{r1}\cos\theta_{r1} + l_{r2}\cos(\theta_{r1}+\theta_{r2}) \end{bmatrix}$$

Hand Jacobian

$$\begin{aligned} J_h(\theta) &= \begin{bmatrix} l_{l1}c_{l1} + l_{l2}c_{l12} & l_{l2}c_{l12} & 0 & 0 \\ -l_{l1}s_{l1} - l_{l2}s_{l12} & -l_{l2}s_{l12} & 0 & 0 \\ 0 & 0 & l_{r1}c_{r1} + l_{r2}c_{r12} & l_{r2}c_{r12} \\ 0 & 0 & -l_{r1}s_{r1} - l_{r2}s_{r12} & -l_{r2}s_{r12} \end{bmatrix} \\ &= \begin{bmatrix} y_l & l_{l2}c_{l12} & 0 & 0 \\ -x_l - b/2 & -l_{l2}s_{l12} & 0 & 0 \\ 0 & 0 & y_r & l_{r2}c_{r12} \\ 0 & 0 & -x_r + b/2 & -l_{r2}s_{r12} \end{bmatrix} \end{aligned}$$

$$J_h^{-1}(\theta) = \begin{bmatrix} -\frac{l_{l2}s_{l12}}{\det(J_l)} & -\frac{l_{l2}c_{l12}}{\det(J_l)} & 0 & 0 \\ \frac{(x_l+b/2)}{\det(J_l)} & \frac{y_l}{\det(J_l)} & 0 & 0 \\ 0 & 0 & -\frac{l_{r2}s_{r12}}{\det(J_r)} & -\frac{l_{r2}c_{r12}}{\det(J_r)} \\ 0 & 0 & \frac{(x_r-b/2)}{\det(J_r)} & \frac{y_r}{\det(J_r)} \end{bmatrix}$$

$$\dot{J}_h(\theta) = \begin{bmatrix} -l_{l1}s_{l1}\dot\theta_{l1} - l_{l2}s_{l12}\dot\theta_{l12} & -l_{l2}s_{l12}\dot\theta_{l12} & 0 & \\ -l_{l1}c_{l1}\dot\theta_{l1} - l_{l2}c_{l12}\dot\theta_{l12} & -l_{l2}c_{l12}\dot\theta_{l12} & 0 & \\ 0 & 0 & -l_{r1}s_{r1}\dot\theta_{r1} - l_{r2}s_{r12}\dot\theta_{r12} & -l_{r2}s_{r12}\dot\theta_{r12} \\ 0 & 0 & -l_{r1}c_{r1}\dot\theta_{r1} - l_{r2}c_{r12}\dot\theta_{r12} & -l_{r2}c_{r12}\dot\theta_{r12} \end{bmatrix}$$

$$\det(J_l) = -y_l\, l_{l2}s_{l12} + (x_l + b/2)l_{l2}c_{l12}$$

$$\det(J_r) = -y_r l_{r2}s_{r12} + (x_r + b/2)l_{r2}c_{r12}$$

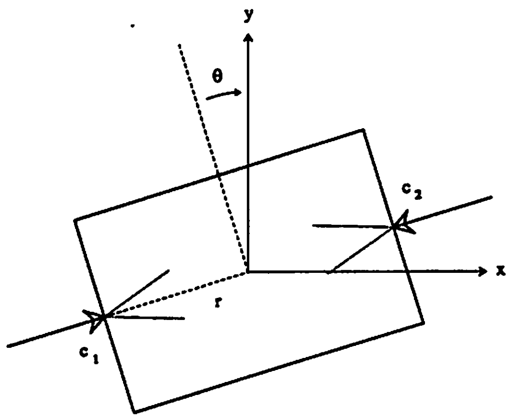

Figure A.2: Two-fingered grasp of a box

Grasp kinematics

The grasp map for two fingers grasping a box (shown above) is

$$G = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -r\sin(\phi) & r\cos(\phi) \end{bmatrix}}_{G_1} \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ r\sin(\phi) & -r\cos(\phi) \end{bmatrix}}_{G_2} \tag{A.1}$$

(all forces are measured with respect to the xy coordinates shown in the figure). The null space of this map is spanned by the vector

$$\begin{bmatrix} \cos\phi \\ \sin\phi \\ -\cos\phi \\ -\sin\phi \end{bmatrix} \tag{A.2}$$

A.2 Dynamics

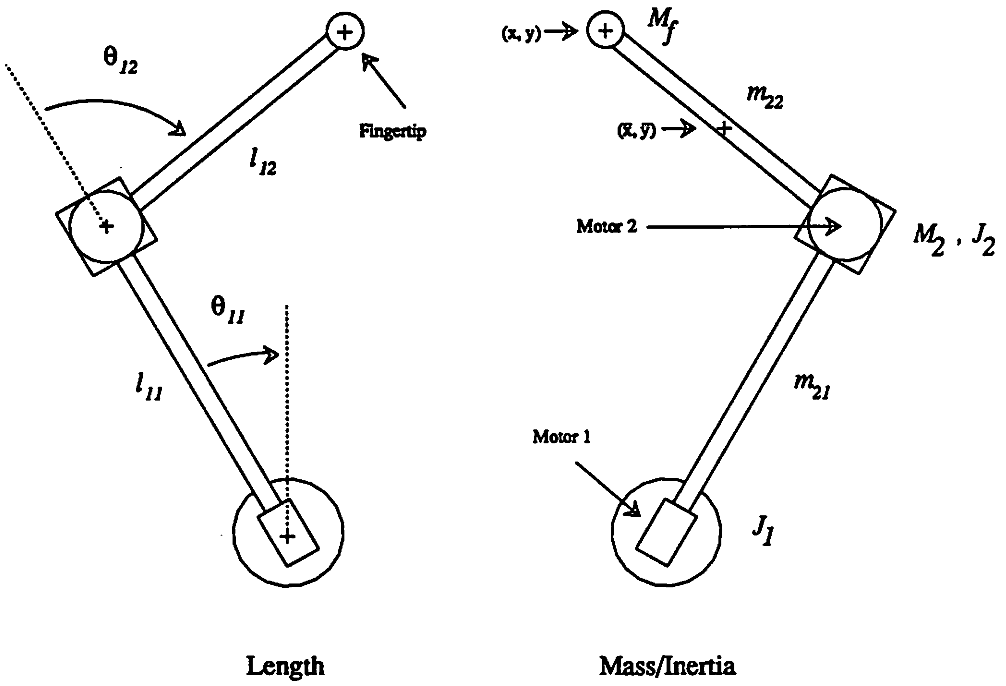

Figure A.3: Dynamic parameters

We derive the dynamic equations for Styx using the Lagrangian formulation. Since the fingers are kinematically identical we will only derive the equations for a single finger. The kinematic energy for each link is given by

$$K_1 = \left(\frac{1}{2}m_1\left(\frac{l_1}{2}\right)^2 + \frac{1}{2}M_2 l_1^2 + \frac{1}{2}J_1\right)\dot{\theta}_1^2$$

$$K_2 = \frac{1}{2}m_2\left(\dot{\bar{x}}^2 + \dot{\bar{y}}^2\right) + \frac{1}{2}M_f\left(\dot{x}^2 + \dot{y}^2\right) + \frac{1}{2}J_2\dot{\theta}_2^2$$

where

$$\begin{aligned} (x, y) &= \text{position of the end effector} \\ (\bar{x}, \bar{y}) &= \text{center of mass for link 2} \end{aligned}$$

We can calculate the velocity of the end effector from the Jacobian (derived earlier)

$$\begin{aligned} x &= l_1 \sin(\theta_1) + l_2 \sin(\theta_2 + \theta_2) + b/2 \\ y &= l_1 \cos(\theta_1) + l_2 \cos(\theta_2 + \theta_2) \end{aligned}$$

$$
\begin{aligned}
\dot{x} &= \Big(l_1\cos(\theta_1) + l_2\cos(\theta_1+\theta_2)\Big)\dot{\theta}_1 + \Big(l_2\cos(\theta_2+\theta_1)\Big)\dot{\theta}_2 \\
\dot{y} &= -\Big(l_1\sin(\theta_1) + l_2\sin(\theta_1+\theta_2)\Big)\dot{\theta}_1 - \Big(l_2\sin(\theta_2+\theta_1)\Big)\dot{\theta}_2
\end{aligned}
$$

Now we can square the terms and combine them

$$
\begin{aligned}
\dot{x}^2 + \dot{y}^2 &= l_1^2\dot{\theta}_1^2 + l_2^2\Big(\dot{\theta}_1^2 + 2\dot{\theta}_1\dot{\theta}_2 + \dot{\theta}_2^2\Big) + \\
&\quad 2l_1l_2\cos\theta_1\cos(\theta_1+\theta_2)\Big(\dot{\theta}_1^2 + \dot{\theta}_1\dot{\theta}_2\Big) + \\
&\quad 2l_1l_2\sin\theta_1\sin(\theta_1+\theta_2)\Big(\dot{\theta}_1^2 + \dot{\theta}_1\dot{\theta}_2\Big) \\
&= l_1^2\dot{\theta}_1^2 + l_2^2\Big(\dot{\theta}_1^2 + 2\dot{\theta}_1\dot{\theta}_2 + \dot{\theta}_2^2\Big) + 2l_1l_2\cos\theta_2\Big(\dot{\theta}_1^2 + \dot{\theta}_1\dot{\theta}_2\Big)
\end{aligned}
$$

The calculation for the velocity of the center of mass of link 2 is identical with $l_2/2$ substituted for l_2. Substituting these velocities into the kinetic energy equation we get the kinetic energy of the second link in terms of the joint positions and velocities

$$
\begin{aligned}
K_2 &= \frac{1}{2}(m_2 + M_f)l_1^2\dot{\theta}_1^2 + \frac{1}{2}J_2\dot{\theta}_2^2 + \\
&\quad \frac{1}{2}m_2\left(\frac{l_2}{2}\right)^2\Big(\dot{\theta}_1^2 + 2\dot{\theta}_1\dot{\theta}_2 + \dot{\theta}_2^2\Big) + m_2l_1\left(\frac{l_2}{2}\right)\cos\theta_2\Big(\dot{\theta}_1^2 + \dot{\theta}_1\dot{\theta}_2\Big) + \\
&\quad \frac{1}{2}M_fl_2^2\Big(\dot{\theta}_1^2 + 2\dot{\theta}_1\dot{\theta}_2 + \dot{\theta}_2^2\Big) + M_fl_1l_2\cos\theta_2\Big(\dot{\theta}_1^2 + \dot{\theta}_1\dot{\theta}_2\Big)
\end{aligned}
$$

Now we can use Lagrange's equation

$$
\tau_i = \frac{d}{dt}\frac{\partial L}{\partial\dot{\theta}_i} - \frac{\partial L}{\partial\theta_i}; \qquad L = K_1 + K_2 + U
$$

where U is a potential energy term, which is zero for Styx. To simplify the equations we make the following substitutions

$$
\begin{aligned}
A_1 &= m_1\left(\frac{l_1}{2}\right)^2 + M_2l_1^2 + J_1 + (m_2 + M_f)l_1^2 \\
A_2 &= J_2 \\
B_1 &= m_2\left(\frac{l_2}{2}\right)^2 + M_fl_2^2 \\
B_2 &= 2m_2l_1\left(\frac{l_2}{2}\right) + 2M_fl_1l_2
\end{aligned}
$$

which gives a Lagrangian of

$$
\begin{aligned}
L &= K_1 + K_2 \\
&= \frac{1}{2}A_1\dot{\theta}_1^2 + \frac{1}{2}A_2\dot{\theta}_2^2 + \frac{1}{2}B_1\Big(\dot{\theta}_1^2 + 2\dot{\theta}_1\dot{\theta}_2 + \dot{\theta}_2^2\Big) + \frac{1}{2}B_2\cos\theta_2\Big(\dot{\theta}_1^2 + \dot{\theta}_1\dot{\theta}_2\Big)
\end{aligned}
$$

Working out all the partial terms:

$$
\begin{aligned}
\frac{\partial L}{\partial \dot{\theta}_1} &= A_1\dot{\theta}_1 + B_1\dot{\theta}_1 + B_1\dot{\theta}_2 + B_2\cos\theta_2\dot{\theta}_1 + \frac{1}{2}B_2\cos\theta_2\dot{\theta}_2 \\
\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_1}\right) &= A_1\ddot{\theta}_1 + B_1\ddot{\theta}_1 + B_1\ddot{\theta}_2 + B_2\cos\theta_2\left(\ddot{\theta}_1 + \frac{1}{2}\ddot{\theta}_2\right) - B_2\sin\theta_2\left(\dot{\theta}_1\dot{\theta}_2 + \frac{1}{2}\dot{\theta}_2^2\right) \\
\frac{\partial L}{\partial \theta_1} &= 0 \\
\tau_1 &= \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_1}\right)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial L}{\partial \dot{\theta}_2} &= A_2\dot{\theta}_1 + B_1\dot{\theta}_1 + B_1\dot{\theta}_2 + \frac{1}{2}B_2\cos\theta_2\dot{\theta}_1 \\
\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_2}\right) &= A_2\ddot{\theta}_2 + B_1\ddot{\theta}_1 + B_1\ddot{\theta}_2 + \frac{1}{2}B_2\cos\theta_2\ddot{\theta}_1 - \frac{1}{2}B_2\sin\theta_2\dot{\theta}_1\dot{\theta}_2 \\
\frac{\partial L}{\partial \theta_2} &= -\frac{1}{2}B_2\sin\theta_2\left(\dot{\theta}_1^2 + \dot{\theta}_1\dot{\theta}_2\right) \\
\tau_2 &= A_2\ddot{\theta}2 + B_1\ddot{\theta}_1 + B_1\ddot{\theta}_2 + \frac{1}{2}B_2\cos\theta_2\ddot{\theta}_1 + \frac{1}{2}B_2\sin\theta_2\dot{\theta}_1^2
\end{aligned}
$$

Giving us the complete dynamics:

$$
\tau = \begin{bmatrix} A_1 + B_1 + B_2\cos\theta_2 & B_1 + \frac{1}{2}B_2\cos\theta_2 \\ B_1 + \frac{1}{2}B_2\cos\theta_2 & A_2 + B_1 \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} + \begin{bmatrix} B_2\sin\theta_2(-\frac{1}{2}\dot{\theta}_2^2 - \dot{\theta}_1\dot{\theta}_2) \\ \frac{1}{2}B_2\sin\theta_2\dot{\theta}_1^2 \end{bmatrix}
$$

A.3 Styx parameters

		Left finger	Right finger
Length	Link length	$l_{11} = 15.24$ cm	$l_{21} = 15.24$ cm
		$l_{12} = 12.16$ cm	$l_{22} = 12.55$ cm
	Fingertip radius	$r_f = 1.91$ cm	

		Left finger	Right finger
Mass	Link mass	$m_{11} = 53$ g	$m_{21} = 53$ g
		$m_{12} = 20$ g	$m_{22} = 17$ g
	Motor mass	$M_{11} = 328$ g	$M_{21} = 328$ g
	Fingertip mass	$M_f = 3$ g	

Inertia	Motor inertia	$J_{11} = J_{21} = 18.0$ g cm^2
		$J_{12} = J_{22} = 1.74$ g cm^2

Appendix B

STYX control algorithm sources

```
/*
 * joint.c - resolved acceleration control in joint space
 *
 * Richard M. Murray
 * July 6, 1988
 *
 */

#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <styx/hctl.h>
#include <styx/isr.h>
#include "styx.h"
#include "joint.h"

/* Function declarations */
int joint_control(), joint_update();

/* Display variables */
int joint_cfreq, joint_tfreq;
struct pid_structure joint_pid[5];
#define jpid joint_pid

/* Local variables */
static int th1_2, th2_2;
static double I[4][5], theta[4];
#define POS2RAD 1.7453292e-3        /* convert count to radians: 2*PI / RES11 */

/* Variables for computing the inputs */
double far jpos[1024][4], far jacc[1024][4];
extern struct opdata {double pos[3], acc[3];} far traj[1024];
extern int input_count, input_index;
extern int input, input_char;
static int joint_index;

/* Display tables */
#include "joint.tbl"

/* Initialize joint control */
joint_setup()
{
    register int i;

    /* Set up menu links */
    menutbl[Watch] = joint_watch;

    /* Initialize the inertia matrix */
    I[0][0] = Mf11 * styx[0].gain;  I[0][1] = Mf12 * styx[0].gain;
    I[1][0] = Mf12 * styx[1].gain;  I[1][1] = Mf22 * styx[1].gain;

    I[2][2] = Mf33 * styx[2].gain;  I[2][3] = Mf34 * styx[2].gain;
    I[3][2] = Mf34 * styx[3].gain;  I[3][3] = Mf44 * styx[3].gain;

    /* calculate desired end effector location */
    joint_inverse(theta, pid[0].desired, pid[1].desired, pid[2].desired);
    for (i = 0; i < 4; ++i) {
        jpid[i].desired = theta[i];
        jpid[i].accel = 0;
    }

    /* Set up interrrupt routines */
    isr_set_cfreq(joint_cfreq);
    isr_set_tfreq(joint_tfreq);
    isr_set_croutine(joint_control);
    isr_set_troutine(joint_update);
}

/* Joint control law */
joint_control()
{
    register i;
    flag(0, 'C');

    /* Get new position and acceleration */
    joint_index = input_index;
    input_update(1);
    if (control_on && input != Setpoint) {
        for (i = 0; i < 4; ++i) {
            jpid[i].desired = jpos[joint_index][i];
            jpid[i].accel = jacc[joint_index][i];
        }
    } else
        flag(9, 0x47);

    /* Read in the angles and convert them */
    hctl_read_position_tbl(HCTL_ACTUAL, hctl_actual);

    /* PD controller in joint space */
    th1_1  = ((hctl_actual[0] - styx[0].reference) * NUM11) / DEN11;
    jpid[0].error = jpid[0].desired - th1_1 * POS2RAD;
    jpid[0].output = jpid[0].accel + jpid[0].K *
                    (jpid[0].error - jpid[0].Kz * jpid[0].last_error);
    jpid[0].last_error = jpid[0].error;

    th1_2 = ((hctl_actual[1] - styx[1].reference) * NUM12) / DEN12;
    jpid[1].error = jpid[1].desired - th1_2 * POS2RAD;
    jpid[1].output = jpid[1].accel + jpid[1].K *
                    (jpid[1].error - jpid[1].Kz * jpid[1].last_error);
    jpid[1].last_error = jpid[1].error;

    /* Deleted encoder scaling - 10/27/88 */
    th2_1  = (hctl_actual[2] - styx[2].reference);
    jpid[2].error = jpid[2].desired - th2_1 * POS2RAD;
    jpid[2].output = jpid[2].accel + jpid[2].K *
                    (jpid[2].error - jpid[2].Kz * jpid[2].last_error);
    jpid[2].last_error = jpid[2].error;

    th2_2 = ((hctl_actual[3] - styx[3].reference) * NUM22) / DEN22;
    jpid[3].error = jpid[3].desired - th2_2 * POS2RAD;
    jpid[3].output = jpid[3].accel + jpid[3].K *
                    (jpid[3].error - jpid[3].Kz * jpid[3].last_error);
    jpid[3].last_error = jpid[3].error;

    if (control_on) {
        flag(1, 0x27);

        /* Multiply by the inertia matrix */
        ipwm[0] = I[0][0]*jpid[0].output + I[0][1]*jpid[1].output + I[0][4];
        ipwm[1] = I[1][0]*jpid[0].output + I[1][1]*jpid[1].output + I[1][4];
        ipwm[2] = I[2][2]*jpid[2].output + I[2][3]*jpid[3].output + I[2][4];
        ipwm[3] = I[3][2]*jpid[2].output + I[3][3]*jpid[3].output + I[3][4];

        /* Store pulse widths in table */
        for (i = 0; i < 4; ++i) {
            if (ipwm[i] < 0) {
                ipwm[i] -= styx[i].offset;
                hctl_pwm[i] = ipwm[i] < -styx[i].limit ?
                              -styx[i].limit : ipwm[i];

            } else if (ipwm[i] > 0) {
                ipwm[i] += styx[i].offset;
                hctl_pwm[i] = ipwm[i] > styx[i].limit ?
                              styx[i].limit : ipwm[i];

            } else
                hctl_pwm[i] = 0;
        }

    } else {
        flag(1, 0x47);
        for (i = 0; i < 4; ++i)
            hctl_pwm[i] = 0;
    }

    hctl_write_register_tbl(HCTL_PWM, hctl_pwm);
    dump_update();
}

static int joint_update()
{
    static double r;

    flag(2, 'T');
    flag(3, 0x27);

    /* Calculate the end effector location */
    l11_s1 = l11_sin[th1_1 + TBLOFF];
    l11_c1 = l11_cos[th1_1 + TBLOFF];
    th1_12 = th1_1 + th1_2 + TBLOFF;
    l12_s12 = l12_sin[th1_12];
    l12_c12 = l12_cos[th1_12];

    l21_s1 = l21_sin[th2_1 + TBLOFF];
    l21_c1 = l21_cos[th2_1 + TBLOFF];
    th2_12 = th2_1 + th2_2 + TBLOFF;
    l22_s12 = l22_sin[th2_12];
    l22_c12 = l22_cos[th2_12];

    x_1 = l11_s1 + l12_s12;
    x_2 = l21_s1 + l22_s12;
    y_1 = l11_c1 + l12_c12;
    y_2 = l21_c1 + l22_c12;

    dy = y_1 - y_2;
    dx = x_1 - x_2 + BASELINE;
    x = (x_1 + x_2)/2;
    y = (y_1 + y_2)/2;

#   ifdef UNUSED
    atan2_87(dy = y_1 - y_2, dx = x_1 - x_2 + BASELINE, &r); t = r;
    d_squared = dx*dx + dy*dy;
    sqrt_87(d_squared, &r); d = r;
#   endif

    /* Add finger force here (instead of in ISR) */
    I[0][4] = (dy*x_1 - dx*y_1) * jpid[4].K * styx[0].gain;
    I[1][4] = (dy*l12_s12 - dx*l12_c12) * jpid[4].K * styx[1].gain;
    I[2][4] = (dx*y_2 - dy*x_2) * jpid[4].K * styx[2].gain;
    I[3][4] = (dx*l22_c12 - dy*l22_s12) * jpid[4].K * styx[3].gain;

}

/* Inverse kinematics  - this causes stack problems !! */
joint_inverse(th, xd, yd, td)
double th[4], xd, yd, td;
{
    double alpha, beta, xy2;
    double x_1d, y_1d, x_2d, y_2d;
    double th1_1d, th1_2d, th2_1d, th2_2d;
#   define PI_2 (3.14159265358979/2)

    /* Calculate the end effector positions */
    alpha = object_radius * cos(td);
    beta  = object_radius * sin(td);

    x_1d = (2*xd + alpha - BASELINE) / 2;
    y_1d = (2*yd + beta) / 2;
    x_2d = (2*xd - alpha + BASELINE) / 2;
    y_2d = (2*yd - beta) / 2;

    /* Calculate the desired joint angles */
    /* Choose the signs of the angles based on the usual configuration */
    xy2 = x_1d*x_1d + y_1d*y_1d;
    /*! Finger 1, joint 1 encoder spins backwards => negate angle !*/
    th[1] = -acos((xy2 - L11*L11 - L12*L12) / (2*L11*L12));
    th[0] = (double) PI_2 - atan2(y_1d, x_1d) +
            acos((xy2 + L11*L11 - L12*L12) / (2*L11*sqrt(xy2)));

    xy2 = x_2d*x_2d + y_2d*y_2d;
    th[3] = acos((xy2 - L21*L21 - L22*L22) / (2*L21*L22));
    th[2] = (double) PI_2 - atan2(y_2d, x_2d) -
            acos((xy2 + L21*L21 - L22*L22) / (2*L21*sqrt(xy2)));
}

/* Calculate joint position and accelerations */
joint_input()
{
    register int i, j;
    double F;

    ddprompt("solving inverse kinematics...");

    for (i = 0; i < input_count; ++i) {
        (void) joint_inverse(theta, traj[i].pos[0],
                             traj[i].pos[1], traj[i].pos[2]);
        for (j = 0; j < 4; ++j) jpos[i][j] = theta[j];
    }

    /* Figure out the acceleration */
    F = 4 / (double) (isr_cfreq * isr_cfreq);
    for (i = 0; i < input_count; ++i) {
        int last = (i == 0 ? input_count-1 : i-1);
        int next = (i < input_count-1 ? i+1 : 0);

        for (j = 0; j < 4; ++j)
            jacc[i][j] = (jpos[last][j] - 2*jpos[i][j] + jpos[next][j]) / F;
    }
    ddprompt("");
}
```

```
/*
 * natural.c - natural control law for STYX
 *
 * Richard M. Murray
 * August 4, 1988
 *
 */

#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <styx/hctl.h>
#include <styx/isr.h>
#include "styx.h"
#include "dynamics.h"

/* Function declarations */
int natural_control(), natural_update();

/* Variables for natural control and update */
static int dynamic_size;        /* from torque.c */

/* Display variables */
int natural_cfreq, natural_tfreq;
struct pid_structure natural_pid[4];
char natural_title[] = "natural controller";

static double detj1, detj2;
static double dx_div2, dy_div2;
static double dxi_div_d2, dyi_div_d2;
static double mdiv2, size_ratio;
static double Ji0, Ji1;
static double Jv1, Jv2, Jv3, Jv4, T1, T2;
static double thdot[4], thdot12, thdot34, pos2rad;

static double M[4][4], I[4][4], J[4][4];

#define TWOPI 6.2831852

/* Display tables */
#include "natural.tbl"

/* Initialize natural control */
natural_setup()
{
    /* Set up menu links */
    menutbl[Watch] = natural_watch;

    /* Initialize constants */
    mdiv2 = Mo/2;
    pos2rad = (TWOPI / TBLOFF) * tfreq;

    /* Mass matrix */
    M[0][0] = Mf11; M[0][1] = Mf12;
    M[1][0] = Mf21; M[1][1] = Mf22;
    M[2][2] = Mf33; M[2][3] = Mf34;
    M[3][2] = Mf43; M[3][3] = Mf44;

    /* Set up interrrupt routines */
    isr_set_cfreq(natural_cfreq);
    isr_set_tfreq(natural_tfreq);
    isr_set_croutine(natural_control);
    isr_set_troutine(natural_update);
```

```
}

#define MUL(i,j) I[i][j]*pid[j].accel + J[i][j]*pid[j].output

/* natural control law */
natural_control()
{
    register i;
    static double r;

    /* Get new position and acceleration */
    flag(0, 'C');
    input_update(1);

    /* Read in the angles and convert them */
    hctl_read_position_tbl(HCTL_ACTUAL, hctl_actual);

    /* Look up trigonemtric data in tables */
    th1_1  = TBLOFF + ((hctl_actual[0] - styx[0].reference) * NUM11) / DEN11;
    l11_s1 = l11_sin[th1_1];
    l11_c1 = l11_cos[th1_1];

    th1_12 = th1_1 +  ((hctl_actual[1] - styx[1].reference) * NUM12) / DEN12;
    l12_s12 = l12_sin[th1_12];
    l12_c12 = l12_cos[th1_12];

    th2_1 =  TBLOFF + ((hctl_actual[2] - styx[2].reference) * NUM21) / DEN21;
    l21_s1 = l21_sin[th2_1];
    l21_c1 = l21_cos[th2_1];

    th2_12 = th2_1 +  ((hctl_actual[3] - styx[3].reference) * NUM22) / DEN22;
    l22_s12 = l22_sin[th2_12];
    l22_c12 = l22_cos[th2_12];

    /* PD controller in operational space */
    x_1 = l11_s1 + l12_s12;
    x_2 = l21_s1 + l22_s12;
    x = (x_1 + x_2)/2;
    pid[0].error = pid[0].desired - x;
    pid[0].output = natural_pid[0].K *
                    (pid[0].error - natural_pid[0].Kz * pid[0].last_error);
    pid[0].last_error = pid[0].error;

    y_1 = l11_c1 + l12_c12;
    y_2 = l21_c1 + l22_c12;
    y = (y_1 + y_2)/2;
    pid[1].error = pid[1].desired - y;
    pid[1].output = natural_pid[1].K *
                    (pid[1].error - natural_pid[1].Kz * pid[1].last_error);
    pid[1].last_error = pid[1].error;

    atan2_87(dy = y_1 - y_2, dx = x_1 - x_2 + BASELINE, &r); t = r;
    pid[2].error = pid[2].desired - t;
    pid[2].output = natural_pid[2].K *
                    (pid[2].error - natural_pid[2].Kz * pid[2].last_error);
    pid[2].last_error = pid[2].error;

    if (control_on) {
        flag(1, 0x27);

        /* Multiply by the inertia matrix */
        ipwm[0] = MUL(0, 0) + MUL(0, 1) + MUL(0, 2) + I[0][3];
```

```
            ipwm[1] = MUL(1, 0) + MUL(1, 1) + MUL(1, 2) + I[1][3];
            ipwm[2] = MUL(2, 0) + MUL(2, 1) + MUL(2, 2) + I[2][3];
            ipwm[3] = MUL(3, 0) + MUL(3, 1) + MUL(3, 2) + I[3][3];

            /* Store pulse widths in·table */
            for (i = 0; i < 4; ++i) {
                if (ipwm[i] < 0) {
                    ipwm[i] -= styx[i].offset;
                    hctl_pwm[i] = ipwm[i] < -styx[i].limit ?
                                  -styx[i].limit : ipwm[i];

                } else if (ipwm[i] > 0) {
                    ipwm[i] += styx[i].offset;
                    hctl_pwm[i] = ipwm[i] > styx[i].limit ?
                                  styx[i].limit : ipwm[i];

                } else
                    hctl_pwm[i] = 0;
            }

        } else {
            flag(1, 0x47);
            for (i = 0; i < 4; ++i)
                hctl_pwm[i] = 0;
        }

        hctl_write_register_tbl(HCTL_PWM, hctl_pwm);
        dump_update();
    }

    static int natural_update()
    {
        static double r;
        register int k, j, i;

        flag(2, 'T');
        flag(3, 0x27);

        /* Use the joint angles and positions from the latest controller run */
        /* Distance between fingertips (no controller) */
        /* Calculate distance between fingers; dx, dy from controller */
        d_squared = dx*dx + dy*dy;
        sqrt_87(d_squared, &r); d = r;

        /* Object size */
        size_ratio = dynamic_size ? 0.5 : object_radius/d;

        detj1 = l11_s1*l12_c12 - l11_c1*l12_s12;
        detj2 = l21_s1*l22_c12 - l21_c1*l22_s12;

        dx_div2 = dx * size_ratio;
        dy_div2 = dy * size_ratio;

        M[0][0] = Mf11 / detj1;  M[0][1] = Mf12 / detj1;  M[1][1] = Mf22 / detj1;
        M[2][2] = Mf33 / detj2;  M[2][3] = Mf34 / detj2;  M[3][3] = Mf44 / detj2;

        dxi_div_d2 = dx * Io / d_squared;
        dyi_div_d2 = dy * Io / d_squared;

        I[0][0] = (J[0][0] = Jt00 * styx[0].gain) +
                  (M[0][0] * Ji00 + M[0][1] * Ji10) * styx[0].gain;
        I[1][0] = (J[1][0] = Jt10 * styx[1].gain) +
                  (M[0][1] * Ji00 + M[1][1] * Ji10) * styx[1].gain;
    I[2][0] = (J[2][0] = Jt20 * styx[2].gain) +
              (M[2][2] * Ji20 + M[2][3] * Ji30) * styx[2].gain;
    I[3][0] = (J[3][0] = Jt30 * styx[3].gain) +
              (M[2][3] * Ji20 + M[3][3] * Ji30) * styx[3].gain;

    I[0][1] = (J[0][1] = Jt01 * styx[0].gain) +
              (M[0][0] * Ji01 + M[0][1] * Ji11) * styx[0].gain;
    I[1][1] = (J[1][1] = Jt11 * styx[1].gain) +
              (M[0][1] * Ji01 + M[1][1] * Ji11) * styx[1].gain;
    I[2][1] = (J[2][1] = Jt21 * styx[2].gain) +
              (M[2][2] * Ji21 + M[2][3] * Ji31) * styx[2].gain;
    I[3][1] = (J[3][1] = Jt31 * styx[3].gain) +
              (M[2][3] * Ji21 + M[3][3] * Ji31) * styx[3].gain;

    I[0][2] = (J[0][2] = Jt02 * styx[0].gain) +
              (M[0][0] * (Ji0 = Ji02) + M[0][1] * (Ji1 = Ji12)) * styx[0].gain;
    I[1][2] = (J[1][2] = Jt12 * styx[1].gain) +
              (M[0][1] * Ji0 + M[1][1] * Ji1) * styx[1].gain;
    I[2][2] = (J[2][2] = Jt22 * styx[2].gain) +
              (M[2][2] * (Ji0 = Ji22) + M[2][3] * (Ji1 = Ji32)) * styx[2].gain;
    I[3][2] = (J[3][2] = Jt32 * styx[3].gain) +
              (M[2][3] * Ji0 + M[3][3] * Ji1) * styx[3].gain;

    /* Add finger force here (instead of in ISR) */
    I[0][3] = (dy*x_1 - dx*y_1) * natural_pid[3].K * styx[0].gain;
    I[1][3] = (dy*l12_s12 - dx*l12_c12) * natural_pid[3].K * styx[1].gain;
    I[2][3] = (dx*y_2 - dy*x_2) * natural_pid[3].K * styx[2].gain;
    I[3][3] = (dx*l22_c12 - dy*l22_s12) * natural_pid[3].K * styx[3].gain;
}
```

```
/*
 * stiff.c - stiff control law for STYX
 *
 * Richard M. Murray
 * August 4, 1988
 *
 */

#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <styx/hctl.h>
#include <styx/isr.h>
#include "styx.h"
#include "dynamics.h"

/* Function declarations */
int stiff_control(), stiff_update();

/* Variables for stiff control and update */
static int dynamic_size;        /* from torque.c */

/* Display variables */
int stiff_cfreq, stiff_tfreq;
struct pid_structure stiff_pid[4];
char stiff_title[] = "stiff controller";

static double detj1, detj2;
static double dx_div2, dy_div2;
static double dxi_div_d2, dyi_div_d2;
static double mdiv2, size_ratio;
static double Ji0, Ji1;
static double Jv1, Jv2, Jv3, Jv4, T1, T2;
static double thdot[4], thdot12, thdot34, pos2rad;

static double M[4][4], I[4][4], J[4][4];

#define TWOPI 6.2831852

/* Display tables */
#include "stiff.tbl"

/* Initialize stiff control */
stiff_setup()
{
    /* Set up menu links */
    menutbl[Watch] = stiff_watch;

    /* Initialize constants */
    mdiv2 = Mo/2;
    pos2rad = (TWOPI / TBLOFF) * tfreq;

    /* Set up interrrupt routines */
    isr_set_cfreq(stiff_cfreq);
    isr_set_tfreq(stiff_tfreq);
    isr_set_croutine(stiff_control);
    isr_set_troutine(stiff_update);

}

#define MUL(i,j) I[i][j] * (pid[j].accel + stiff_pid[j].Kp * \
                 (pid[j].error - pid[j].last_error)) + J[i][j] * pid[j].output
```

```
/* stiff control law */
stiff_control()
{
    register i;
    static double r;

    /* Get new position and acceleration */
    flag(0, 'C');
    input_update(1);

    /* Read in the angles and convert them */
    hctl_read_position_tbl(HCTL_ACTUAL, hctl_actual);

    /* Look up trigonemtric data in tables */
    th1_1  = TBLOFF + ((hctl_actual[0] - styx[0].reference) * NUM11) / DEN11;
    l11_s1 = l11_sin[th1_1];
    l11_c1 = l11_cos[th1_1];

    th1_12 = th1_1 +  ((hctl_actual[1] - styx[1].reference) * NUM12) / DEN12;
    l12_s12 = l12_sin[th1_12];
    l12_c12 = l12_cos[th1_12];

    th2_1 =  TBLOFF + ((hctl_actual[2] - styx[2].reference) * NUM21) / DEN21;
    l21_s1 = l21_sin[th2_1];
    l21_c1 = l21_cos[th2_1];

    th2_12 = th2_1 +  ((hctl_actual[3] - styx[3].reference) * NUM22) / DEN22;
    l22_s12 = l22_sin[th2_12];
    l22_c12 = l22_cos[th2_12];

    /* PD controller in operational space */
    x_1 = l11_s1 + l12_s12;
    x_2 = l21_s1 + l22_s12;
    x = (x_1 + x_2)/2;
    pid[0].error = pid[0].desired - x;
    pid[0].output = stiff_pid[0].K *
                    (pid[0].error - stiff_pid[0].Kz * pid[0].last_error);

    y_1 = l11_c1 + l12_c12;
    y_2 = l21_c1 + l22_c12;
    y = (y_1 + y_2)/2;
    pid[1].error = pid[1].desired - y;
    pid[1].output = stiff_pid[1].K *
                    (pid[1].error - stiff_pid[1].Kz * pid[1].last_error);

    atan2_87(dy = y_1 - y_2, dx = x_1 - x_2 + BASELINE, &r); t = r;
    pid[2].error = pid[2].desired - t;
    pid[2].output = stiff_pid[2].K *
                    (pid[2].error - stiff_pid[2].Kz * pid[2].last_error);

    if (control_on) {
        flag(1, 0x27);

        /* Multiply by the inertia matrix */
        ipwm[0] = MUL(0, 0) + MUL(0, 1) + MUL(0, 2) + I[0][3];
        ipwm[1] = MUL(1, 0) + MUL(1, 1) + MUL(1, 2) + I[1][3];
        ipwm[2] = MUL(2, 0) + MUL(2, 1) + MUL(2, 2) + I[2][3];
        ipwm[3] = MUL(3, 0) + MUL(3, 1) + MUL(3, 2) + I[3][3];

        /* Store pulse widths in table */
        for (i = 0; i < 4; ++i) {
            if (ipwm[i] < 0) {
                ipwm[i] -= styx[i].offset;
                hctl_pwm[i] = ipwm[i] < -styx[i].limit ?
                              -styx[i].limit : ipwm[i];

            } else if (ipwm[i] > 0) {
                ipwm[i] += styx[i].offset;
                hctl_pwm[i] = ipwm[i] > styx[i].limit ?
                              styx[i].limit : ipwm[i];

            } else
                hctl_pwm[i] = 0;
        }

    } else {
        flag(1, 0x47);
        for (i = 0; i < 4; ++i)
            hctl_pwm[i] = 0;
    }

    /* Save errors */
    pid[0].last_error = pid[0].error;
    pid[1].last_error = pid[1].error;
    pid[2].last_error = pid[2].error;

    hctl_write_register_tbl(HCTL_PWM, hctl_pwm);
    dump_update();
}

static int stiff_update()
{
    static double r;
    register int k, j, i;

    flag(2, 'T');
    flag(3, 0x27);

    /* Use the joint angles and positions from the latest controller run */
    /* Distance between fingertips (no controller) */
    /* Calculate distance between fingers; dx, dy from controller */
    d_squared = dx*dx + dy*dy;
    sqrt_87(d_squared, &r); d = r;

    /* Object size */
    size_ratio = dynamic_size ? 0.5 : object_radius/d;

    detj1 = l11_s1*l12_c12 - l11_c1*l12_s12;
    detj2 = l21_s1*l22_c12 - l21_c1*l22_s12;

    dx_div2 = dx * size_ratio;
    dy_div2 = dy * size_ratio;

    M[0][0] = Mf11 / detj1;  M[0][1] = Mf12 / detj1;  M[1][1] = Mf22 / detj1;
    M[2][2] = Mf33 / detj2;  M[2][3] = Mf34 / detj2;  M[3][3] = Mf44 / detj2;

    dxi_div_d2 = dx * Io / d_squared;
    dyi_div_d2 = dy * Io / d_squared;

    I[0][0] = (J[0][0] = Jt00 * styx[0].gain) +
              (M[0][0] * Ji00 + M[0][1] * Ji10) * styx[0].gain;
    I[1][0] = (J[1][0] = Jt10 * styx[1].gain) +
              (M[0][1] * Ji00 + M[1][1] * Ji10) * styx[1].gain;
    I[2][0] = (J[2][0] = Jt20 * styx[2].gain) +
              (M[2][2] * Ji20 + M[2][3] * Ji30) * styx[2].gain;
    I[3][0] = (J[3][0] = Jt30 * styx[3].gain) +
                (M[2][3] * Ji20 + M[3][3] * Ji30) * styx[3].gain;

    I[0][1] = (J[0][1] = Jt01 * styx[0].gain) +
              (M[0][0] * Ji01 + M[0][1] * Ji11) * styx[0].gain;
    I[1][1] = (J[1][1] = Jt11 * styx[1].gain) +
              (M[0][1] * Ji01 + M[1][1] * Ji11) * styx[1].gain;
    I[2][1] = (J[2][1] = Jt21 * styx[2].gain) +
              (M[2][2] * Ji21 + M[2][3] * Ji31) * styx[2].gain;
    I[3][1] = (J[3][1] = Jt31 * styx[3].gain) +
              (M[2][3] * Ji21 + M[3][3] * Ji31) * styx[3].gain;

    I[0][2] = (J[0][2] = Jt02 * styx[0].gain) +
              (M[0][0] * (Ji0 = Ji02) + M[0][1] * (Ji1 = Ji12)) * styx[0].gain;
    I[1][2] = (J[1][2] = Jt12 * styx[1].gain) +
              (M[0][1] * Ji0 + M[1][1] * Ji1) * styx[1].gain;
    I[2][2] = (J[2][2] = Jt22 * styx[2].gain) +
              (M[2][2] * (Ji0 = Ji22) + M[2][3] * (Ji1 = Ji32)) * styx[2].gain;
    I[3][2] = (J[3][2] = Jt32 * styx[3].gain) +
              (M[2][3] * Ji0 + M[3][3] * Ji1) * styx[3].gain;

    /* Add finger force here (instead of in ISR) */
    I[0][3] = (dy*x_1 - dx*y_1) * stiff_pid[3].K * styx[0].gain;
    I[1][3] = (dy*l12_s12 - dx*l12_c12) * stiff_pid[3].K * styx[1].gain;
    I[2][3] = (dx*y_2 - dy*x_2) * stiff_pid[3].K * styx[2].gain;
    I[3][3] = (dx*l22_c12 - dy*l22_s12) * stiff_pid[3].K * styx[3].gain;
}
```

```
/*
 * torque.c - generalized computed torque algorithm
 *
 * Richard M. Murray
 * March 16, 1988
 *
 */

#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <styx/ddisp.h>
#include <styx/dinit.h>
#include <styx/hctl.h>
#include <styx/isr.h>
#include <styx/ddisp.h>
#include "styx.h"
#include "torque.h"

/* Function declarations */
int torque_control(), torque_update();

/* Variables for torque control and update */
int torque_cfreq, torque_tfreq;
struct pid_structure torque_pid[4];
int dynamic_size = 1;

/* Define tables now that we have declared all our variables */
#include "torque.tbl"

/* Local variables */
static double detj1, detj2;
static double m11_div_detj1, m12_div_detj1, m21_div_detj2, m22_div_detj2;
static double dx_div_2detj1, dy_div_2detj1, dx_div_2detj2, dy_div_2detj2;
static double dxi_div_d2, dyi_div_d2;
static double mdiv2, size_ratio;
static int sign;

/* Initialize torque control */
torque_setup()
{
    /* Set up menu links */
    /*! deleted to save space !*/

    /* Set up interrrupt routines */
    isr_set_cfreq(torque_cfreq);
    isr_set_tfreq(torque_tfreq);
    isr_set_croutine(torque_control);
    isr_set_troutine(torque_update);

    /* Initialize constants */
    mdiv2 = Mo/2;
}

/* Torque control law */
torque_control()
{
    register i;
    static double r;

    flag(0, 'C');
    input_update(1);
```

```
/* Read in the angles and convert them */
hctl_read_position_tbl(HCTL_ACTUAL, hctl_actual);

/* Look up trigonemtric data in tables */
th1_1  = TBLOFF + ((hctl_actual[0] - styx[0].reference) * NUM11) / DEN11;
l11_s1 = l11_sin[th1_1];
l11_c1 = l11_cos[th1_1];

th1_12 = th1_1 +  ((hctl_actual[1] - styx[1].reference) * NUM12) / DEN12;
l12_s12 = l12_sin[th1_12];
l12_c12 = l12_cos[th1_12];

th2_1 =  TBLOFF + ((hctl_actual[2] - styx[2].reference) * NUM21) / DEN21;
l21_s1 = l21_sin[th2_1];
l21_c1 = l21_cos[th2_1];

th2_12 = th2_1 +  ((hctl_actual[3] - styx[3].reference) * NUM22) / DEN22;
l22_s12 = l22_sin[th2_12];
l22_c12 = l22_cos[th2_12];

/* PD controller in operational space */
x_1 = l11_s1 + l12_s12;
x_2 = l21_s1 + l22_s12;
x = x_1 + x_2;
pid[0].error = pid[0].desired - x;
pid[0].output = pid[0].accel + torque_pid[0].K *
                (pid[0].error - torque_pid[0].Kz * pid[0].last_error);
pid[0].last_error  = pid[0].error;

y_1 = l11_c1 + l12_c12;
y_2 = l21_c1 + l22_c12;
y = y_1 + y_2;
pid[1].error = pid[1].desired - y;
pid[1].output = pid[1].accel + torque_pid[1].K *
                (pid[1].error - torque_pid[1].Kz * pid[1].last_error);
pid[1].last_error  = pid[1].error;

atan2_87(dy = y_1 - y_2, dx = x_1 - x_2 + BASELINE, &r); t = r;
pid[2].error = pid[2].desired - t;
pid[2].output = pid[2].accel + torque_pid[2].K *
                (pid[2].error - torque_pid[2].Kz * pid[2].last_error);
pid[2].last_error  = pid[2].error;

if (control_on) {
    flag(1, 0x27);

    /* Convert workspace torques to joint torques */
    ipwm[0] = J11*pid[0].output + J12*pid[1].output +
              J13*pid[2].output - J14;
    ipwm[1] = J21*pid[0].output + J22*pid[1].output +
              J23*pid[2].output - J24;
    ipwm[2] = J31*pid[0].output + J32*pid[1].output +
              J33*pid[2].output - J34;
    ipwm[3] = J41*pid[0].output + J42*pid[1].output +
              J43*pid[2].output - J44;

    /* Store pulse widths in table */
    for (i = 0; i < 4; ++i) {
        if (ipwm[i] < 0) {
            ipwm[i] -= styx[i].offset;
            hctl_pwm[i] = ipwm[i] < -styx[i].limit ?
                          -styx[i].limit : ipwm[i];
        } else if (ipwm[i] > 0) {
                ipwm[i] += styx[i].offset;
                hctl_pwm[i] = ipwm[i] > styx[i].limit ?
                                styx[i].limit : ipwm[i];
            } else
                hctl_pwm[i] = 0;
        }

    } else {
        flag(1, 0x47);
        for (i = 0; i < 4; ++i)
            hctl_pwm[i] = 0;
    }

    hctl_write_register_tbl(HCTL_PWM, hctl_pwm);
    dump_update();
}

static int mass_update()
{
    static double r;

    flag(2, 'T');
    flag(3, 0x27);

    /* Use the joint angles and positions from the latest controller run */
    /* Distance between fingertips (no controller) */
    /* Calculate distance between fingers; dx, dy from controller */
    d_squared = dx*dx + dy*dy;
    sqrt_87(d_squared, &r); d = r;

    /* Object size */
    size_ratio = dynamic_size ? 0.5 : object_radius/d;

    /* Common terms */
    detj1 = l11_s1*l12_c12 - l11_c1*l12_s12;
    detj2 = l21_s1*l22_c12 - l21_c1*l22_s12;

    m11_div_detj1 = M11 / detj1;
    m12_div_detj1 = M12 / detj1;
    m21_div_detj2 = M21 / detj2;
    m22_div_detj2 = M22 / detj2;

    dxi_div_d2 = dx * Io / d_squared;
    dyi_div_d2 = dy * Io / d_squared;

    dx_div_2detj1 = dx * size_ratio / detj1;
    dy_div_2detj1 = dy * size_ratio / detj1;
    dx_div_2detj2 = dx * size_ratio / detj2;
    dy_div_2detj2 = dy * size_ratio / detj2;

    /* Reduced equations */
    J11 = (mdiv2 * y_1 - m11_div_detj1 * l12_s12)    * styx[0].gain;
    J21 = (mdiv2 * l12_c12 + m12_div_detj1 * x_1)    * styx[1].gain;
    J31 = (mdiv2 * y_2 - m21_div_detj2 * l22_s12)    * styx[2].gain;
    J41 = (mdiv2 * l22_c12 + m22_div_detj2 * x_2)    * styx[3].gain;

    J12 = (- m11_div_detj1 * l12_c12 - mdiv2 * x_1) * styx[0].gain;
    J22 = (m12_div_detj1 * y_1 - mdiv2 * l12_s12) * styx[1].gain;
    J32 = (- m21_div_detj2 * l22_c12 - mdiv2 * x_2) * styx[2].gain;
    J42 = (m22_div_detj2 * y_2 - mdiv2 * l22_s12) * styx[3].gain;

    J13 = (M11 * (dy_div_2detj1 * l12_s12 - dx_div_2detj1 * l12_c12) -
           dyi_div_d2 * y_1 - dxi_div_d2 * x_1)          * styx[0].gain;
    J23 = (M12 * (dx_div_2detj1 * y_1 - dy_div_2detj1 * x_1) -
          dxi_div_d2 * l12_s12 - dyi_div_d2 * l12_c12) * styx[1].gain;
    J33 = (M21 * (dx_div_2detj2 * l22_c12 - dy_div_2detj2 * l22_s12) +
          dyi_div_d2 * y_2 + dxi_div_d2 * x_2)         * styx[2].gain;
    J43 = (M22 * (dy_div_2detj2 * x_2 - dx_div_2detj2 * y_2) +
          dxi_div_d2 * l22_s12 + dyi_div_d2 * l22_c12) * styx[3].gain;

    /* Add finger force here (instead of in ISR) */
    J14 = (dx*y_1     - dy*x_1)     * torque_pid[3].K * styx[0].gain;
    J24 = (dx*l12_c12 - dy*l12_s12) * torque_pid[3].K * styx[1].gain;
    J34 = (dy*x_2     - dx*y_2)     * torque_pid[3].K * styx[2].gain;
    J44 = (dy*l22_s12 - dx*l22_c12) * torque_pid[3].K * styx[3].gain;
}

/* Torque update law */
torque_update()
{
    mass_update();
}
```

```
/*
 * dynamics.h - reduced dynamic equations
 *
 * Richard M. Murray
 * August 4, 1988
 *
 */

#define Ji00  -l12_s12
#define Ji10  x_1
#define Ji20  -l22_s12
#define Ji30  x_2

#define Ji01  -l12_c12
#define Ji11  y_1
#define Ji21  -l22_c12
#define Ji31  y_2

#define Ji02  (dy_div2 * l12_s12 - dx_div2 * l12_c12)
#define Ji12  (dx_div2 * y_1 - dy_div2 * x_1)
#define Ji22  (dx_div2 * l22_c12 - dy_div2 * l22_s12)
#define Ji32  (dy_div2 * x_2 - dx_div2 * y_2)

#define Jt00  mdiv2 * y_1
#define Jt10  mdiv2 * l12_c12
#define Jt20  mdiv2 * y_2
#define Jt30  mdiv2 * l22_c12

#define Jt01  -mdiv2 * x_1
#define Jt11  -mdiv2 * l12_s12
#define Jt21  -mdiv2 * x_2
#define Jt31  -mdiv2 * l22_s12

#define Jt02  (-dyi_div_d2 * y_1 - dxi_div_d2 * x_1)
#define Jt12  (-dxi_div_d2 * l12_s12 - dyi_div_d2 * l12_c12)
#define Jt22  (dyi_div_d2 * y_2 + dxi_div_d2 * x_2)
#define Jt32  (dxi_div_d2 * l22_s12 + dyi_div_d2 * l22_c12)

#define Jt03  -(dx*y_1     - dy*x_1)     * pid[3].K
#define Jt13  -(dx*l12_c12 - dy*l12_s12) * pid[3].K
#define Jt23  -(dy*x_2     - dx*y_2)     * pid[3].K
#define Jt33  -(dy*l22_s12 - dx*l22_c12) * pid[3].K
```

Bibliography

[Bej74] A. K. Bejczy. *Robot Arm Dynamics and Control.* Technical Report 33-699, Jet Propulsion Laboratory, 1974.

[CFAB86] B. Chen, R. Fearing, B. Armstrong, and J. Burdick. Nymph: a multiprocessor for manipulation applications. In *IEEE International Conference on Robotics and Automation*, pages 1731–1736, 1986.

[CHS88] A. Cole, J. Hauser, and S. Sastry. Kinematics and control of multifingered hands with rolling contact. In *IEEE International Conference on Robotics and Automation*, pages 228–233, 1988.

[DLSS88] J. Demmel, G. Lafferrier, J. Schwartz, and M. Sharir. Theortical and experimental studies using a multifinger planar manipulator. In *IEEE International Conference on Robotics and Automation*, pages 390–395, 1988.

[Hay86] S. Hayati. Hybrid postion/force control of multi-arm cooperating robots. In *IEEE International Conference on Robotics and Automation*, pages 82–89, 1986.

[Hsu88] P. Hsu. *Control of Mechanical Manipulators.* PhD thesis, Department of Electrical Engineering, University of California, Berkeley, California, 1988.

[JWBI86] S. Jacobsen, J. Wood, K. Bigger, and E. Iverson. The Utah/MIT hand: work in progress. *International Journal of Robotics Research*, 4(3):221–250, 1986.

[Ker84] J. Kerr. *An Analysis of Multi-fingered Hands.* PhD thesis, Stanford University, Department of Mechanical Engineering, 1984.

[Kod84] D. Koditschek. Natural motion for robot arms. In *Proceedings of the 23rd Conference on Decision and Control*, pages 733–735, 1984.

[LHS88] Z. Li, P. Hsu, and S. Sastry. On kinematics and control of multifingered hands. In *IEEE International Conference on Robotics and Automation*, pages 384–389, 1988.

[LWP80] J. Y. S. Luh, M. W. Walker, and R. P. Paul. Resolved acceleration control of mechanical manipulators. *IEEE Transactions on Automatic Control*, AC-25, 1980.

[MHS89] R. Murray, P. Hsu, and S. Sastry. Dynamics and control of constrained robot systems. In *IEEE International Conference on Robotics and Automation*, 1989. (submitted).

[Ngu86] V. Nguyen. *The Synthesis of Stable Force-Closure Grasp.* Master's thesis, Massachusetts Institute of Technology, 1986.

[NNY87] Y. Nakamura, K. Nagai, and T. Yoshikawa. Mechanics of coordinative manipulation of multiple robot mechanisms. In *IEEE International Conference on Robotics and Automation*, pages 991–998, 1987.

[Oka82] T. Okada. Computer control of multijointed finger system for precis object-hanling. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-12(3):289–299, May/June 1982.

[OS88] R. Ortega and M. W. Spong. *Adaptive Motion Control of Rigid Robots: A Tutoral.* Technical Report, Coordinated Sciences Lab., University of Illinois, Urbana, Illinois, 1988.

[Sad87] N. Sadegh. *Adaptive Control of Mechanical Manipulators: Stability and Robustness Analysis.* PhD thesis, Department of Mechanical Engineering, University of California, Berkeley, California, 1987.

[Sal82] J. K. Salisbury. *Kinematic and Force Analysis of Articulated Hands.* PhD thesis, Stanford University, Department of Mechanical Engineering, 1982.

[SL87] J. E. Slotine and W. Li. On the adaptive control of robot manipulators. *The International Journal of Robotics Research*, 6:49–59, 1987.

[SNH*86] D. M. Siegel, S. Narasimhan, J. M. Hollerbach, D. Kriegman, and G. Gerpheide. Computational architecture for the Utah/MIT hand. In *IEEE International Conference on Robotics and Automation*, pages 1884–1889, 1986.

[TBY86] T. Tarn, A. Bejczy, and X. Yuan. Control of two coordinated robots. In *IEEE International Conference on Robotics and Automation*, pages 1193–1202, 1986.

[VD87] S. T. Venkataraman and Theodore E. Djaferis. Multivariable feedback control of the JPL/Stanford hand. In *IEEE International Conference on Robotics and Automation*, pages 77–82, 1987.

[WB88] J. T. Wen and D. S. Bayard. New class of control laws for robot manipulators. part 1: non-adaptive case. *International Journal of Control*, 47(5):1361–1385, 1988.

[Whi82] D. E. Whitney. Quasi-static assembly of compliantly supported rigid parts. *ASME Journal of Dynamic Systems, Measurement and Control*, 104:65–77, March 1982.

[ZL85] Y.F. Zheng and J.Y.S. Luh. Control of two coordiantes robots in motion. In *IEEE Conference on Decision and Control*, pages 1761–1766, 1985.