Robotic Control of Sliding-Object Motion and Orientation

by

Hemanshu M. Lakhani

B.S. Massachusetts Institute of Technology
(1989)

SUBMITTED TO THE DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
August, 1990

Signature of Author [illegible]
Department of Aeronautics and Astronautics
August 27, 1990

Certified by [illegible]
Harold L. Alexander
Professor, Department of Aeronautics and Astronautics
Thesis Supervisor

Accepted by [illegible]
Professor Harold Y. Wachman, Chairman
Departmental Graduate Committee

Robotic Control of Sliding-Object Motion and Orientation

by

Hemanshu M. Lakhani

Submitted to the Department of Aeronautics and Astronautics in partial fulfillment of the requirements for the degree of Master of Science

Abstract

Precise reorientation and repositioning of sliding objects presents an interesting control problem. It is necessary to understand these control issues in order to perform several robotic manipulation tasks. These tasks include maneuvering sliding payloads and reorienting objects witihin the grasp of a manipulator. In each case, it may be necessary to precisely control the object's position and orientation, and so adequate models of the object's motion in the presence of robotic as well as environmental constraints are required.

Reorienting and repositioning sliding objects in a plane involves three degrees of freedom; however, only two degrees of motion are available by pushing the object at a point. Therefore a successful reorientation generally requires compound motions. The planning issues regarding the determination of suitable compound motions are an important aspect of the reorientation problem. This thesis addresses the planning problem, and presents several reorientation strategies involving compound rotations of a sliding object.

An analysis of sliding-object motion in the presence of frictional contact is also presented. The analysis leads to linearized models for straight-line and curvilinear motion of sliding objects. The models are used to derive feedback control strategies for controlling the motions.

Simulations of the controller designs confirm the validity of the models and provide insight into the stability characteristics of the system. Results show that the plant can be stabilized with pure proportional feedback of the measured states. Experiments performed on a two-degree-of-freedom manipulator testbed further validate the dynamic model used and demonstrate practical sliding-object control.

Thesis Supervisor: Harold L. Alexander

Acknowledgements

Several individuals have contributed in some way or another to the success of this work. This is an opportunity to appreciate their support.

I owe much to Professor Alexander for serving both as the thesis supervisor and as a source of advice for the many technical as well as personal decisions which I have confronted along the way. Technically, he has always maintained a clear vision of the direction in which the research would serve best to follow. More importantly, he has helped me to acquire that vision without asking me to sacrifice my personal research goals. Personally, he has provided me with candid advice about school and industry, and has enthusiastically supported me in my non-technical pursuits.

I must also acknowledge the many employees of Section 347 at the Jet Propulsion Laboratory for their support during the incipient stages of this research. In particular, I am grateful to both Henry Stone and Paolo Fiorini for unselfishly taking on an inexperienced internship student and helping him to establish worthy research goals. In addition, I would like to thank Ed Barlow for giving me *carte blanche* with his tools, Zoltan Szakaly for impeccable technical support, Tim Ohm for his help in constructing the experimental testbed, and Helen Greiner for her valuable technical and moral support. Their friendships made for a generally great stay in California.

Several people in the Space Systems Laboratory have provide me with valuable technical insight. Most notably are Ali Azar and Jud Hedgecock. In addition I would like to thank Steve Crooks and Lisa Woo for the many weekend vacations in Bedford and Jean Spinelli for the many interesting conversations we had during times when I felt compelled to take a break from my work.

Last, but certainly not least, I would like to thank my parents and my two sisters for their continued support not only during this research but throughout the last five years at MIT.

Table of Contents

List of Figures

Chapter 1
Introduction

1.1 Motivation: Grasping, Pushing, Reorienting

In the context of robotic manipulation, it is important to understand the interaction between an ungrasped object and its surrounding environment because this interaction precludes any attempt to grasp or manipulate the object. During a grasping maneuver, the object is not entirely constrained by the robot until it has been fully grasped. Instead, the object's motion is governed by both the contact with the manipulator as well as the contact with the environment. Another manipulation task is to control the motion of a sliding object by pushing the object with the manipulator. Again, the objects motion is largely governed by its contact with the environment. Finally, one can consider the task of *in*-grasp reorientation, or reorientation within the gripper after the object has been grasped. For all three of these manipulation tasks, it is important to characterize the object's reaction to manipulator as well as environmental constraints. To shed further light on the problem, we consider the three basic robotic tasks and the degree to which they rely on some knowledge of an object's interaction with its environment.

1.1.1 Grasping

The most common manipulation task is that of grasping, in which the goal is to have the object completely constrained by the manipulator. A successful or stable grasp requires static stability; therefore, it is essential that during the grasping maneuver, any contact between the end-effector and the object should ultimately result in steady state grasping forces that render the grasped object statically stable. For instance, one can imagine grasping an object with a parallel-jaw gripper in which one jaw makes contact with the object before the other. In this example, it is necessary to be able to predict the object's reaction to the contact with the first jaw in order to be certain that when the second jaw makes contact with the object, the resulting grasp will be stable.

A more interesting grasping problem is that of grasping an object with a dexterous manipulator such as the three-fingered Salisbury hand [Salisbury 1985]. With this end-effector, the added dexterity allows us to use a finger to actively control the motion of the object during grasp so that the object will be positioned appropriately to be grasped by the three fingers. In short, a successful grasp requires proper passive or controlled motion of the object while the object is not completely constrained by the end-effector.

1.1.2 Pushing

Instead of grasping the object, it may be desirable to simply push it along a surface. One can imagine applications such as the automatic reorientation of objects on a warehouse floor or, on a smaller scale, the automatic orientation of small parts in an assembly line. In each of these cases, the object's motion is predominantly determined by the contact with the pushing surface and the frictional contact between the object and the surface on which it is sliding.

1.1.3 Reorienting

By reorientation, we are referring to the specific category of in-grasp reorientation, since reorientation by means of pushing a sliding object on a table is included in the previous section. An in-grasp reorientation can be useful in order to achieve a more stable grasp, because an alternative orientation is more conducive to certain workspace constraints.

One category of in-grasp reorientation is reorientation involving a multi-fingered gripper. With this type of end-effector, a grasped object's orientation can be changed by simply moving grasp points to different locations, maintaining a stable grasp at all times. Fearing [Fearing 1986] has studied this problem quite thoroughly. In our study, however, we are primarily concerned with object reorientation achieved by sliding contact between the end-effector and the object.

Another category of in-grasp reorientation involves parallel-jaw grippers. Because these grippers are of minimal dexterity, the object can not be manipulated by the end-effector alone. Instead, an external surface can provide a contact force which causes the object to slide in the grasp of the parallel jaws. For instance, a peg grasped by a parallel-jaw gripper can be reoriented by pushing the peg against a wall. This problem is, interestingly, the inverse of the case in which a robotic end-effector slides an object along an external surface. With the parallel-jaw reorientation, the gripper represents the surface and the external object does the pushing. In either case, it is once again necessary to be able to predict the object's motion in the presence of these contact frictions and forces.

1.2 Previous Work

Mason [Mason 1985], in his doctoral work, has studied the pushing of objects in some detail. Because his work serves as much of the theoretical foundation of this research, it is important in this section to address Mason's contributions in this field. In the following sections, we identify areas of the field in which our study departs from Mason's as well as areas in which our study serves as an extension of Mason's work.

One of Mason's primary goals was to provide a better understanding of manipulation by establishing a good theoretical understanding of the planar sliding motion of objects. To this effect, he derived a mathematical framework for characterizing this motion, and used this framework to focus on methods of eliminating uncertainties in the position and orientation of the object. More specifically, he demonstrated that the sliding object's sense of rotation can be predicted by inspection, thus providing methods for eliminating uncertainties without sensory feedback, or as stated in [Mason 1985], without adaptive motion of the manipulator. Finally, Mason developed a set of manipulator operation primitives which, unlike those commonly used, are useful for operations involving uncertainties in the object's position and orientation.

1.3 Problem Statement

The goal of this thesis is to develop mathematical models of the planar motion of sliding objects, and to use these models to design control systems for pushing objects along prescribed trajectories. It is apparent from the mathematical analysis that planar sliding-object motion can be characterized instantaneously as curvilinear motion, meaning a rotation about some instantaneous center of rotation. The control system design, therefore, is based on this representation. In addition, it is useful to consider the special case of motion along a straight line, or equivalently curvilinear motion about an infinitely distant center of rotation.

The study also addresses the general problem of object reorientation strategies, for both the case of reorienting objects on a table and the case of in-grasp object reorientation with parallel-jaw grippers. These reorientation strategies are developed in the context of the basic motions derived from the modelling.

1.4 Overview

As mentioned earlier, much of the theoretical background relating to the planar motion of objects is founded upon Mason's analysis. The analysis in Chapter 2 of planar sliding-object motion, therefore, very closely resembles that of Mason's doctoral thesis. The text of Mason's thesis is strongly recommended as a supplement to this analysis.

In our analysis of planar motion, we make many any of the same assumptions that Mason makes, primarily that friction forces obey the Coulomb model and that these friction forces dominate inertial forces. These assumptions lead to a formulation of the equations of motion of the sliding object from which we can fully characterize the object's motion in the presence of frictional contact. We are concerned in our analysis with the general case in which the pressure over the contact region is assumed to be constant.

The object motion is analyzed in the context of two different pushing constraints: fixed contact and sliding contact. Chapter 2 concludes with the development of geometric and mathematical models of sliding-object motion for tracking both straight-line and curvilinear trajectories. These models assume fixed contact at the pushing constraint

Chapter 3 addresses the hardware used to implement the control systems designed to push an object along a desired trajectory. A brief description of the manipulator and sensor designs are given along with appropriate derivations of the manipulator kinematics and dynamics.

The primary departure in our work from Mason's analysis is that we are concerned with corrective motions of the manipulator in controlling sliding-object motion. In Chapter 4, we consider the design of a closed-loop control scheme based on the models developed in Chapter 2 for pushing a sliding object along a prescribed straight-line trajectory. The design is analyzed for stability and then simulated to confirm the predicted behavior. Finally, the controller is implemented in hardware to compare the simulation with the actual plant and to judge the validity of the assumptions made during modelling.

In Chapter 5, we present several strategies for arbitrarily reorienting and repositioning an object by pushing it with a manipulator. The main emphasis of the chapter is on developing geometric tools to derive the set of curvilinear trajectories which result in a desired reorientation. The chapter also specifically addresses in-grasp object reorientation with parallel-jaw grippers.

Having outlined some reorientation strategies involving curvilinear motion, we address in Chapter 6 the design of a control system for pushing objects along curvilinear trajectories. The analysis parallels that of Chapter 4, and again includes simulation and experimental results.

Conclusions are presented in Chapter 7, along with suggestions for further work.

1.5 Contributions of this Thesis

The following are the principal contributions of this research:

- The detailed calculations of rotation center loci which provide great insight into the sliding-object motion of differently-shaped objects. In fact, it is determined from the loci that for objects with uniform pressure distributions at the sliding contact, the locus can be approximated as a straight line which very nearly passes through the mass center. This approximation allows us to be able predict the object's motion with knowledge of just the pushing point and the line connecting the pushing point and the center of mass, or the *pure translation* line. This fact is further exploited to arrive at state-space models of the motion which are applicable to a large class of object cross-sections.

- The development of dynamic models for sliding-object motion along straight-line and curvilinear trajectories.

- The design and analysis of controllers appropriate for controlling straight-line and curvilinear motion. For straight-line motion, we find that the plant can be stabilized with pure proportional feedback of the state measurements. This is possible as a result of derivative coupling between the states. With the addition of an integral term, the object can be made to track a prescribed straight-line trajectory with zero steady-state error. For general curvilinear motion, the radius of curvature acts as a disturbance input on one of the states. Once again, the plant can be stabilized with pure proportional feedback.

- The development of geometric methods for determining trajectories to arbitrarily reposition objects. These methods are classified in terms of the number of distinct curvilinear motions required to perform the maneuver. In addition, the practicality of the reorientation strategies is related mathematically to the number of degrees of freedom of the specified motion.

- Development of strategies for the in-grasp reorientation of objects within the grasp of parallel-jaw grippers— in particular, the case of a peg-in-gripper reorientation. In addition, some insight is provided into the use of local sensors, such as tactile array sensors, for this application. This discussion includes image processing issues regarding the extraction of object position and orientation information from the tactile data.

Chapter 2 Planar Sliding-Object Motion

In this chapter we introduce the fundamental equations describing sliding frictional contact during planar motion. As mentioned before, these equations are presented in the context of several important assumptions. First, the frictional forces are being modelled according to *Coulomb's Law*, which states that the tangential force of friction of a sliding object acts opposite to the direction of motion and has magnitude proportional to the normal force. The constant of proportionality is known as the *coefficient of dynamic friction*. The important realization here is that the object is assumed to be in motion. For stationary objects, the proportionality constant is higher and is referred to as the *coefficient of static friction*. While the Coulomb model does not completely characterize all the nonlinear phenomenon of frictional contact, it is sufficiently accurate for a large class of problems. For the sake of avoiding nonlinearities in the model, we will assume in our analysis that the two coefficients of friction are the same. This assumption is quite crucial, especially since motion of sliding objects can be derived from frictional force and moment equilibria. We will see in a later chapter that the presence of stiction greatly influences the control system design.

The assumption that the motion is planar refers to the fact that all applied forces and resulting frictional forces act in the plane of motion. Thus, the applied forces do not produce a moment except perpendicular to the plane of motion. In addition, we will assume that the motion is quasi-static, meaning that the frictional forces are dominant and the inertial forces are negligible. This assumption is generally valid for low velocities. In his analysis, Mason [Mason 1985] provides a more detailed argument for the validity of this assumption as he compares phase-plane plots of both dynamic and quasi-static systems. He concludes that even for velocities approaching 30 cm/sec, the dynamic trajectories agree well with the quasi-static trajectories.

Having modelled the frictional effects we can analyze the force and moment equilibria during a pushing motion in order to predict the resulting sliding-object motion. We describe the instantaneous motion of a sliding object in terms of an instantaneous center of rotation. The chapter proceeds with the results of calculations of the instantaneous center of rotation as a function of applied force direction for several differently shaped objects. These calculations are categorized by the two different types of contact constraints: fixed contact and sliding contact.

The loci of instantaneous rotation centers provides leads to a geometric description of the siding-objects behavior, from which we extract the equations of motion for the object. The chapter concludes with a complete linearized state-space representation of the plant suitable for control system design and development.

2.1 Frictional Force and Moment

The expressions introduced in this section refer to the diagram in Figure 2.1 of an object during general planar motion on a surface. At each differential element located by $\vec{x}$, we can write the increment of normal force, df_N, as

$$df_N = p(\vec{x})dA \tag{2.1}$$

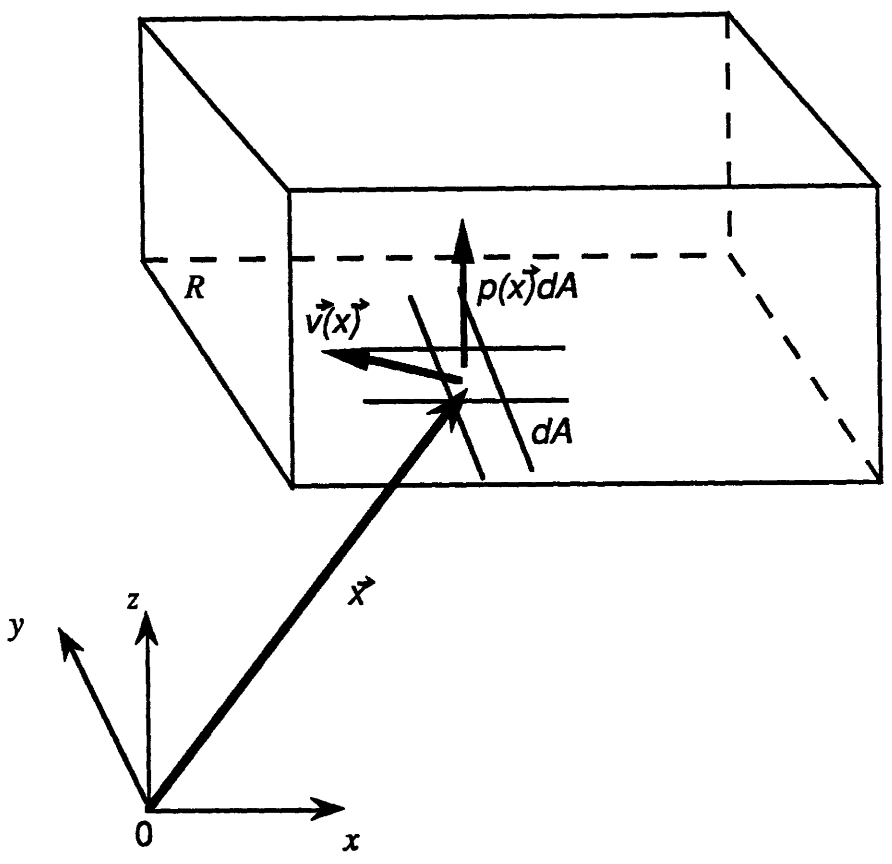

Figure 2.1 Object during general planar motion on a surface

where $p(\vec{x})$ is the presure acting at $\vec{x}$ and dA is the area of the differential element. Coulomb's Law then gives as the tangential force of friction at the differential element

$$d\vec{f}_f = -\mu\frac{\vec{v}_x}{|\vec{v}_x|}p(\vec{x})dA \tag{2.2}$$

where μ is the dynamic coefficient of friction and $\vec{v}_x$ is the velocity at $\vec{x}$. This is a force of value μdf_N acting in the direction opposed to the local velocity. In order to get the total frictional force, we integrate over the contact region R.

$$\vec{f}_f = \int_R -\mu\frac{\vec{v}_x}{|\vec{v}_x|}p(\vec{x})dA \tag{2.3}$$

The total frictional moment about the origin of the reference coordinate system is found by computing the cross product of the frictional force at each differential element dA with the distance to the origin and then integrating these differential moments over the contact region R.

$$\vec{m}_f = \int_R \vec{x} \times -\mu \frac{\vec{v}_x}{|\vec{v}_x|} p(\vec{x}) dA \tag{2.4}$$

Here, the frictional moment is a vector which points in the same direction as $\hat{k}$, the unit vector along the z-axis. In our analysis, however, we will be concerned only with the magnitude of the frictional moment. With this understanding, the vector notation will henceforth be omitted.

2.2 Planar Motion

At a given instant of time, the motion of a sliding object can be described as a pure rotation about some instantaneous center of rotation located in the plane. Pure translation is merely the unique case of a rotation about an infinitely distant center of rotation. Referring to Figure 2.2, let $\vec{x}_{cr}$ be the instantaneous center of rotation, and ω the angular velocity about this center of rotation in the the right-hand sense (thus ω is negative in Figure 2.2).

The velocity at each differential element located by $\vec{x}$ is given by

$$\vec{v} = \omega \hat{k} \times (\vec{x} - \vec{x}_{cr}) \tag{2.5}$$

from which we get

$$\frac{\vec{v}_x}{|\vec{v}_x|} = sgn(\omega) \hat{k} \times \frac{\vec{x} - \vec{x}_{cr}}{|\vec{x} - \vec{x}_{cr}|} \tag{2.6}$$

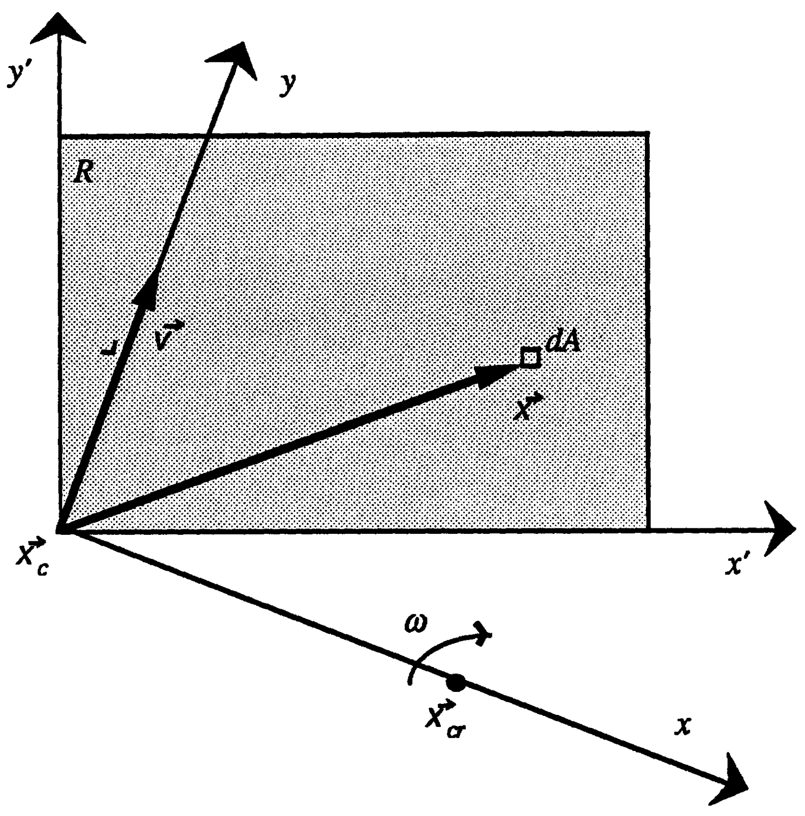

Figure 2.2 Sliding-object motion with fixed contact.

where the sgn() function is given by

$$\operatorname{sgn}(\omega) = \begin{cases} 1, & \text{if } \omega > 0 \\ -1, & \text{if } \omega \le 0 \end{cases} \tag{2.7}$$

Substituting these equations into our earlier expressions for the total frictional force and moment, we get

$$\vec{f}_f = -\mu \operatorname{sgn}(\omega)\, \hat{k} \times \int_R \frac{\vec{x} - \vec{x}_{cr}}{|\vec{x} - \vec{x}_{cr}|} p(\vec{x}) dA \tag{2.8}$$

$$m_f = -\mu \operatorname{sgn}(\omega) \int_R \vec{x} \times \left(\hat{k} \times \frac{\vec{x} - \vec{x}_{cr}}{|\vec{x} - \vec{x}_{cr}|}\right) p(\vec{x})\, dA \tag{2.9}$$

Here both the total frictional force and moment are functions of the instantaneous center of rotation. Note they are independent of the magnitude of the rotation rate, ω.

2.2.1 Fixed Contact

Suppose we represent pushing as imposing a certain velocity $\vec{v}_c$ at the contact point $\vec{x}_c$. In order to characterize the motion of the sliding object of Figure 2.2, we need only to determine the corresponding instantaneous center of rotation. Fortunately, an appropriately defined reference coordinate system yields a simple constraint equation which can be used to find the instantaneous rotation center.

Let the origin of the reference coordinate system be located at the contact point, $\vec{x}_c$. If we write the moment equilibrium equations for moments about this origin, we find that

$$\sum M_{about\,\vec{x}_c} = 0 \;\Rightarrow\; m_f = 0 \qquad (2.10)$$

since the contact force cannot exhibit a moment about $\vec{x}_c$. Again, we are assuming quasi-static motion and therefore, the inertial forces are assumed to be negligible. The above expression is referred to by Mason as the *quasi-static equation.* The root of this equation yields the desired instantaneous center of rotation. As it stands, equation (2.10) represents a vector equation in two unknowns, the x and y coordinates of the instantaneous center of rotation. The problem is underconstrained, and requires a search in two dimensions for the instantaneous center of rotation. We can further constrain the problem, however, by realizing that the vector locating the instantaneous center of rotation, $\vec{x}_{cr}$, must at any given time be orthogonal to the applied velocity vector, $\vec{v}_c$, as shown in Figure 2.2. For fixed contact, the applied velocity is presumably known a priori, thus the direction of $\vec{x}_{cr}$ is predetermined, and the two dimensional search for the instantaneous center of rotation is reduced to a one dimensional search along a line perpendicular to $\vec{v}_c$.

2.2.2 Locus of Rotation Centers for Fixed Contact Motion

Solving the quasi-static equation for a range of applied velocity directions yield the locus of instantaneous rotation centers for the object. Equation (2.10) may be solved using a Newton iteration [Strang 1986] of the form

$$x_{n+1} = x_n - \frac{f_n}{f'_n} \tag{2.11}$$

where x is the displacement of the instantaneous center of rotation along the perpendicular to the pushing direction, f is the value of the left hand side of equation (2.10) for a given value of x, and f' is the value of the derivative of f with respect to x. In these calculations, it is assumed that the coefficient of friction, μ, is unity and that the contact pressure distribution is also uniformly unity. The former assumption has no effect on the calculation—the solution of the quasi-static equation is the same regardless of the magnitude of μ. The latter assumption, however, requires some justification. In reality, one can never have an entirely uniform pressure distribution. Microscopic variations in the surface texture of a sliding object can result in significant variations in the local pressure distributions, particularly in the case of non-compliant surfaces. We will assume in our analysis, however, that there exists a compliant contact, and therefore that there is even support. This assumption does not preclude the solution of the quasi-static equation. It merely allows one to use an iteration such as the one described above to find the solution. The solution can just as well be found for an arbitrary pressure distribution (such as that sensed by a tactile sensor), although it is likely that more sophisticated (and adaptive) root-solvers will be required to cope with discontinuous pressure distributions. Mason discusses some of these different numerical approaches in his thesis, as he addresses the highly discontinuous pressure distributions associated with finite-point supports.

Figure 2.3 shows a polar plot of the locus of instantaneous centers of rotation for a uniformly dense object with a square cross section being pushed at one of its corners. The pushing angle is measured right-handedly relative to the pure translation line, while the instantaneous center of rotation corresponding to a given pushing angle θ is located a distance r along the direction perpendicular to the pushing angle. Thus, a pushing angle of 0° corresponds to pushing along the diagonal through the center of mass of the object, and the appropriate center of rotation would be located along $\theta = -90°$. In this example, the pushing angle is varied from 5° - 135° at increments of 0.5°. The centers of rotation are accurate to 5 decimal places.

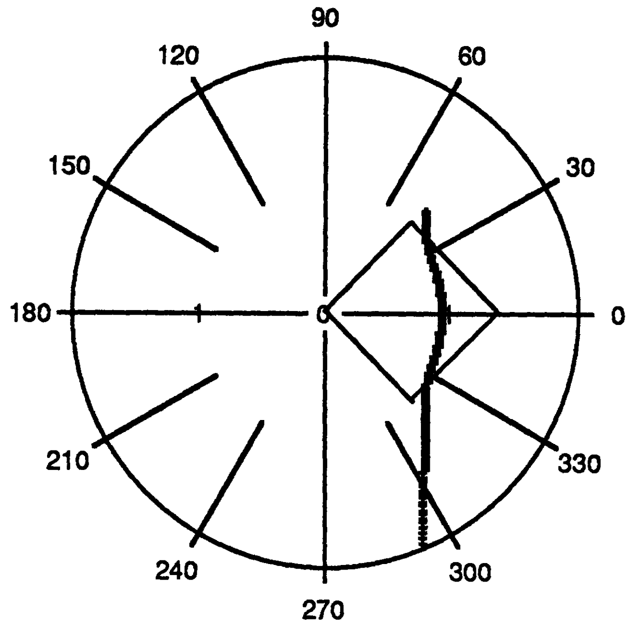

Figure 2.3 Plot of the locus of instantaneous centers of rotation for a square object.

We observe in this plot that the magnitude of the instantaneous center of rotation goes to infinity as the pushing angle approaches 0°. This is the special case in which the applied velocity vector acts through the center of mass of the object, and therefore the object is undergoing pure translation (or a rotation about an infinitely distant center of rotation). During pure translation, the total frictional force reduces to a single force acting at the center of pressure [Mason 1985]. For uniform pressure distributions, the center of pressure is equivalent to the center of mass. The pure translation line is also important in that the locus is symmetric about this line. This symmetry holds regardless of the shape of the object. We note finally that the locus cannot not pass through the center of mass.

For modelling purposes, it is convenient to approximate the locus as a straight line. At pushing angles between 0° and 45°, this approximation is quite good. The slope of the locus approximates -1 and the line approximating this part of the locus is perpendicular to the pure translation line. Figure 2.4 shows a plot of the deviation of the locus from the straight line which the locus approximates asymptotically. The horizontal axis represents the lateral displacement of the center of rotation. The vertical axis represents the perpendicular distance between the instantaneous center of rotation and the straight line approximation. Both distances is normalized with the length of the side of the square. We note again that for displacements up to 1.5 units, the deviation is small– less than 0.5% of a unit. For higher pushing angles, the deviation becomes as large as 14%.

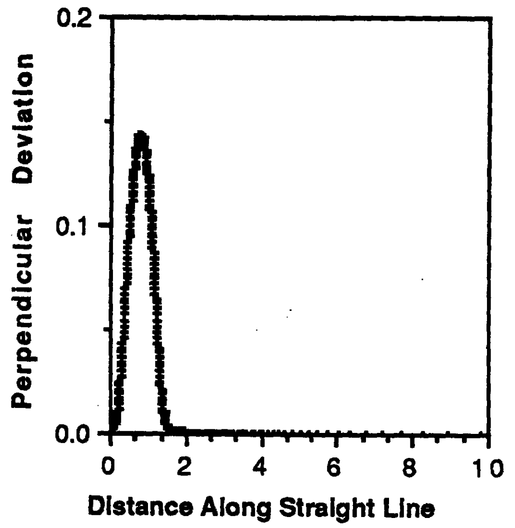

Figure 2.4 Plot of the deviation of the locus from a straight line approximation.

As would be expected, the change in the magnitude of the vector locating the instantaneous center of rotation is not proportional to the pushing angle. Figure 2.5 shows a plot of the this magnitude against the pushing angle. For pushing angles greater than 45°, the distance to the rotation center is very insensitive to changes in the pushing angle. For pushing angles between 0° and 45°, however, the center of rotation is very sensitive to changes in the pushing angles, as seen by the slope of the data approaching infinity. From a control standpoint, this is very important. In the two regimes of pushing angles, different gains would be required to effect similar changes in the location of the instantaneous center of rotation.

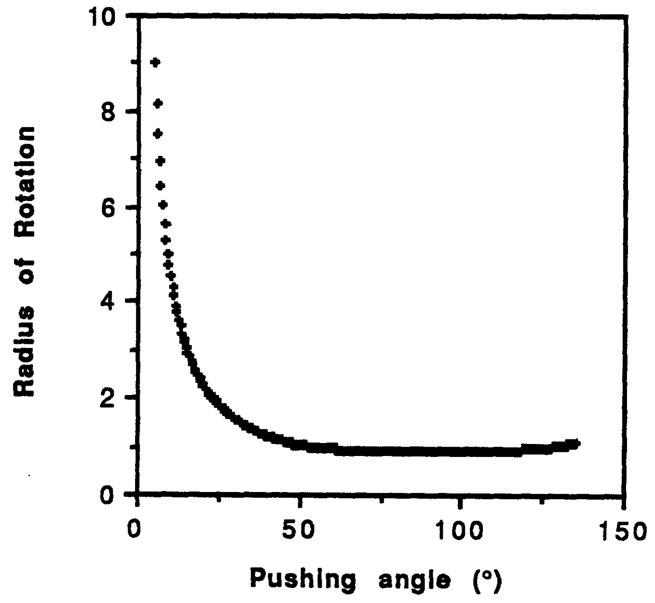

Figure 2.5 Plot of radius of rotation vs. pushing angle.

Appendix 1 shows the results of the calculation of rotation center loci for objects with differently shaped cross-sections, including rectangles of different aspect ratios, triangles, and ellipses of different eccentricities. The loci for these objects are remarkably similar, and it is noted that much of the analysis presented above for the square cross section applies equally well to these other cross sections.

2.2.2 Sliding Contact

While the forthcoming analysis will be concerned only with the case of fixed contact, it is worth discussing the calculation of the instantaneous center of rotation for a sliding contact at the pushing point since it more clearly brings into light the force and moment equilibrium in a quasi-static system. Some applications such as in-grasp object reorientation may also be constrained to employ sliding contact.

The immediate problem with the sliding contact case is that the velocity at the point of contact is no longer predetermined. The applied force, however, is known to lie on an edge of the *friction cone* as diagrammed in Figure 2.6. The friction cone is a simply geometric interpretation of Coulomb's Law. During sliding, the tangential force of friction is directly proportional to the normal force at the point of contact,

$$f_T = \mu f_N \tag{2.12}$$

The angle between these two components of the contact force is then given by

$$\begin{aligned} \alpha &= \tan^{-1}\left(\frac{f_T}{f_N}\right) \\ &= \tan^{-1}\mu \end{aligned} \tag{2.13}$$

While losing a constraint on the velocity (the tangential component is now unknown), we have gained a constraint on the direction of the applied force.

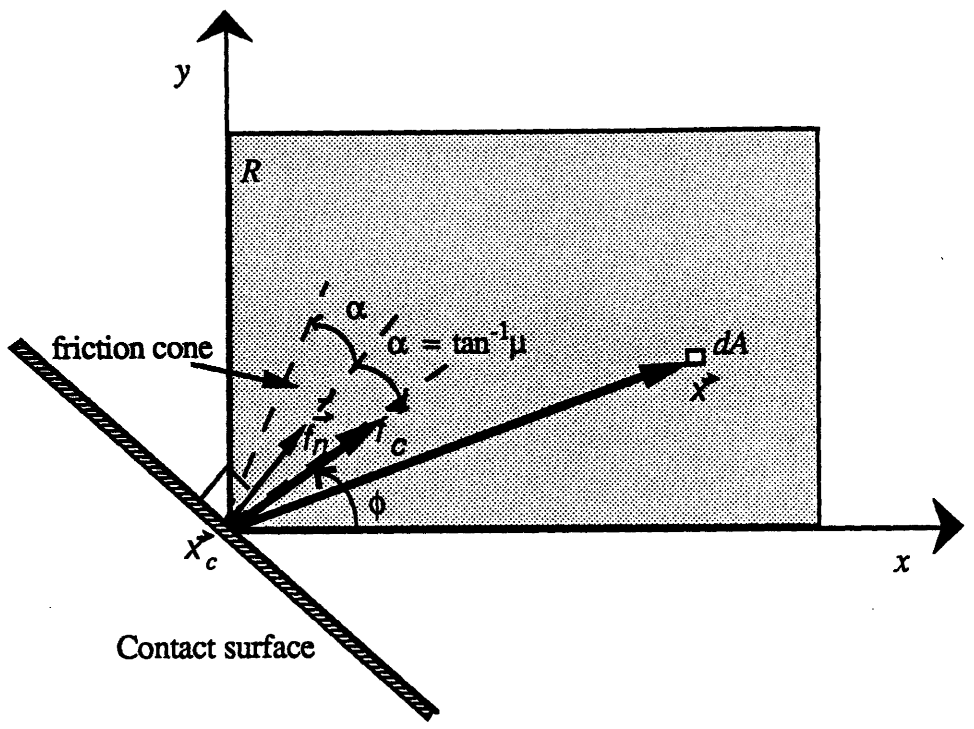

Figure 2.6 Sliding-object Motion with sliding contact.

We now consider the equations of motion for the quasi-static system as derived from force and moment equilibrium.

$$\sum F = 0 \Rightarrow \vec{f}_c + \vec{f}_f = 0 \tag{2.14}$$

$$\sum M_{about\, \vec{x}_c} = 0 \Rightarrow m_f = 0 \tag{2.15}$$

We note immediately that the instantaneous center of rotation is no longer constrained to lie on a line as was the case for fixed contact when our knowledge of the velocity at the contact point constrained the instantaneous center of rotation to lie in a direction perpendicular to the applied velocity vector. Equation (2.14), therefore, is a vector equation in four unknowns: the two components, x_{cr} and y_{cr}, of the position vector locating the instantaneous center of rotation and the two components of the vector defining the contact force, $\vec{f}_c$. Equation (2.15) is a scalar equation in the two unknowns x_{cr} and y_{cr}. If we specify the direction of the contact force, ϕ, (the contact force must lie on an edge of the friction cone), we can reduce the number of unknowns to three: x_{cr} , y_{cr}, and f_c, the contact force magnitude.

$$\begin{bmatrix} f_c\cos\phi \\ f_c\sin\phi \end{bmatrix} + \begin{bmatrix} ff_x(x_{cr}, y_{cr}) \\ ff_y(x_{cr}, y_{cr}) \end{bmatrix} = 0 \tag{2.16}$$

$$m_f(x_{cr}, y_{cr}) = 0 \tag{2.17}$$

We have reduced the system to three equations and three unknowns. The simultaneous root of these equations yields the location of the instantaneous center of rotation as well as the magnitude of the contact force which just cancels the total frictional force.

We can numerically compute the simultaneous root of these equations using a vector form of the Newton iteration described previously [Strang 1986]

$$\vec{x}_{n+1} = \vec{x}_n - J_n^{-1}\vec{g}_n \tag{2.18}$$

where $\vec{x}$ represents the vector of unknowns, x_{cr} , y_{cr}, and f_c, and $\vec{g}$ is the vector function representing the values of the left-hand side of equations (2.16) and (2.17) for a given $\vec{x}$. J is the Jacobian matrix of the partial derivatives of $\vec{g}$ with respect to the the components of $\vec{x}$. Using this iteration, the instantaneous center of rotation can generally be computed to an accuracy of 4 decimal places with convergence occurring in about 4 seconds, or 5 steps.

It is important to understand here that the shape of the locus of centers of rotation for the sliding contact case is similar to that for the fixed contact case. This not very surprising, since the contact point moves in *some* direction (though not a *specified* direction) for sliding-contact as well as fixed-contact pushing. If, then, we know the motion of the contact point, the problem reduces to the fixed contact case. Mason exploits this realization in his analysis, and provides conceptually a simpler approach to determining the rotation center for sliding contact. For all possible motions of the contact point, the corresponding instantaneous centers of rotation are calculated using the fixed contact techniques previously detailed. Given these centers of rotation, we can easily compute the associated contact force vector for sliding contact using equations (2.16) and (2.17). We now have a look-up table relating applied contact forces to the resulting instantaneous centers of rotation as well as to the motion of the contact point. The calculation here uses less storage and is more convenient since it uses the same equations as those for fixed contact.

We note finally that the locus of rotation centers for the sliding contact case is actually a subset of the fixed contact locus since not all contact point velocities can be realized with sliding contact. The realizable contact point velocities depend on the size of the friction cone, or on the coefficient of friction, μ.

2.3 Plant Model

In this section, we develop a geometric and mathematical model of the sliding object. This model represents the equations of motion for the object, and can be linearized to yield a state-space form appropriate for linear control system design and analysis.

From the derivations of previous sections, we can see that the motion of a sliding object is conveniently expressed as a pure rotation about some instantaneous center of rotation. In developing a model for this sliding motion, we begin, however, by studying the specific case of straight-line motion, or a rotation about an infinitely distant center of rotation. This problem more presents the state variables used to describe the motion. In addition, the study of straight-line motion serves to introduce the techniques used to develop more general-purpose models of sliding-object motion. In the next section, similar techniques are used to derive a model of the general curvilinear motion of sliding objects.

2.3.1 Straight-Line Motion

Figure 2.7 diagrams the straight-line motion under consideration. The object is meant to follow the trajectory defined by the line l_2. The object's current position relative to this line is given by two state variables: ϕ, which represents the angular deviation of the object (or the pure translation line, l_1) from the desired trajectory, and d, which represents the perpendicular distance from the contact point, x_c, to the desired straight-line trajectory. The angle θ is considered to be the input, or pushing angle relative to l_2. The line p is the straight-line approximation of the rotation center locus for the square object of Figure 2.7, and δ is the distance along l_1 from the contact point to the locus approximation.

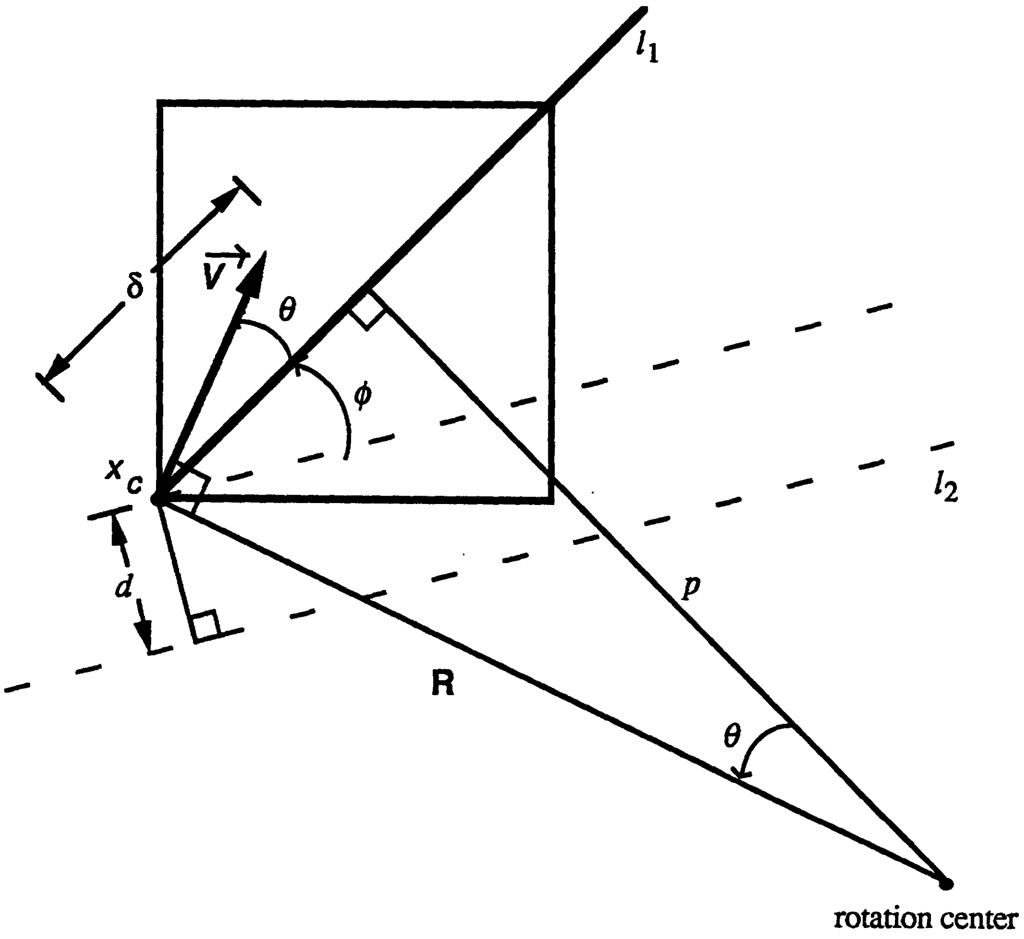

Figure 2.7 Geometric model of straight-line motion.

Having identified the state variables, we can proceed to derive expressions for the time derivatives, $\dot{\phi}$ and $\dot{d}$, of the states. First, we note that $\dot{d}$ is simply the component of the applied velocity in the direction of the line whose length is d. If we designate the magnitude of the applied velocity as V_o, we can then write

$$\dot{d} = V_o \sin(\phi + \theta) \tag{2.19}$$

In order to find an expression for $\dot{\phi}$, we note that the sliding object would complete one revolution about the current rotation center in a time given by

$$\frac{2\pi R}{V_o}$$

where R is given by $\delta/\sin\theta$. In this time, the angle ϕ has been displaced by -2π radians. Therefore, $\dot{\phi}$, which is the change in ϕ per unit time, can be expressed as

$$\begin{aligned}\dot{\phi} &= \frac{\Delta\phi}{\Delta t} \\ &= -\frac{V_o}{R} \\ &= -\frac{V_o \sin\theta}{\delta} \qquad (2.20)\end{aligned}$$

Equations (2.19) and (2.20) together represent the equations of motion for the sliding object.

Linearized State-Space Representation

In order to consider the application of linear control theory to the control of the straight-line motion described above, it is necessary to linearize equations (2.19) and (2.20) about some operating point. For a small angle x we can make the approximation that $\sin x \approx x$. Applying this linearization to our state equations with the assumption that both ϕ and θ are small, we get

$$\dot{\phi} = -\frac{V_o\theta}{\delta} \tag{2.21}$$

$$\dot{d} = V_o(\phi + \theta) \tag{2.22}$$

Rewriting in matrix notation, we get

$$\begin{bmatrix} \dot{\phi} \\ \dot{d} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ V_o & 0 \end{bmatrix} \begin{bmatrix} \phi \\ d \end{bmatrix} + \begin{bmatrix} -V_o/\delta \\ V_o \end{bmatrix} \theta \tag{2.23}$$

As a quick test of this model, we can simulate the response of the system to a constant non-zero pushing direction. This input is identical to pushing the corner of a sliding object along a straight line. The local pushing angle increases until contact is lost between the pushing surface and the object. Figure 2.8 shows the response of the system to this type of input. Here we have plotted the cross section of a square object. The asterisk indicates the pushing point. Note that this point is travelling along a straight line with slope 10°. The response of the sliding object is very much like one would expect— the begins to rotate about a distant center of rotation and as the local pushing angle increases, it "pirouettes" in a spiral. After having rotated a total of 90° relative to the line of pushing, the object loses contact with the pushing surface.

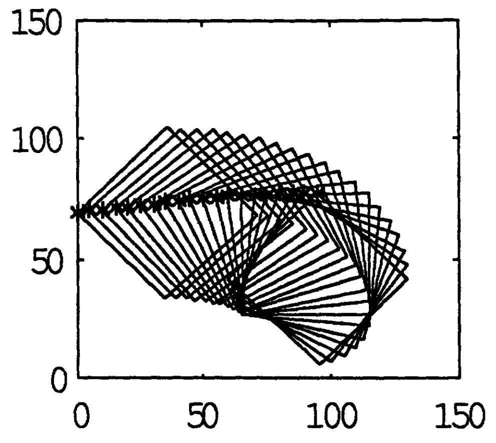

Figure 2.8 Response of system to a constant global pushing angle of 10°.

Similarly, we can observe the response of the system to a constant *local* pushing angle (i.e. a constant θ as defined in Figure 2.7). The results are shown in Figure 2.9. As we would expect, the object pushed with a constant pushing angle rotates about a constant center of rotation.

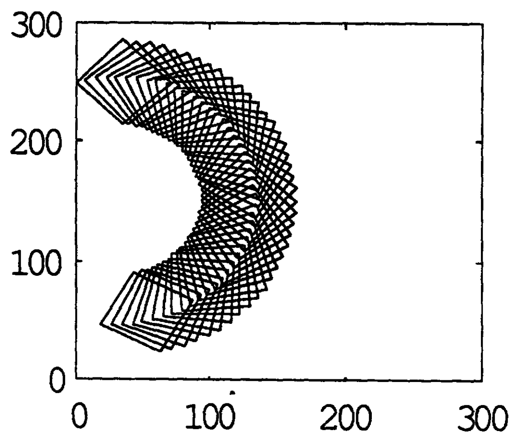

Figure 2.9 Response of system to a constant local pushing angle of 20°.

2.3.2 Curvilinear Motion

Figure 2.10 diagrams general curvilinear motion for a sliding-object. Note that the description of the objects orientation with respect to the desired trajectory is very similar to the case for straight-line motion. ϕ represents the angular deviation of the object from the desired trajectory and d represents the perpendicular distance from the contact point x_c to the desired curvilinear trajectory. The object is intended to track the arc centered at the desired rotation center of the curvilinear motion and with radius of rotation ρ. Line l_2 is perpendicular to the radius and represents the direction of the instantaneous tàngential velocity of an object tracking this desired trajectory.

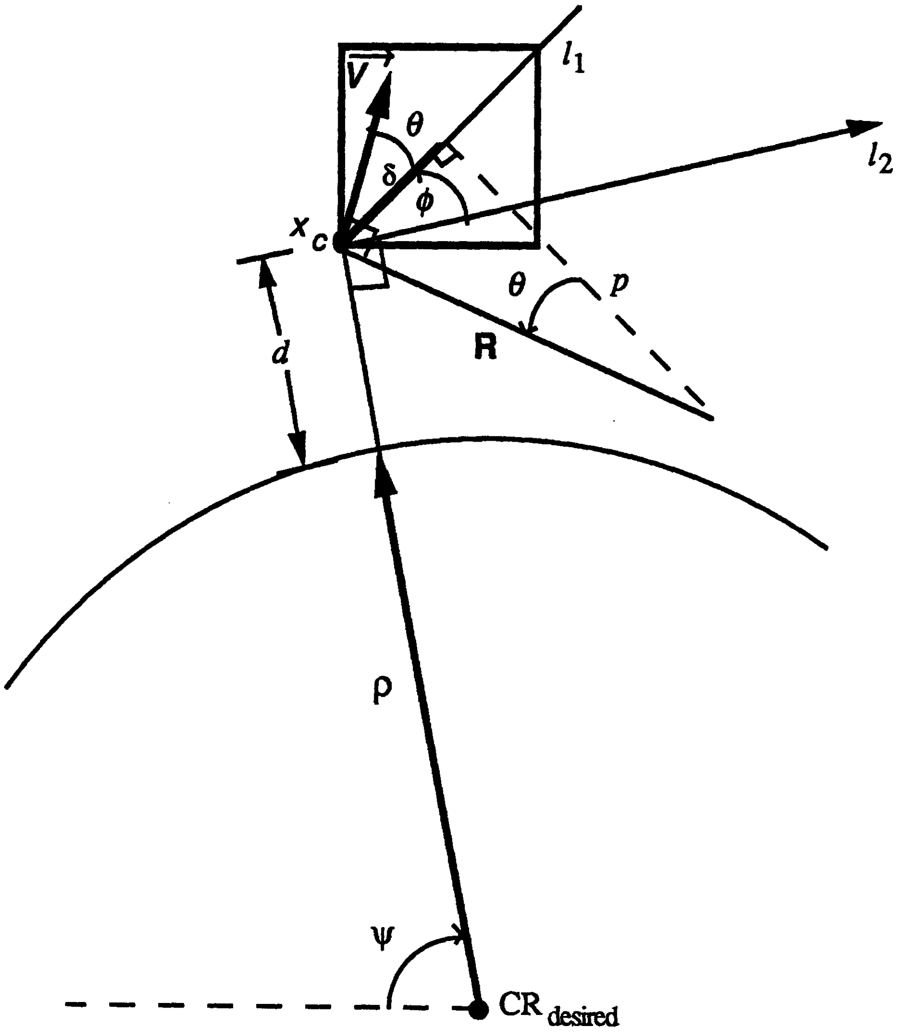

Figure 2.10 Geometric Model of Curvilinear Motion.

The state equation for d is the same as that for the straight-line case

$$\dot{d} = V_o \sin(\phi + \theta) \tag{2.24}$$

We recognize that the line to which ϕ is referenced, l_1, is no longer fixed. Instead, this line rotates about the desired rotation center with angular velocity $\dot{\Psi}$. Therefore, the change in ϕ per unit time is given by $\dot{\Psi}$ less the change in ϕ per unit time relative to line l_1. The latter term is the same as for the straight-line case, while $\dot{\Psi}$ can be expressed in terms of the total radius of rotation and the instantaneous tangential velocity in the direction given by l_1. The state equation for ϕ is therefore given by

$$\begin{aligned} \dot{\phi} &= \dot{\psi} - \frac{V_o \sin\theta}{\delta} \\ &= \frac{V_o \cos(\phi + \theta)}{\rho + d} - \frac{V_o \sin\theta}{\delta} \end{aligned} \tag{2.25}$$

Linearization

As was for the case of straight-line motion, it is useful to linearize equations (2.24) and (2.25) for control system analysis. We will once again use the linearization

$$\sin x \approx x$$

and, in addition, we have

$$\cos x \approx 1$$

and

$$\frac{1}{\rho + d} \approx \frac{1}{\rho} - \frac{d}{\rho^2}$$

The latter linearization is also derived from a Taylor series expansion about $d = 0$. After applying these linearizations to our state equations, we obtain

$$\dot{\phi} = \frac{V_o}{\rho}\left(1 - \frac{d}{\rho}\right) - \frac{V_o\theta}{\delta} \tag{2.26}$$

$$\dot{d} = V_o(\phi + \theta) \tag{2.27}$$

or in matrix notation

$$\begin{bmatrix} \dot{\phi} \\ \dot{d} \end{bmatrix} = \begin{bmatrix} 0 & -\frac{V_o}{\rho^2} \\ V_o & 0 \end{bmatrix} \begin{bmatrix} \phi \\ d \end{bmatrix} + \begin{bmatrix} -V_o/\delta \\ V_o \end{bmatrix} \theta + \begin{bmatrix} V_o/\rho \\ 0 \end{bmatrix} \tag{2.28}$$

Note that as ρ is allowed to approach infinity, equation (2.28) above becomes identical to equation (2.23). For these very large rotation centers, the curvilinear model, as would be expected, degenerates to to the straight-line model.

Chapter 3 Hardware Design and Low-Level Control

It is desirable to compare the sliding-object model derived in the previous chapter with an actual plant in order to understand the model's limitations as well as the effects of the underlying assumptions made while developing the model. A plant consisting of a robotic manipulator pushing a sliding object was therefore used to test motion control algorithms. These control algorithms are derived from the sliding-object model of Chapter 2, and are discussed in greater detail in the forthcoming chapters.

3.1 Design Requirements

In our model of sliding objects, the act of pushing is represented by a velocity imposed at a specified point of contact on the object. For planar motion, the coordinate system is two dimensional; therefore, we require a two degree-of-freedom manipulator in order to be able to move this point of contact along any direction in the plane. This is not to be confused with the object's three degrees of freedom of motion, two in translation and one in rotation about the axis perpendicular to the plane of motion. The rotational degree of motion degenerates for the case of point objects.

The manipulator is equipped with position encoders which sense the joint displacements. In addition to sensing joint positions, it is necessary to be able to determine the sliding-object's position and orientation during the pushing motion. For objects sliding on a table, this information can be determined conveniently by attaching to the tip of the manipulator a *pusher* which can rotate with the sliding object. The pusher is then connected to an encoder so that the magnitude of the object rotation can be sensed. If the manipulator always makes contact with the object, then the cartesian position of the contact point is known through the manipulator kinematics.

For manipulation tasks such as peg reorientation within a gripper, a tactile array sensor is more appropriate since it can be local to the gripper, and because the active workspace is coincident with the tactile workspace. Appendix 2 provides details on the use of these sensors. Clearly, these array sensors are not as appropriate as the local sensor described above for pushing objects on a table, a situation in which a larger active workspace is desired. For this application, the pusher described above appropriately allows the range of motion to be dictated by the manipulator workspace rather than the tactile workspace.

3.2 Design Implementation

Figure 3.1 shows a schematic of the two DOF manipulator used in the experiments. Note that the two degrees of freedom are realized by a prismatic joint and a revolute joint. While any combination of two joint types would suffice, this particular arrangement allows for a simple mechanical design as well as a convenient rectangular workspace. Figure 3.2 shows the manipulator workspace.

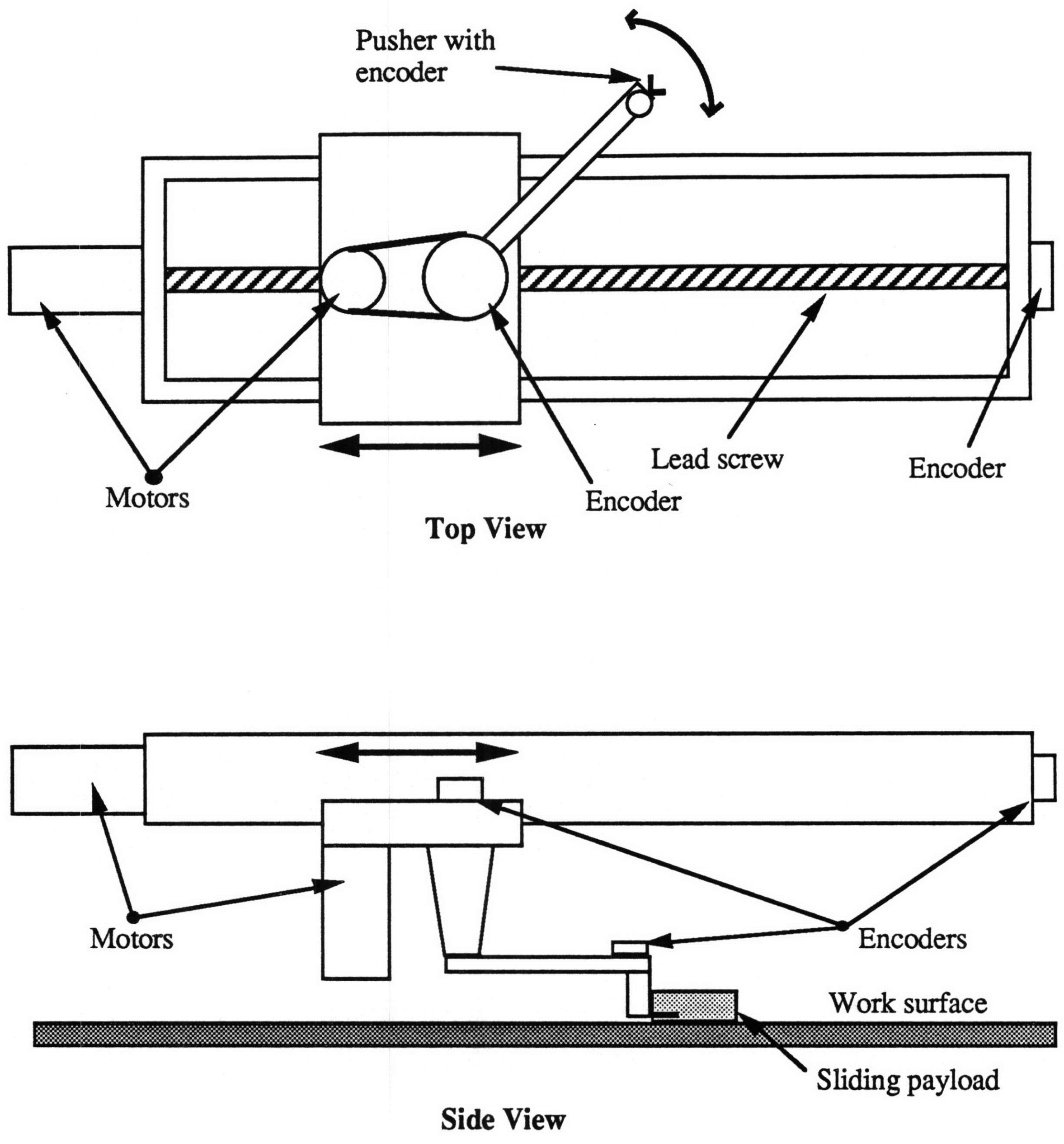

Figure 3.1 Schematic of 2 DOF manipulator.

The prismatic joint uses a lead screw drive to push a platform along teflon rails. The revolute joint is mounted on this platform and is driven by a motor via a reducing belt drive which provides amplified torque.

An encoder is mounted at the end of the lead screw drive so as to measure angular displacement of the lead screw. This rotation is easily converted to translational displacement of the platform via the pitch angle of the lead screw. A second encoder is mounted at the revolute joint (after the pulley reduction) in order to measure angular position of this joint.

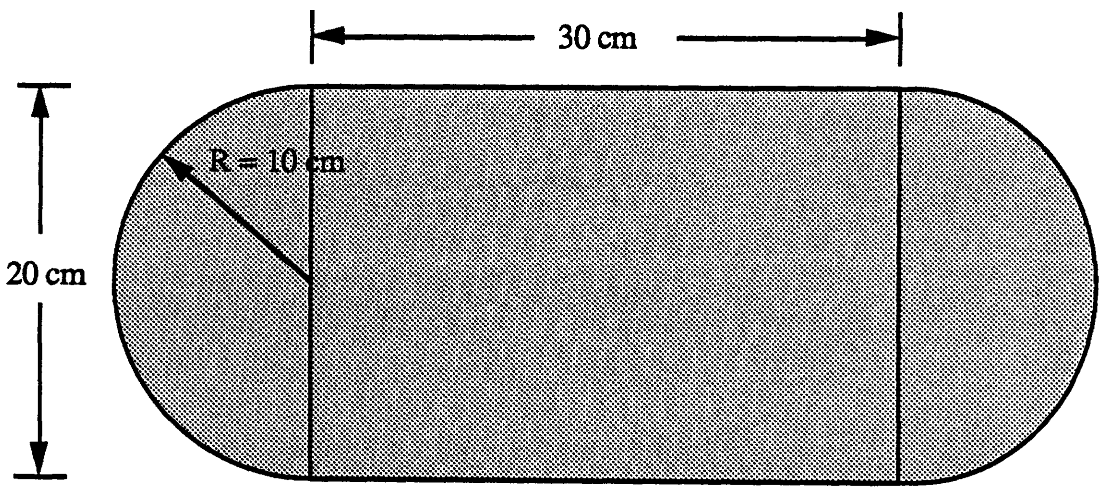

Figure 3.2 Manipulator workspace.

The two motors are driven by power amplifiers which in turn are driven by a 2-axis motion control board. The motion control board implements a PID control loop based on the encoder signals. It is also possible to operate the motion control board in a velocity mode, thereby allowing one to servo on a desired velocity instead of position. Software drivers written in C provide access to the board's functions. Appendix 3 gives a detailed description of the motion control board as well as the accompanying software drivers.

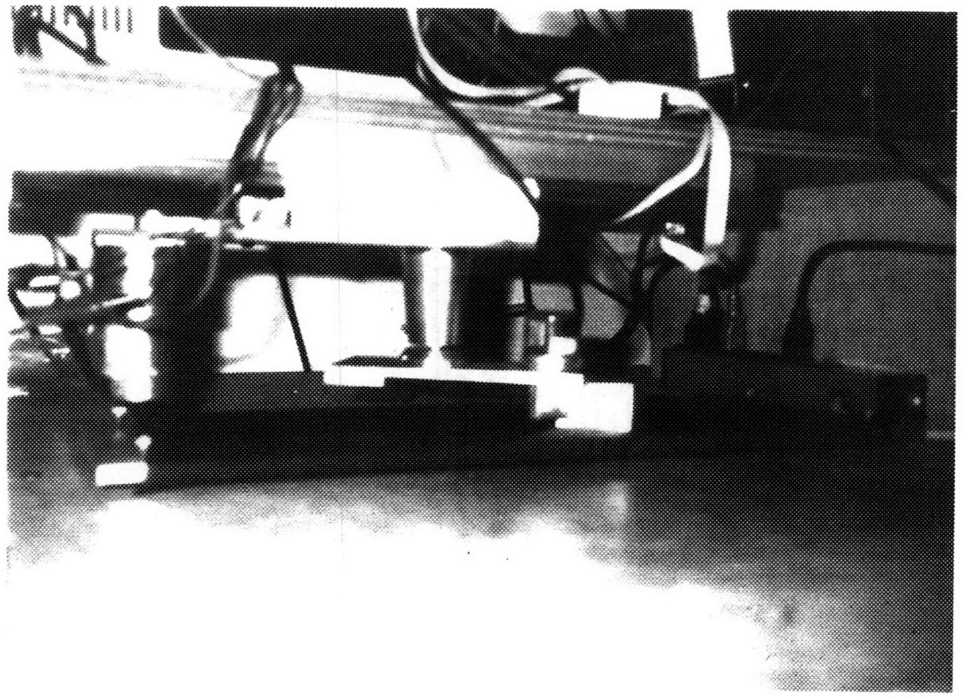

Figure 3.3 Experimental Testbed.

3.2.1 Sensing

The pusher described above consists of a shaft collar with a square notch so that the flange can capture a corner of the sliding block. The collar is coupled to the shaft of an encoder which sits at the end of the revolute link. Thus, so long as the corner of the object is always captured in the V-groove flange, the orientation of the object can be determined from the encoder count. For certain large pushing angles, it is not possible to push the object while still maintaining appropriate contact with the flange sensor (the sensor would slide off of the object). Therefore, inherent in the mechanical design of this sensor is a saturation limit on the pushing angle.

3.2.2 Manipulator Kinematics

The following derivation of the manipulator kinematics refers to Figure 3.4.

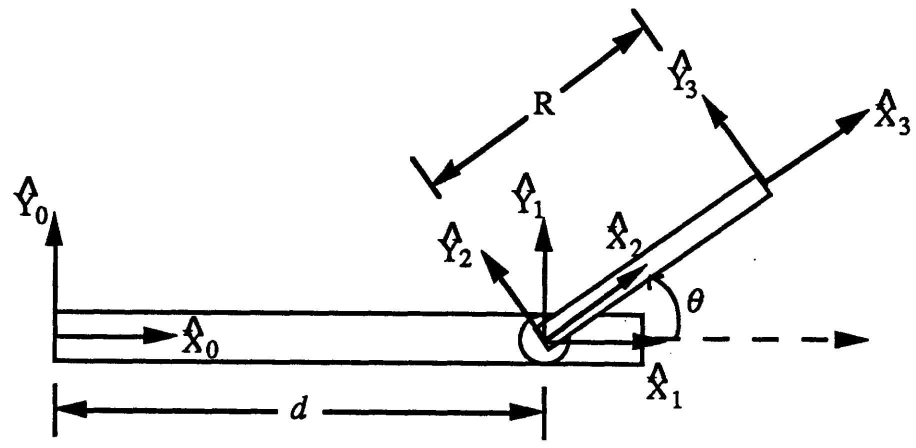

Figure 3.4 Coordinate frame assignments for manipulator kinematics.

The purpose of this calculation is to to obtain a transformation relating cartesian coordinates defined in frame 3 to the same coordinates defined in the base frame, frame 0. For the two degree-of-freedom manipulator, this transformation is quite straightforward. The prismatic joint contributes a translation d along $\hat{X}_0$ while the revolute joint contributes a rotation around $\hat{Z}_1$ by θ. The length of the revolute link is a fixed translation, R. In the familiar notation used by Craig [Craig 1986], we have therefore

$$
{}^0_2T = \text{TRANS}(\hat{X}_0, d)\,\text{ROT}(\hat{Z}_1, \theta)\,\text{TRANS}(\hat{X}_1, R)
$$

$$
= \begin{bmatrix} \cos\theta & -\sin\theta & 0 & R\cos\theta + d \\ \sin\theta & \cos\theta & 0 & R\sin\theta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

3.2.3 Manipulator Dynamics

We would also like to relate cartesian velocities of the tip of the manipulator to the appropriate joint velocities. The necessary relation is the Jacobian, and is defined by the expression

$$
{}^i v = {}^i J(\Theta)\,\dot{\Theta} \tag{3.1}
$$

where v is the 3 X 1 vector of planar cartesian velocities in frame i,

$$
\begin{bmatrix} v_x \\ v_y \end{bmatrix}
$$

and Θ is the 2 X 1 vector of joint velocities

$$
\begin{bmatrix} d \\ \theta \end{bmatrix}
$$

The Jacobian matrix is computed by propagating link velocities from the base frame 0 to the tool frame 3. The Jacobians for our 2 degree-of-freedom manipulator expressed in the tool frame as well as the base frame are given by

$$ {}^{3}J = \begin{bmatrix} \cos\theta & 0 \\ -\sin\theta & R \end{bmatrix} \tag{3.2} $$

$$ \begin{aligned} {}^{0}J &= {}^{0}_{3}R\,{}^{3}J \\ &= \begin{bmatrix} 1 & -R\sin\theta \\ 0 & R\cos\theta \end{bmatrix} \end{aligned} \tag{3.3} $$

Note that the Jacobian has been written as a square matrix, indicating that the angular velocity of the tip has been excluded. This choice is justified by the fact we are only concerned with the translational velocity of frame 2. In addition, it will be necessary to invert the Jacobian in order to relate joint velocities to cartesian velocities, in which case the square matrix is necessary.

Note that the determinant of the Jacobian matrix is zero when $\theta = 90°$. The manipulator exhibits a singularity in this position, since a degree of freedom is lost in the motion of the tip. The tip can move only in the direction of the prismatic joint.

3.2.4 Cartesian Velocity Controller

In the plant model of Chapter 2, the input is a pushing angle referenced to the tool frame. Given an initial velocity magnitude, this input then completely specifies the velocity of the contact point. In order to effect these desired control motions, we must be able to express the velocity of the contact in joint space, or in terms of the joint velocities required to generate the specified cartesian velocity at the tip.

In the experiment, a joint based controller is implemented in order to provide control of the cartesian velocity of the tip. A block diagram of this controller is given in Figure 3.5.

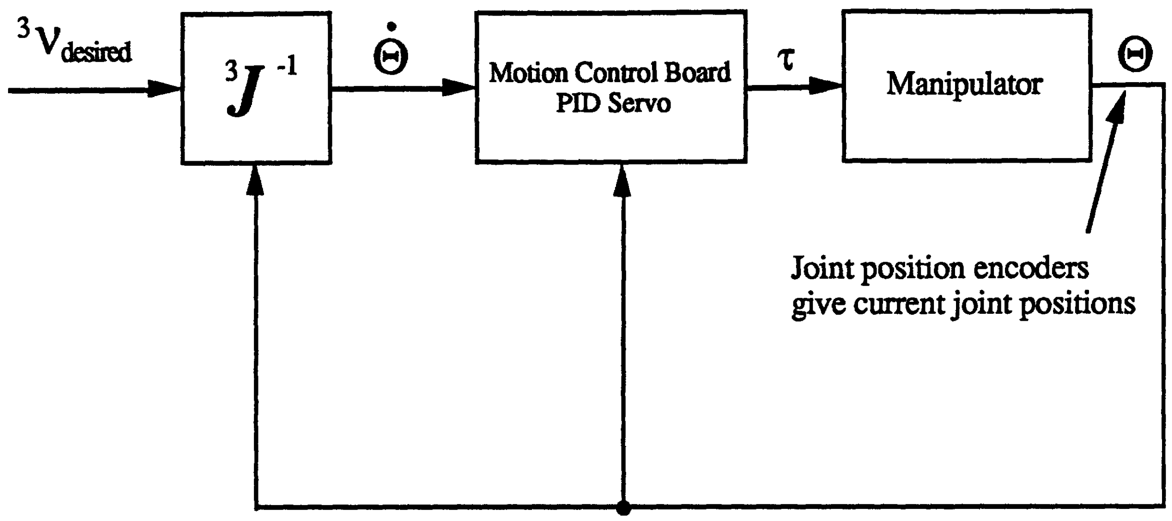

Figure 3.5 Block diagram of joint-based cartesian velocity controller.

Given a desired cartesian velocity described in the tool frame, the controller uses the inverse Jacobian, ${}^3J^{-1}$, to relate these velocities to the required joint velocities. These joint velocities then serve as inputs to the PID servo generated by the motion control board. We have a joint based controller because the input to the PID filter is a set of joint position errors. An alternative approach would result in generating errors in cartesian space and then generating control signals based on these errors; however, because the motion control board servos on the position dictated by the encoders mounted at the joints, this approach is not possible. Also, as Craig notes [Craig 1986], the joint velocities, in general, are not computed using the inverse Jacobian relationship since this computation is typically very expensive. Rather, a finite difference calculation is performed on the absolute joint positions in order to obtain a first order approximation of the velocities. Fortunately, the Jacobian of our two degree-of-freedom manipulator is easily expressed with a 2 X 2 matrix whose inverse can be computed analytically. Therefore, our joint based controller does perform the inverse Jacobian calculation in order to obtain the required joint velocities.

3.2.5 Work Surface

An important assumption made during the calculation of rotation center loci for sliding objects was that the pressure distribution throughout the region of contact is constant. It is important that the experiment simulate these conditions, since even a small concentration of pressure somewhere in the region of sliding contact can result in a markedly different locus of rotation centers.

The sliding object used in the experiments is a square uniform aluminum block whose surfaces have been flat-lapped so as to achieve as much smoothness as possible. It slides on a flat-lapped aluminum plate whose surface is covered by felt. The felt provides for the necessary compliance needed to further assure an even pressure distribution.

The point of contact between the manipulator and the sliding block is maintained close to table level in order to simulate planar motion.

Chapter 4
Control And Experiments With Straight-Line Motion

In this chapter, we use our plant model for nearly straight-line motion of sliding objects in order to arrive at control schemes for maneuvering an object along a prescribed straight-line trajectory. The approach is to analyze the stability characteristics of the open-loop plant, and then to consider different compensators which can be used to shape the closed-loop response of the system. The closed-loop system is simulated in software in order to observe the stability characteristics of the various compensators. In addition, this type of simulation is used both to test the validity of the linearizations used in our derivation of the state equations and to incorporate other nonlinearities in our model, such as input saturation. Finally, the control system is implemented with the hardware. The results of the software simulations can then be compared to the response of the actual plant. The chapter concludes with a discussion of the differences between the software simulation and the experimental results.

4.1 Controllability and Observability

We recall from equation (2.23) the linearized state-space representation of the plant

$$\begin{aligned}\begin{bmatrix}\dot{\phi}\\ \dot{d}\end{bmatrix} &= \mathrm{A}\begin{bmatrix}\phi\\ d\end{bmatrix} + \mathbf{b}\theta\\ &= \begin{bmatrix}0 & 0\\ \mathrm{V_o} & 0\end{bmatrix}\begin{bmatrix}\phi\\ d\end{bmatrix} + \begin{bmatrix}-\mathrm{V_o}/\delta\\ \mathrm{V_o}\end{bmatrix}\theta \end{aligned} \tag{4.1}$$

The plant is inherently unstable, with both eigenvalues of the system equal to zero. In order to stabilize the system response, the system must be controllable. A controllable system is one in which it is possible through the input to transfer the system from *any* initial state to *any* other final state in a *finite* time [Friedland 1986]. A system without this property may contain unstable or oscillatory modes which can not be damped with control inputs. Note that this analysis applies to the linearized form of the plant model. Using the algebraic condition for controllability defined by Friedland [Friedland 1986], we find for the controllability matrix of this plant

$$\mathbf{Q} = [\ \mathbf{b} \quad \mathbf{Ab}\ ] = \begin{bmatrix} -\frac{V_o}{\delta} & 0 \\ V_o & -\frac{V_o^2}{\delta} \end{bmatrix} \qquad (4.2)$$

The system is said to be controllable if the rank of **Q** is equal to the order of the system. In this case, the rank of **Q** is 2 (**Q** is nonsingular), and since our plant is second order, we have a positive test for controllability.

A system is said to be *observable* if it is possible to determine any state from a finite record of the output [Friedland 1986]. If the states cannot be determined, then it is not possible to control state behavior through control inputs. It is convenient if all states are directly measured with appropriate sensors; however it is possible to have an observable system by directly measuring only some of the states as long as sufficient coupling exists between the states. The observability of a system can be checked by performing an algebraic test similar to that for controllability [Friedland 1986]. Given as our observation vector

$$\mathbf{C} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

indicating that both of our state variables ϕ and d are directly measured, we find that the observability matrix

$$N = [\quad C^T \quad A^TC^T \;] = \begin{bmatrix} 1 & 0 & 0 & V_o \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{4.3}$$

has rank 2, equal to the order of the system. Therefore, the system is observable.

4.2 Stability Analysis

As mentioned earlier, the open-loop plant is inherently unstable. Both system eigenvalues are identically zero, and so the plant is a double integrator. The availability of complete state measurements makes the plant stabilizable with pure proportional state feedback. In the next section, we show how the system eigenvalues are related to the close-loop feedback gains on ϕ and d.

4.2.1 Pole-Placement With Full-State Feedback

In a controllable system with all the state variables accessible for measurement, it is possible to place the closed-loop system poles anywhere in the complex s plane. The technique involves pure proportional feedback of the state variables and is referred to as *pole placement.*

For the design of a regulator, in which the reference signal for the states is identically zero, the feedback law is given by

$$u = -\mathbf{kx} = -\left[k_\phi \; k_d\right]\begin{bmatrix} \phi \\ d \end{bmatrix} \tag{4.4}$$

where u is the plant input command, **x** is the 2 X 1 vector of state variables, and **k** is the 1 X 2 vector of proportional feedback gains for the state variables. The method of pole placement relates the desired output to the individual gains in the feedback gain vector **k**.

Substituting the above expression for θ into the state-space formulation of equation (4.1), we get an expression for the closed-loop system in state-space form

$$\begin{bmatrix} \dot{\phi} \\ \dot{d} \end{bmatrix} = \begin{bmatrix} V_o k_\phi / \delta & V_o k_d / \delta \\ V_o(1 - k_\phi) & -V_o k_d \end{bmatrix} \begin{bmatrix} \phi \\ d \end{bmatrix} \tag{4.5}$$

The eigenvalues of the closed-loop system are a function of the feedback gains, and can be chosen arbitrarily. Figure 4.1 shows a plot of the closed-loop system eigenvalues as the feedback gain k_ϕ is kept constant at -10.0 and the feedback gain k_d is varied from -0.4 to 0.0.

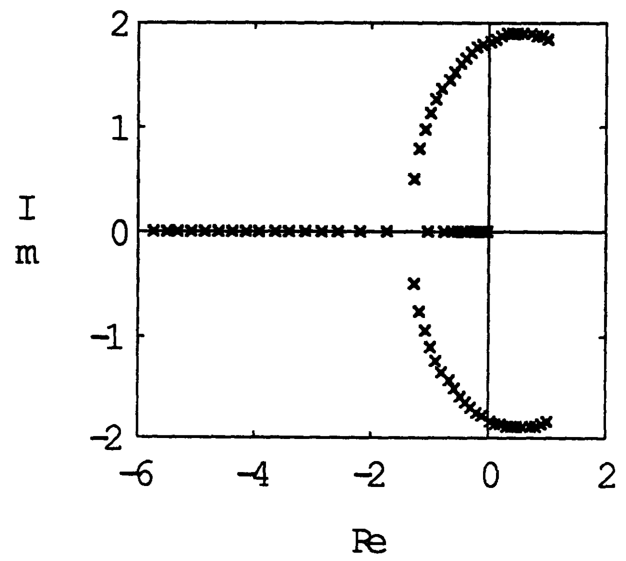

Figure 4.1 Plot of closed-loop system eigenvalues.

In the above plot, the system eigenvalues approach 0 and $-\infty$ as k_d approaches 0.0. Note that the system starts out unstable with $k_d = -0.3$. As this gain approaches zero, the system eigenvalues move into the left-half plane and then spread out at the real axis.

4.2.2 Integral Feedback

Our simulation of the closed-loop behavior of the plant will include an integral feedback for the purpose of removing steady-state error in the output. The feedback law then becomes

$$\theta = -k_\phi \phi - k_d d - k_i \int_{t_0}^{t} d\,dt \tag{4.6}$$

where the integrator is cleared at some time t_0. When we try to substitute this expression into our original state-space formulation of the plant, we realize that we need to create another state variable that we will call h to represent the integral term. The extra state equation is simply

$$\dot{h} = d \tag{4.7}$$

Rewriting the three state equations (2.21), (2.22), and (4.7) in matrix form, we get

$$\begin{bmatrix} \dot{\phi} \\ \dot{d} \\ \dot{h} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ V_o & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \phi \\ d \\ h \end{bmatrix} + \begin{bmatrix} -V_o/\delta \\ V_o \\ 0 \end{bmatrix} \theta \tag{4.8}$$

Now we can substitute the feedback law, equation (4.6), into the above equation to get our closed-loop system equations

$$\begin{bmatrix} \dot{\phi} \\ \dot{d} \\ \dot{h} \end{bmatrix} = \begin{bmatrix} V_o k_\phi/\delta & V_o k_d/\delta & V_o k_i/\delta \\ V_o(1 - k_\phi) & -V_o k_d & -V_o k_i \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \phi \\ d \\ h \end{bmatrix} \tag{4.9}$$

The eigenvalues of the close-loop system are functions of the feedback gains. We can show the result of adding integral feedback by plotting the locus of system eigenvalues as the proportional feedback gains on ϕ and d are held constant and the integral gain is varied. Figure 4.2 shows a plot of the system eigenvalues k_ϕ and k_d held constant at -10.0 and -0.1 respectively. The integral gain is varied from -0.1 to 0.0. The two complex eigenvalues are initially in the right-half plane and move into the left half plane as the integral gain approaches zero.

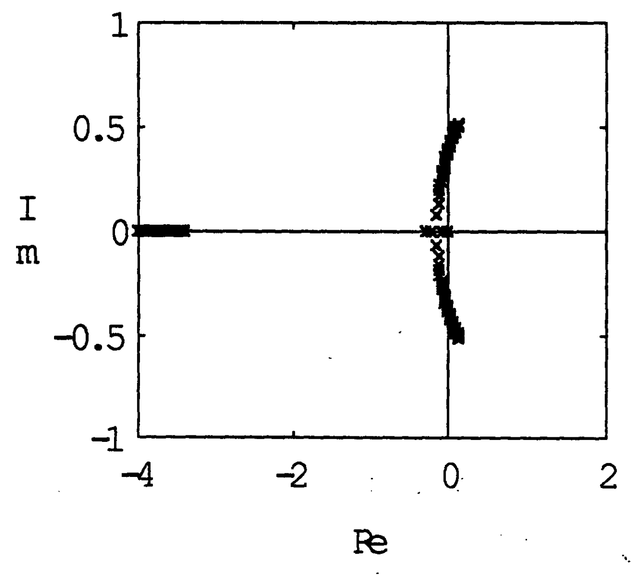

Figure 4.2 Plot of closed-loop system eigenvalues.

4.3 Compensator Design

We have analyzed in the previous section the effects of different compensators on the closed-loop system response. In this section, we consider desired output characteristics, and then design in greater detail the compensators which achieve this output. In short, we would like to relate the desired output characteristics to the compensator gains.

4.3.1 Full-State Feedback Gains

Friedland outlines an algebraic method for determining the feedback gain vector, *k*, given the desired characteristic equation of the system. The formula, known as the Bass-Gura formula, is given by

$$k = \left[(\mathbf{QW})^{T}\right]^{-1}(\hat{a} - a) \tag{4.10}$$

where **Q** is the controllability matrix of section 4.1, *a* is the column vector of coefficients of the characteristic equation of the system in descending order, $\hat{a}$ is the identical vector of coefficients of the *desired* characteristic equation, and **W** is the upper triangular matrix given by

$$\begin{bmatrix} 1 & a_1 \\ 0 & 1 \end{bmatrix}$$

For instance, if we desire that the system poles be placed at $-2 \pm 2i$ in the *s*-plane (damping ratio of 0.707), then our desired characteristic equation is given by

$$s^2 + 2s + 2$$

and the Bass-Gura formula gives for the required feedback gain vector

$$k = \begin{bmatrix} -31.54 \\ -0.7 \end{bmatrix}$$

In the above calculation, V_o is 20 units/second and δ is 35 units.

It is optimistic to think that the method of pole placement can be used to arbitrarily shape the system response. First, the method ignores of many practical issues such as input saturation. With our plant, in particular, it is not possible to have input pushing angles greater than 45° due to the design of the pusher described in Chapter 3. Attempting to create a very fast system using pole placement results in very high gains that are beyond the input saturation limits. It is also important to note that the control system gains are very sensitive to the location of the open-loop poles; therefore, slight changes in the location of the open-loop poles may result in system behavior much different from the expected behavior. Finally, pole placement can be used with a linearized model. Effects such as input saturation are clearly nonlinear phenomenon.

4.4 Straight-Line Control Simulation

Using the equations of motion for the straight-line model, we can simulate the response of the plant to prescribed inputs. More specifically, we can simulate the closed-loop behavior of the plan in response to the various compensators discussed in the previous sections. A series of plots of simulation results are presented in this section

Appendix 3 gives a listing of the code used to implement the simulation. The simulation is that of a full-state feedback controller with integral compensation. The state variables are computed at each time step by solving equations (2.21) and (2.22) with a fourth-order Runge-Kutta finite difference approximation. The input θ is computed at each time step, k, as a function of the state errors according to the following relation

$$\theta^k = k_d e_d + k_\phi e_\phi + k_i \sum_{j=0}^{j=k} e_d \Delta t \tag{4.11}$$

where k_d and k_ϕ are the proportional feedback gains and k_i is the integral gain. Because the nominal value of each state variable is zero, the state error at each time step is equal to the value of the current state. The simulation also incorporates input limiting in order to account for the fact that in the hardware implementation, the pushing angle cannot exceed 45° in either direction.

The plant is a uniform object with square cross-section and sides of length 50 mm. The asterisks in each plot refer to the pushing point at each time step. The line extending from the asterisk is a line connecting the pushing point and the center of mass of the object. It therefore shows the orientation of the object at each time step. The object is controlled to follow a trajectory along the *x*-axis. The state variable ϕ is therefore equal to the slope of the diagonal and the state variable d is the offset of the contact point in the y direction The states have been given initial values of $\phi = 20°$ and $d = 20$ mm.

The magnitude of the applied velocity V_o is 10 mm/second and the time step for the integration is 0.5 seconds. This rather large step size is not inappropriate with an accurate integration scheme such as the one used. The following plots, then, are the results of several simulations for different controller feedback gains.

<u>Simulation 1</u> : k_ϕ = -10.0, k_d = -0.07, k_i = -0.0007

Closed-Loop Poles: -.0119, -.0811, -2.0641

Figure 4.3 shows a typical stable response. Figures 4.4-4.6 show the state and input time histories corresponding to the response. Not that the system takes up to 30 seconds to settle to 5% of the steady-state value. This indicative of a pole near the origin. Also note the input saturation in Figure 4.6.

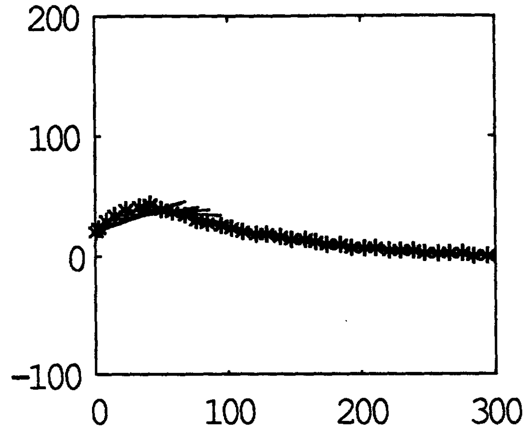

Figure 4.3 Simulation 1.

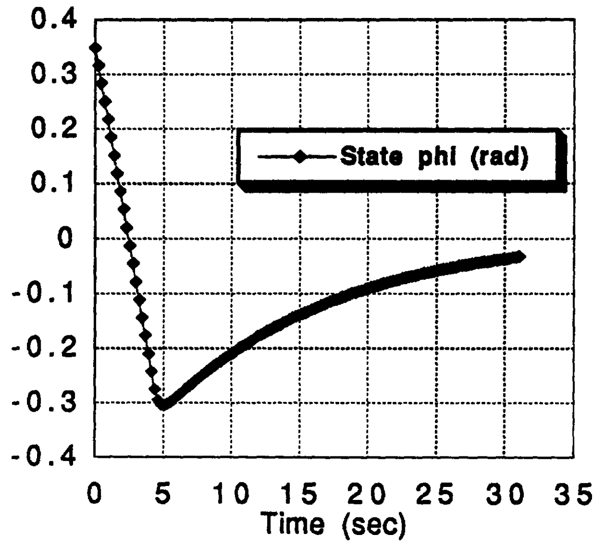

Figure 4.4 Simulation 1. Plot of state ϕ time history.

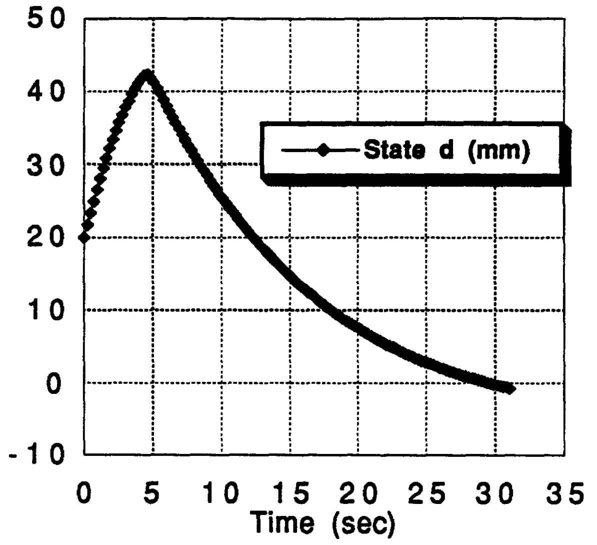

Figure 4.5 Simulation 1. Plot of state *d* time history.

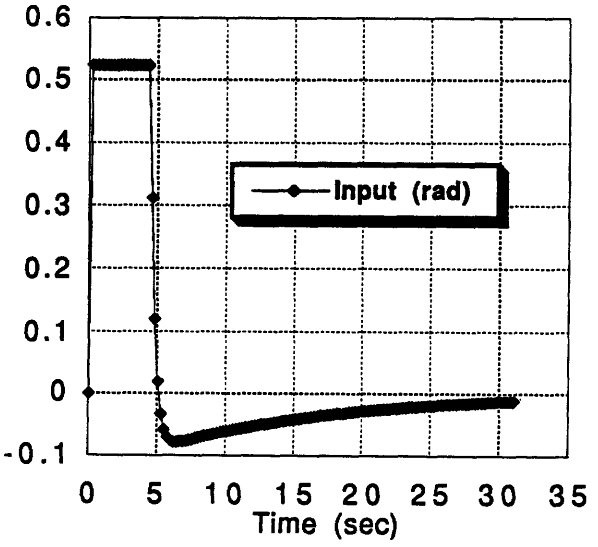

Figure 4.6 Simulation 1. Plot of input time history.

Simulation 2 : k_ϕ = -7.0, k_d = -0.07, k_i = -.0007

Closed-Loop Poles: -.0112, -.1579, -1.1309

In this simulation, the gain on ϕ has been decreased, thus having the effect of decreasing the rate feedback. The system is slightly underdamped as is evident in the small overshoot in the following plot.

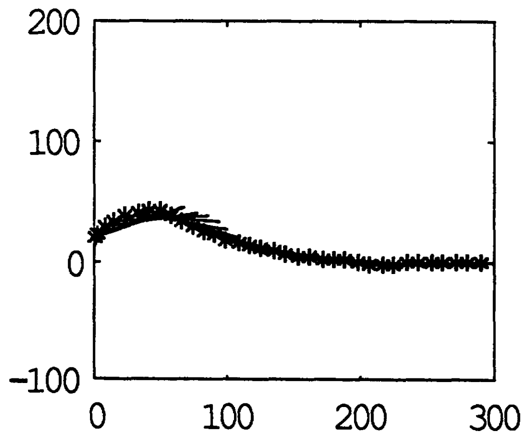

Figure 4.7 Simulation 2.

Simulation 3 : k_ϕ = -20.0, k_d = -0.07, k_i = -.0007

Closed-Loop Poles: -4.9756, -.0194 ± .0052i

The following is an example of an overdamped response. The closed-loop poles are farther apart, with two poles near the origin dominating the response. Thus, we see a very sluggish response.

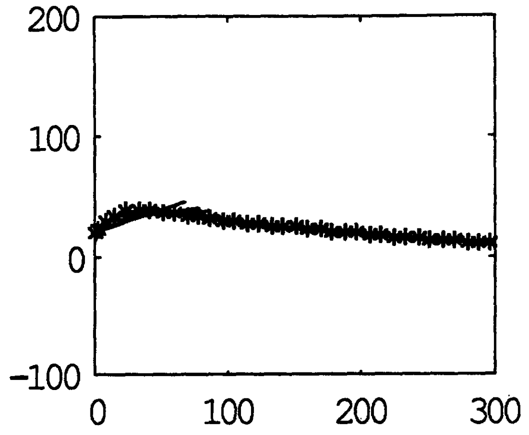

Figure 4.8 Simulation 3.

<u>Simulation 4</u> : k_ϕ = -10.0, k_d = -0.07, k_i = -.0007, Linearized equations

This simulation is identical to Simulation 1, except that the linearized representation of the plant is used. Comparing the results of this simulation with that of the first, we find that the linear model closely resembles the nonlinear model even for large deviations (up to 20°) of the state ϕ and input θ.

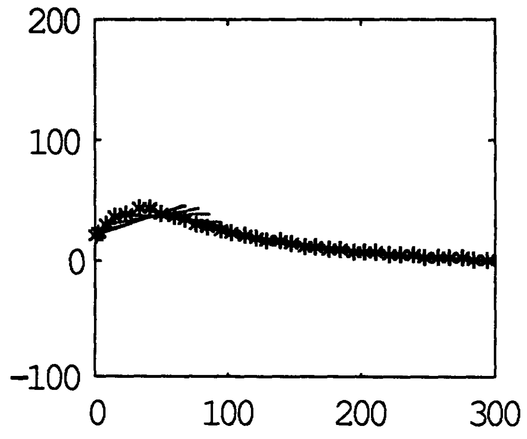

Figure 4.9 Simulation 4.

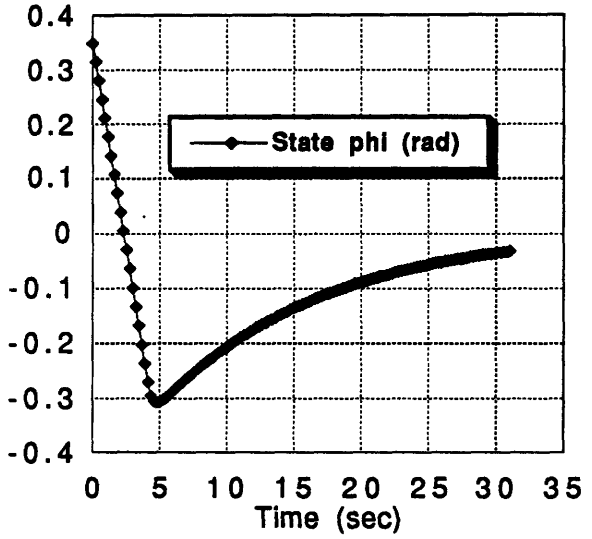

Figure 4.10 Simulation 4. Plot of state ϕ time history.

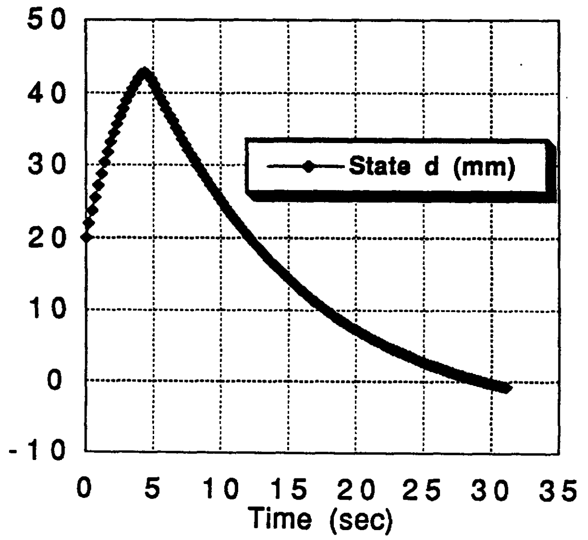

Figure 4.11 Simulation 4. Plot of state *d* time history.

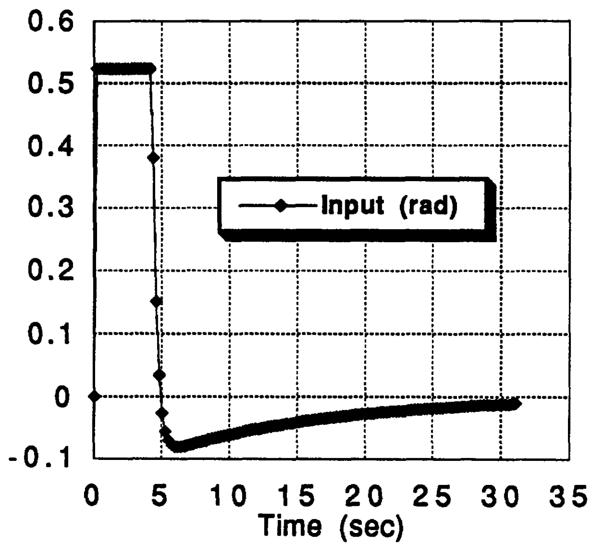

Figure 4.12 Simulation 4. Plot of input time history.

<u>Simulation 5</u> : k_ϕ = -1.0, k_d = -0.1, k_I = -.001

Closed-Loop Poles: 0.3622 ± .3897i, -.0101

The plot in Figure 4.13 is the response of an unstable system—two system poles are in the right-half plane.

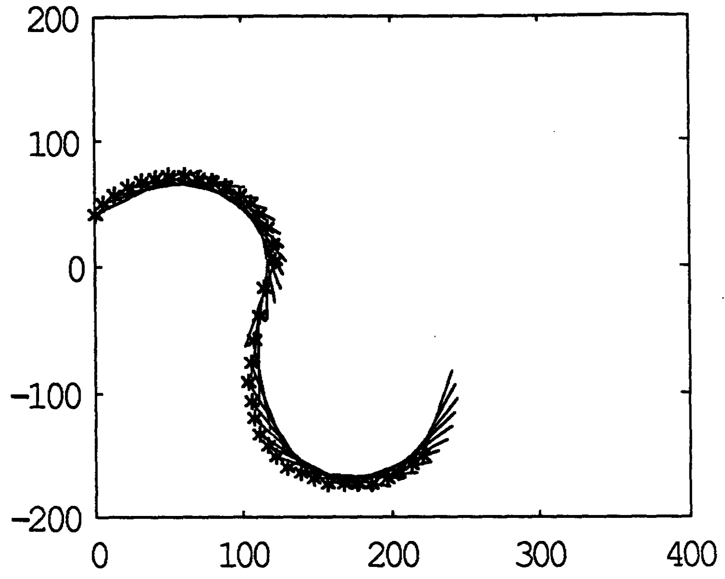

Figure 4.13 Simulation 5.

Simulation 6 : k_ϕ = **-31.54,** k_d = **-0.7,** k_I = **-.0007**

Closed-Loop Poles: -1.0052 ± .9902, -.001

Figure 4.14 shows the response of the system with gains on f and d equal to those calculated in Section 4.3.1. Note that because of the presence of an integral term, the poles are not exactly located at -2 ± 2i. The simulation exhibits much less damping than was desired during our selection of the feedback gains using pole placement. The discrepancy is largely due to the effects of saturation. As stated earlier, the method of pole placement ignores input limiting. In addition, the integral term has the effect of increasing the proportional gain, since it only feeds back integrated errors of the state *d*.

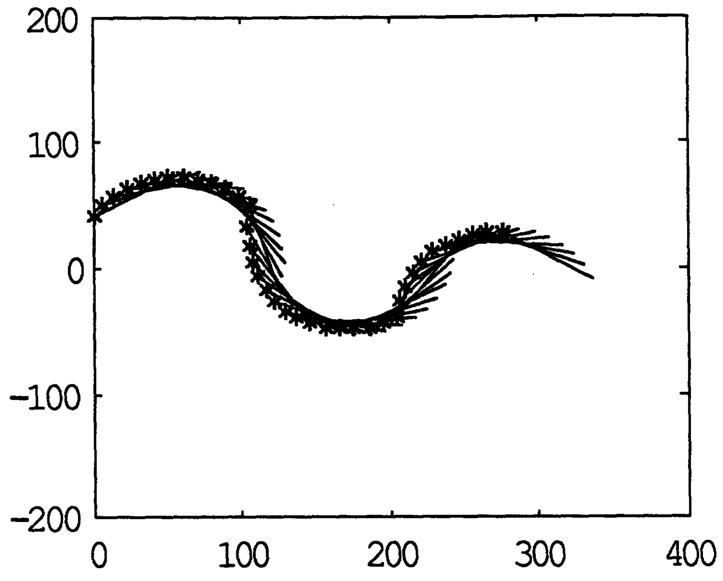

Figure 4.14 Simulation 6.

4.5 Experiments

Figure 4.15 shows a block diagram of the hierarchical control system used to simulate the straight-line control of sliding objects. An error signal is produced by comparing the desired state (represented by the state vector **x**) to the current state. This error is then fed into the transfer function of the compensator in order to obtain an input, or pushing angle. Along with the magnitude of the applied velocity, the pushing angle can be transformed into a cartesian velocity in the tool frame. This cartesian velocity serves as the input to the cartesian velocity controller described in Chapter 3. The current state is determined from both the flange sensor and current joint positions.

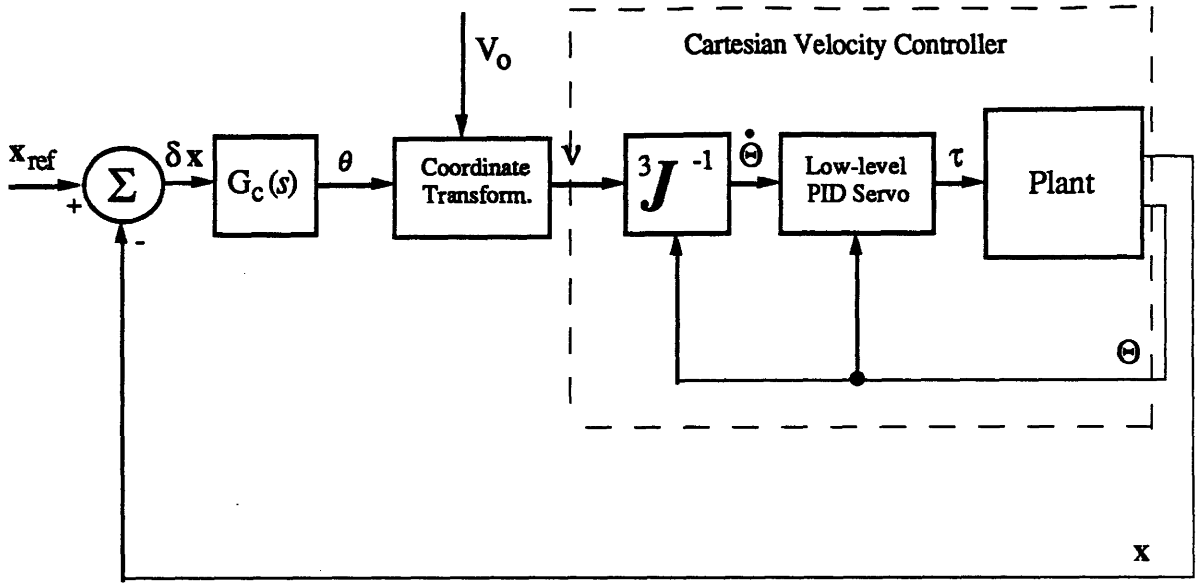

Figure 4.15 Control system block diagram for straight-line control of sliding objects.

The following series of plots show the results of experiments in which the control system is operated with the same set of gains as those in Simulations 1-3 of section 4.4. The initial state values are $\phi = 20°$ and $d = 20.0$ mm. In addition, input saturation at a pushing angle of 30° in either direction is implemented so as to make sure that the pusher which reads the current state ϕ will always make appropriate contact with the sliding object.

Experiment 1 : $k_\phi = -10.0$, $k_d = -0.07$, $k_i = -.0007$

Figures 4.16 - 4.19 show the response of the actual plant to the same set of feedback gains as in Simulation 1 of Section 4.4. We note immediately that the actual plant exhibits much less damping than is predicted by simulation. There is significantly more overshoot, but the response time is slightly faster. Also note that the input is saturated for nearly the entire time of the simulation.

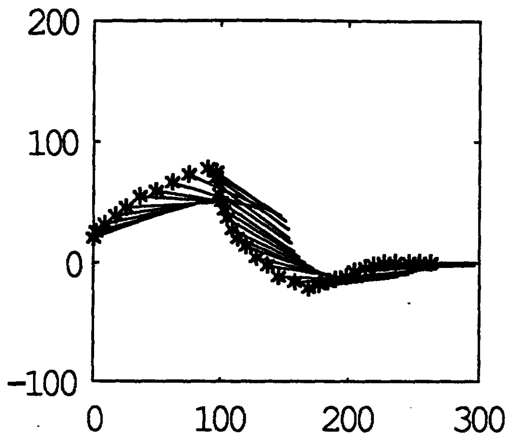

Figure 4.16 Experiment 1.

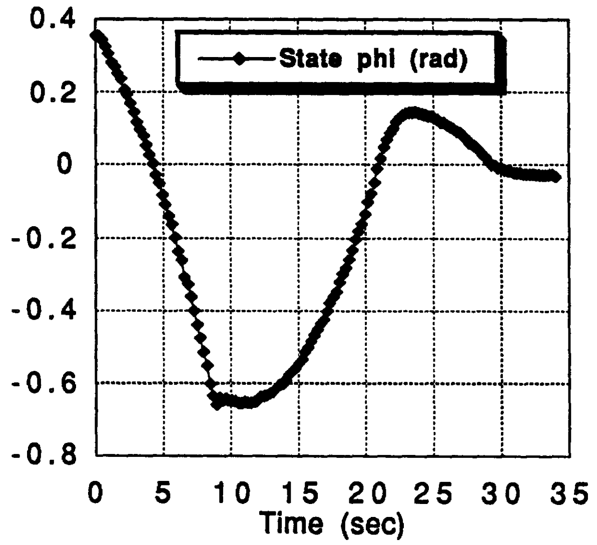

Figure 4.17 Experiment 1. Plot of state ϕ time history.

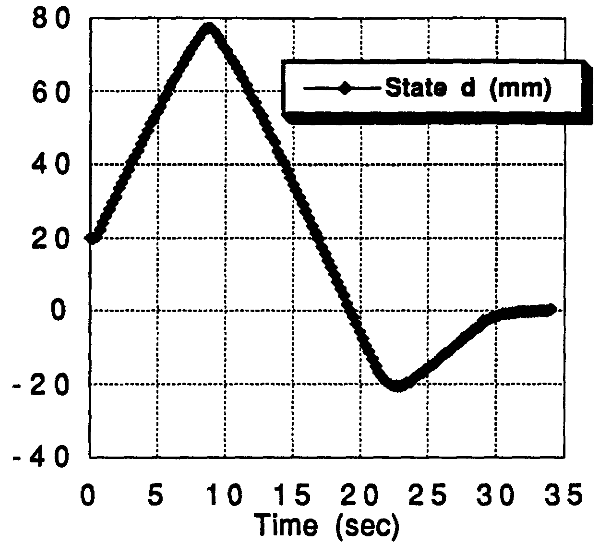

Figure 4.18 Experiment 1. Plot of state *d* time history.

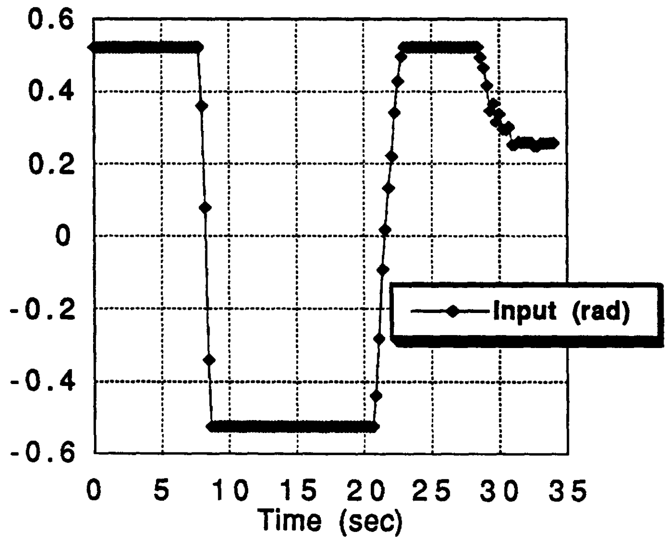

Figure 4.19 Experiment 1. Plot of input time history.

Experiment 2 : k_ϕ = -7.0, k_d = -0.07, k_i = -.0007

Once again, the actual plant response is much less damped than is predicted from simulation.

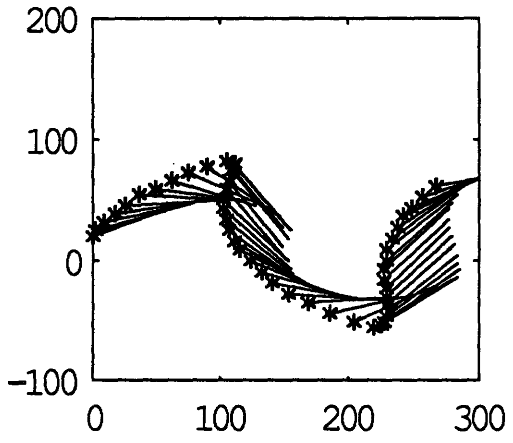

Figure 4.20 Experiment 2.

Experiment 3 : k_ϕ = -20.0, k_d = -0.07, k_i = -.0007

With more feedback of ϕ, the plant exhibits an overdamped response similar to those of Section 4.4.

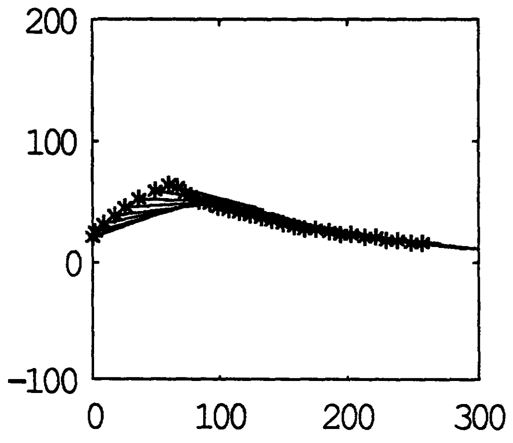

Figure 4.21 Experiment 3.

4.6 Discussion

The most apparent difference between the experimental results and the simulations of Section 4.4 is that the actual plant exhibits much less damping than would be expected. This discrepancy can in part be attributed to the fact that the pressure distribution over the contact region is not exactly uniform. As noted earlier, small changes in the pressure distribution can result in significant changes in the instantaneous center of rotation. Especially for large pushing angles, the object motion is very sensitive to changes in the rotation center.

Another source of discrepancy is that the object is pushed at a point which is not exactly on the plane of the motion. This results in moments generated about an axis parallel to the plane of motion. The effect is that the pressure distribution along the sliding contact is weighted toward one end of the object.

Chapter 5
Reorientation Strategies

Having acquired a mathematical understanding of sliding-object motion, we can begin to consider specific reorientation strategies to achieve arbitrarily specified repositioning of a sliding object. In our analysis, it will be convenient to categorize the different reorientation maneuvers by the complexity of the motions. For this purpose, it has been chosen to regard a rotation about a fixed center of rotation as the most fundamental type of motion. It follows, then, that the reorientation strategies are categorized by the number of such individual rotations required to perform the task. In addition, it is assumed that during the reorientation, the pushing surface is always in contact with the object at the same relative point on the object.

The basic problem is described in Figure 5.1. A uniform object is to be moved from an initial position and orientation *A* to a final position and orientation *B*. Since the object is constrained to move in a plane the system has three degrees of freedom: two in translation and one in rotation about the axis perpendicular to the plane of the object. Only two degrees of motion, however are available by pushing the object at a fixed point. Therefore, compound motion is generally required to achieve the desired reorientation. We can compare this situation to the problem of moving a car to the right by three feet with only the fore/aft and steering control. We require more than one motion to achieve the desired displacement.

Another important constraint which influences the reorientation strategy is that at any given time, the rotation center is limited to the locus of rotation centers. The object cannot be made to rotate about any arbitrary point on the plane. We note also that in this analysis the straight-line approximation of the rotation center locus is considered appropriate. To this effect, the following reorientation strategies have been coonsidered.

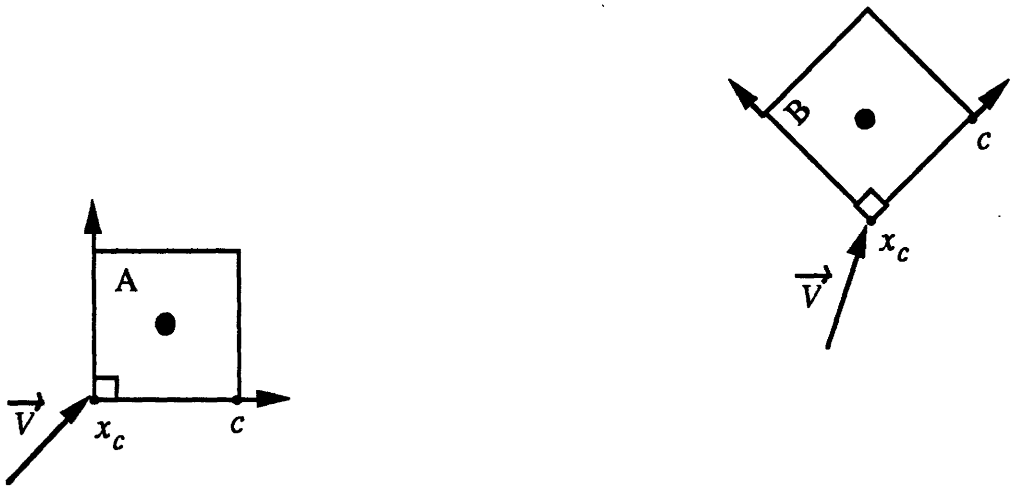

Figure 5.1 A typical reorientation scenario.

5.1 One-Step Maneuvers

By our categorization, a one-step maneuver corresponds to a single rotation about a fixed center of rotation. In practice, this type of reorientation strategy is not generally useful since it is successful only for a small class of problems, namely those in which the initial and final poses (positions and reorientations) are related by a single rotation about a center lying on the initial rotation-center locus. In general, the initial and desired states of the system will not be so related. A more mathematical understanding of these constraints comes from the inherent dimensionality of a motion about a fixed center of rotation. We can think of the trajectory marked by such a motion to be a circle centered around the rotation center with radius equal to the radius of gyration. Geometrically, the circle has dimensionality of two. Given any two points in a plane, one can define a circle with some center and radius which contains the two points. The same cannot hold true for three points, and as we recall from above, planar reorientation of an object requires at the least four initial conditions: the initial and final states of two points. Clearly, a single rotation about a fixed center of rotation cannot not achieve this.

5.2 Two-step maneuvers

Again, by our previous definition, a two-step reorientation requires two distinct rotations, each about a different, fixed center of rotation. From the dimensionality argument, we can hope that two rotations will be sufficient to perform any specified reorientation. In fact, with two rotations, we now have an underconstrained problem, and so there are more than one combination of two rotations that will generally achieve the desired reorientations.

5.2.1 Method of Arc Bisection

The first strategy revolves involves dividing the motion into two reorientations, each dedicated to positioning one of two points on the object. The maneuver is diagrammed in Figure 5.2.

The premise here is that any one point on the object can be moved to its final position with a single rotation. In Figure 5.2, the point in question is the object's rotation-center locus.[1] In order to calculate the center of rotation which displaces the center of mass appropriately, we begin by constructing the perpendicular bisector, *r*, of the line connecting the initial and final position of the locus center. The rotation center must lie on this line since any one point on the line is equidistant from the initial and final positions of the locus center. Next we compute the intersection of *r* with the locus of centers of rotation for the object, *l*. Note that the locus is represented by its straight line approximation. This approximation allows us to construct simple mathematical relations for computing the

[1] Note that here it is relatively important to use the center of the true, rather than the approximate, rotation-center locus since the two are most widely spread at the center.

center of rotation. In practice, any representation of the locus in the form $y = f(x)$ is sufficient. The point of intersection is the center of rotation we seek. A pure rotation about this point will displace the locus center of the object from its initial position to its final

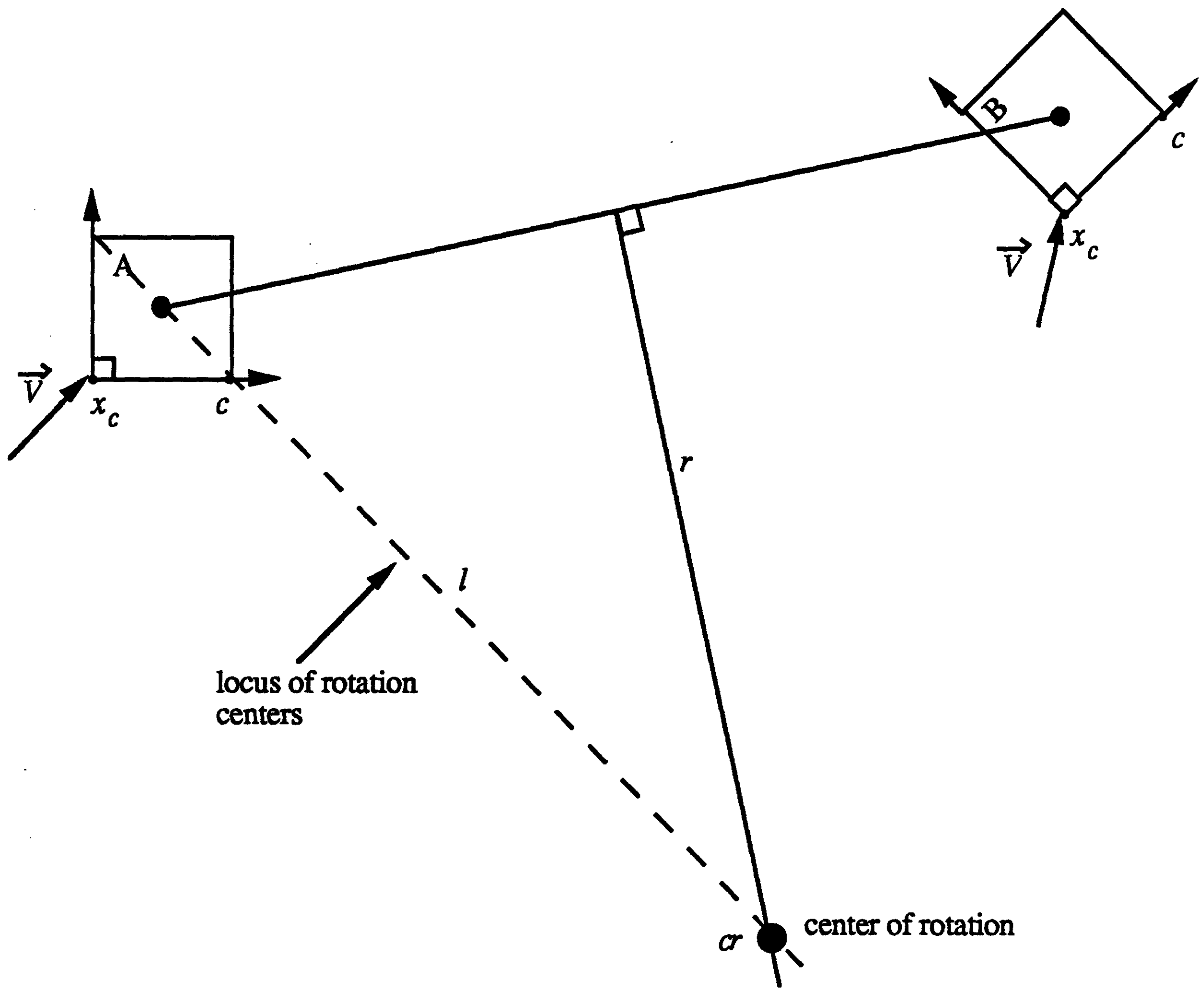

Figure 5.2 Two-step reorientation with initial positioning of the locus center.

position. Once the center of mass of the object has been appropriately displaced, a subsequent pure rotation about the locus center will orient the object as desired.

In practice, the above reorientation strategy would suffer from the center of rotation's not remaining exactly fixed. Small deviations in the pushing angle can result in large changes in the instantaneous center of rotation. One solution to this problem is to compute a new rotation center at every control step. That is, use the technique described above to compute the required center of rotation for the current state of the system. Therefore, at each time step, we are assured that the commanded motion is toward the desired position and orientation. Consistent errors in command tracking, however, may still lead to errors in the final position, and so it is valuable to seek methods that are less sensitive to control errors.

5.2.2 Method of Tangent Arcs

The following analysis is motivated by the desire to generalize two-step reorientation strategies. We require a minimum of two distinct rotations in order to arbitrarily reorient an object; therefore, it follows that all two-step maneuvers, because they utilize the minimum number of rotations, should be *geometrically* alike. It is important here to distinguish between geometric and numerical similarity. Numerically, the trajectories can be different— have different centers of rotation and different radii of gyration. Geometrically (or in shape), however, the trajectories will resemble one another. We first describe this reorientation strategy, and then discuss its underlying workings as well as its limitations.

Figure 5.3 shows the object, A, which is to be reoriented (by pushing at the corner x_c) so as to achieve the position and orientation B. Line l_1 represents the approximate locus of rotation centers which can be achieved by pushing the object A at the corner x_c. Line l_2 represents the corresponding locus for the final position and orientation B. As stated earlier, we are seeking a two-step reorientation consisting of two rotations about fixed centers of rotation. Therefore, the first center of rotation must lie along l_1, while the second must lie along l_2. For now, let's fix the location of the second rotation center CR_2 more or less arbitrarily as shown in the diagram. The second maneuver, then, must follow an arc which lies on the circle prescribed by the center CR_2 and the point given by the final position of the center of mass of the object.

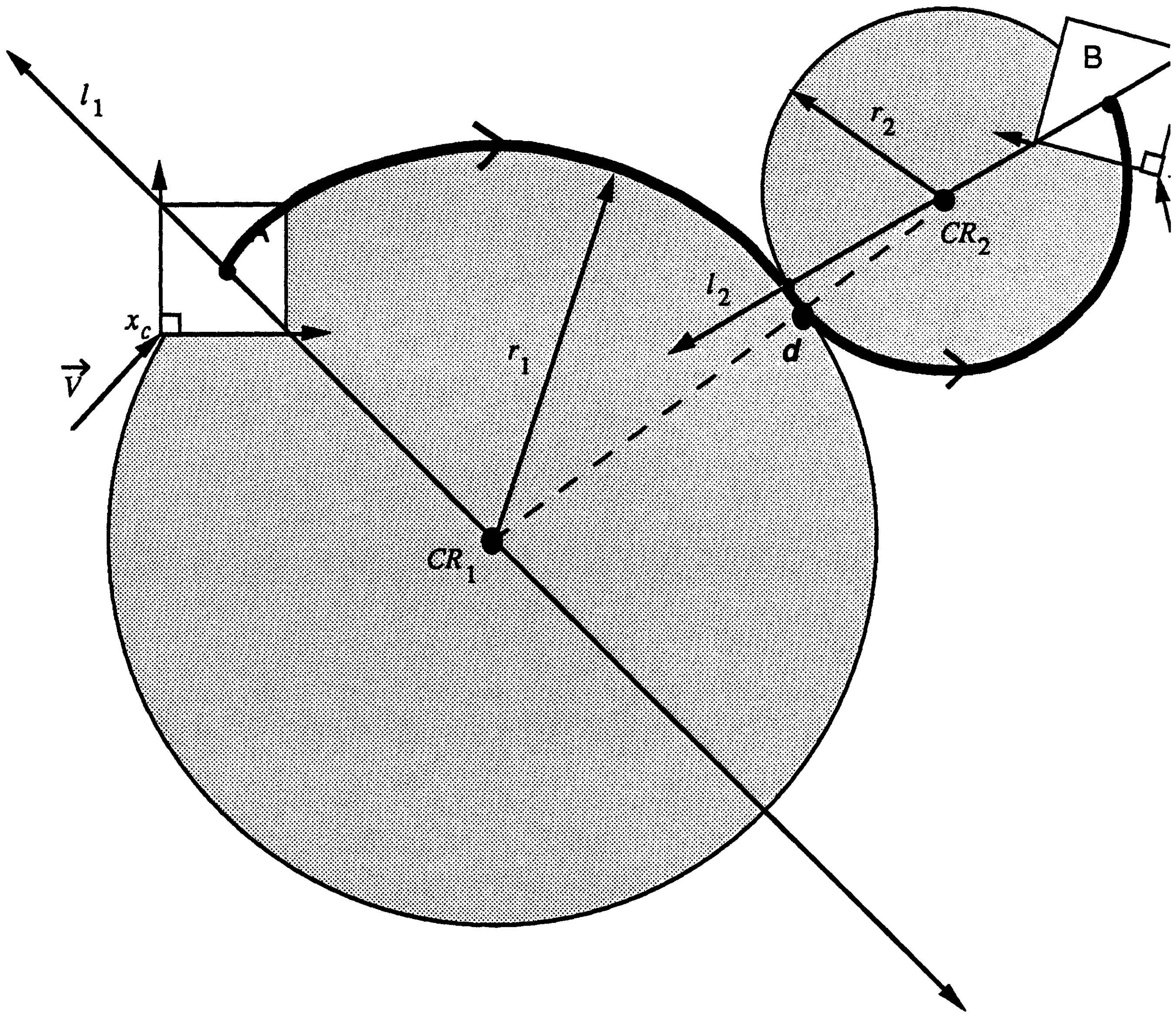

Figure 5.3 General two-step reorientation.

The first rotation is about some point on l_1. Constructing several circles which pass through the initial locus center and which are centered around a point lying on l_1, we begin to see all the possible trajectories for the first rotation. Only one of these trajectory arcs, however, is tangent to the circle describing the second rotation. This trajectory is the one which, when followed by the second rotation, will result in the desired final position and orientation of the object. The problem of finding the two motions about fixed rotation centers has, therefore, been reduced to a geometric search for the point of external tangency of two circles.

Note that the solution rested upon a predetermined second center of rotation. Without this constraint, the solutions are infinite. The problem is in fact inherently underconstrained: each circular trajectory represents, in a sense, two degrees of motion, while the problem only requires us to arbitrarily locate an object with three degrees of freedom in motion. An additional constraint (in this case the choice of r_2) is required in order to achieve uniqueness of solutions.

As stated above, the problem of defining the motions is geometric; however, we can extend the analysis further by developing a set of mathematical equations which are based on the geometric properties of the system and which will yield the desired unknowns: the two centers of rotation, the two two radii of motion, and the point of tangency of the two circles.

There are a total of 8 unknowns: the (x,y) coordinates of the two rotation centers, the two radii, and the (x,y) coordinates of the point of tangency. We require, therefore, 8 independent equations. The first two are derived from the fact that the two centers of rotation must lie on the lines l_1 and l_2, respectively. Given the equations for these two lines in slope-intercept form, we find

$$l_2: \quad y = a_1x + b_1 \Rightarrow y_{cr1} = a_1x_{cr1} + b_1 \tag{5.1}$$

$$l_2: \quad y = a_2x + b_2 \Rightarrow y_{cr2} = a_2x_{cr2} + b_2 \tag{5.2}$$

Next, we write the equations describing the two circles in terms of their centers, their radii, and the one predetermined point on each circle– the initial and final positions of the center of mass (x_c, y_c).

$$(x_{c1} - x_{cr1})^2 + (y_{c1} - y_{cr1})^2 = r_1^2 \tag{5.3}$$

$$(x_{c2} - x_{cr2})^2 + (y_{c2} - y_{cr2})^2 = r_2^2 \tag{5.4}$$

We can also write that the point of tangency, *d*, is contained in both circles:

$$(x_d - x_{cr1})^2 + (y_d - y_{cr1})^2 = r_1^2 \tag{5.5}$$

$$(x_d - x_{cr2})^2 + (y_d - y_{cr2})^2 = r_2^2 \tag{5.6}$$

In order to specifically identify *d* as the point of tangency instead of just a point of intersection, we write that the length of the line connecting the centers of the two circles (and passing through *d*) must be equal to the sum of the two radii.

$$(x_{cr1} - x_{cr2})^2 + (y_{cr1} - y_{cr2})^2 = (r_1 + r_2)^2 \tag{5.7}$$

Finally, we require an equation representing a constraint on the two radii. For instance, we might want the two radii to be equal, in which case the eighth equation is simply $r_1 = r_2$. In the more general case, we write

$$r_1 = f(r_2) \tag{5.8}$$

which determines the nature of the solutions that will be found. Alternatively, the underconstrained problem can be solved according to some optimality criterion. Equations (5.1) - (5.8) can be solved simultaneously for the eight unknowns.

Equation (5.8) presents a rather interesting problem. What factors determine this extra constraint or optimality criterion. In other words, what are some other desired properties of the reorientation? The following are a few considerations.

We can optimize the desired motion with respect to some quantity, such as energy, and therefore choose the minimum energy path. This choice might lead to the trajectory corresponding to the minimum distance travelled by the locus center of the object.

We can band the locus of rotation centers in order to avoid rotation centers which are difficult to obtain. If we calculate the derivative relating changes in pushing angle to corresponding changes in center of rotation, we will find that it goes to infinity as pure translational motion is approached. From a control stand-point, this is not very desirable, since corrective motions are reduced to the same magnitude as system noise, rendering the control system very sensitive to noise. On the other hand, rotation centers which are very insensitive to changes in pushing angle can be equally undesirable, since these correspond a system which is barely controllable, or at least one requiring very high gains for a desired response. The desired trajectory, therefore, would be one which involves motions about rotation centers that are not in either of the two regimes described above.

Referring to the specific case of the peg-in-gripper reorientation, we notice that the system has inherent hysterisis in the sense that each reorientation constrains the locus of possible future reorientations. To begin, there are certain reorientations which cannot be achieved due to mechanical constraints. That is, the gripper represents a finite workspace. In addition, once the peg has been pushed in, it can no longer be pulled out by any reorientation (all reorientations are effected by pushing the peg against a wall). In some sense, the system retains some information about past motions of the peg. This information is manifested in the constraint space of the permissible motions of the peg.

A suitable constraint on the motion of the peg would result in a maneuver which minimizes the extent to which the constraint space of future motions is jeopardized. This can be a maneuver which minimizing translations of the center of mass of the object, or minimizes the amount that the peg is engulfed by the gripper.

As a final note on two-step maneuvers, we note that the first two-step maneuver discussed in this section is merely a specific case of the general two-step reorientation strategy discussed above. More specifically, it is the case where the second radius of rotation is zero. Interestingly, this degenerate case is also the solution for the minimum-distance path, or the minimum distance travelled by the locus center.

5.3 Multi-step Maneuvers

By multi-step, we refer to maneuvers requiring rotations about more than two different rotation centers. This problem is highly underconstrained; therefore, the numerical solutions are once again infinite. This can be appreciated easily if one attempts to push a block along a table toward a final position and orientation. The block can follow any one of numerous paths over the entire workspace.

Multi-step maneuvers present an interesting optimization problem. Basically, it is possible to parameterize both the desired trajectory of the center of the mass as well as the movement of the instantaneous center of rotation. With this formulation, it is possible to generate desired trajectories meeting a variety optimization constraints. In theory, it should be possible, given, a parameterized trajectory of any two points on the object, to generate a parameterized trajectory for the instantaneous center of rotation and a parameterization of the required pushing angle.

5.4 Peg-In-Gripper Repositioning

All of the above reorientation strategies can be applied to the specific case of reorienting a peg grasped by a parallel-jaw gripper. In this section, however, we discuss an alternative strategy which exploits some of the properties of this particular physical system.

The strategy is a three-step maneuver which consists of a pure rotation followed by a pure translation and then another pure rotation. The maneuver is diagrammed in Figure 5.4.

The first pure rotation is about the center of pressure, and is effected by pushing the end of the peg in a direction perpendicular to line l_1. In actuality, one cannot achieve a pure rotation about the center of pressure; however, the assumption is very useful as the aspect ratio of the peg increases, or more specifically, the ratio between the grasped and ungrasped portions of the peg. This initial rotation is continued until the line l_1 passes through the desired center of pressure (of the final position and orientation, B). Next, the peg is pushed axially at its end until the current center of pressure slides over the desired center of pressure. Finally, the peg is rotated once again in the opposite direction until the lines l_1 and l_2 coincide.

This reorientation strategy is unique in that it minimizes the motion of the center of pressure. The drawback is that not only is it a multi-step maneuver, but also that contact is not always maintained between the peg and the pushing surface— the peg is pushed at two different corners. While translations of the peg are minimized, motions of the gripper are increased so as to provide appropriate contacts with the peg. This additional motion of the gripper is undesirable when the workspace is limited or when the workspace contains obstacles.

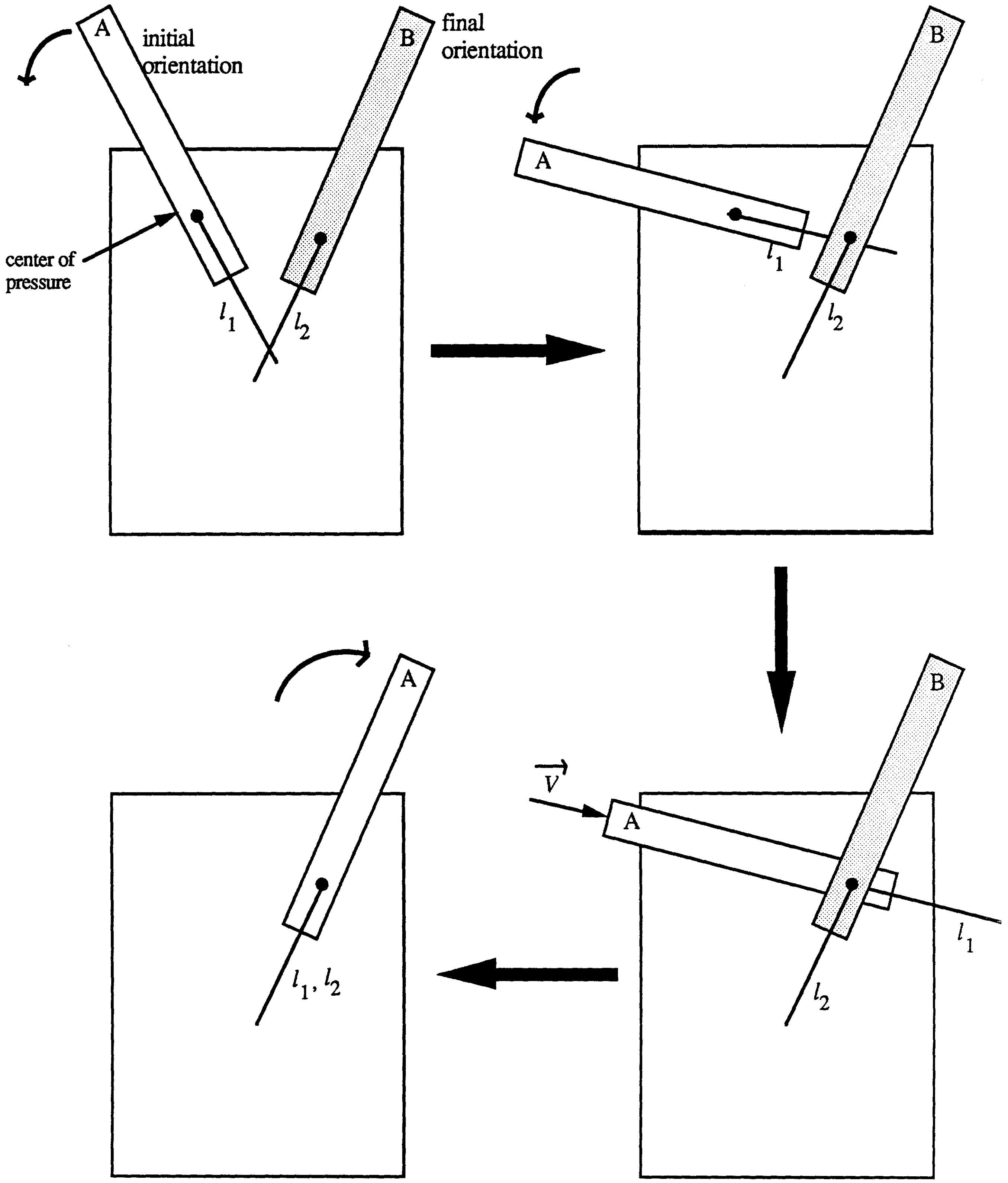

Figure 5.4 3-step peg-in-gripper reorientation.

Chapter 6 Control and Experiments With Curvilinear Motion

In this chapter, we use our plant model for the curvilinear motion of sliding objects to design control schemes for maneuvering a sliding object along a prescribed curvilinear trajectory. As in Chapter 4, we will first analyze the stability of the open-loop system, and then consider different compensator designs for stabilizing the system.

As with straight-line motion, a software simulation of the closed-loop system confirms the behavior of the various controllers. In addition, the simulation enables us to test the validity of the linearizing assumptions and to include other nonlinearities such as input saturation. Experiments are performed with the hardware in order to validate the plant model and to analyze the system response in the presence of unmodelled plant nonlinearities.

The following analysis of the control of curvilinear motion very much parallels the analysis of Chapter 4. Therefore, where a particular formula is not explicitly derived, it can be assumed that more information can be found in Chapter 4.

6.1 Controllability and Observability

We recall from equation (2.28) the linearized state-space representation of the plant for curvilinear motion about an arc of radius ρ,

$$\begin{bmatrix} \dot{\phi} \\ \dot{d} \end{bmatrix} = \begin{bmatrix} 0 & -\frac{V_o}{\rho^2} \\ V_o & 0 \end{bmatrix} \begin{bmatrix} \phi \\ d \end{bmatrix} + \begin{bmatrix} -V_o/\delta \\ V_o \end{bmatrix} \theta + \begin{bmatrix} V_o/\rho \\ 0 \end{bmatrix} \tag{6.1}$$

The eigenvalues of the system are

$$\lambda_1 = \frac{V_o i}{\rho}$$
$$\lambda_2 = \frac{-V_o i}{\rho}$$

The system is therefore marginally stable. In order to be able to stabilize the system, we first require controllability. We find for the controllability matrix of this plant

$$\mathbf{Q} = [\ \mathbf{b} \quad \mathbf{Ab}\ ] = \begin{bmatrix} \frac{-V_o}{\delta} & \frac{-V_o^2}{\rho^2} \\ V_o & \frac{-V_o^2}{\delta} \end{bmatrix} \tag{6.2}$$

The rank of this matrix is 2 and so we have a positive test for controllability.

Next we check for observability. Using an observation vector

$$\mathbf{C} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{6.3}$$

indicating that both of our state variables are measurable, we find that the observability matrix

$$\mathbf{N} = [\ \mathbf{C}^T \quad \mathbf{A}^T\mathbf{C}^T\ ] = \begin{bmatrix} 1 & 0 & 0 & V_o \\ 0 & 1 & \frac{-V_o}{\rho^2} & 0 \end{bmatrix} \tag{6.4}$$

has rank 2, equal to the order of the system. The system is observable.

6.2 Stability Analysis

Our model of curvilinear motion poses a more difficult control problem than for straight-line motion because of the presence of a constant term, V_o/ρ, in the state equation. This constant is independent of both the state and the input, and can therefore be treated as a constant disturbance. In addition, the control system is no longer a regulator. Instead, the reference value of the state ϕ is some constant angle which may be derived from the geometry of the motion. In the closed-loop system, this reference can also be treated as a constant disturbance. Together, these constant disturbances are referred to as *exogenous* inputs [Friedland 1986].

Exogenous inputs into the control system require us to modify the techniques described in Chapter 4 for designing a full-state proportional feedback controller. More specifically, the control input control must counteract the effects of the exogenous inputs so that feedback gains chosen by pole-placement techniques are blind to the presence of the disturbances. The following section outlines a technique for designing a suitable control law which can perform this task.

6.2.1 Full-State Feedback With Exogenous Inputs

The design problem is to devise a suitable control law which sufficiently counteracts the effects of the disturbances. We begin by analyzing the state equations more closely.

From the state equation for $\dot{\phi}$, equation (2.26), we see that in steady state, the input must be equal to

$$\theta_{ss} = \frac{\delta}{\rho} \tag{6.5}$$

since both d and $\dot{\phi}$ are zero in steady state. Then, from the state equation for $\dot{\phi}$, we find that in steady state, ϕ must be

$$\phi_{ss} = -\frac{\delta}{\rho} \tag{6.6}$$

This identical relation is derived geometrically as as shown in Figure 6.1.

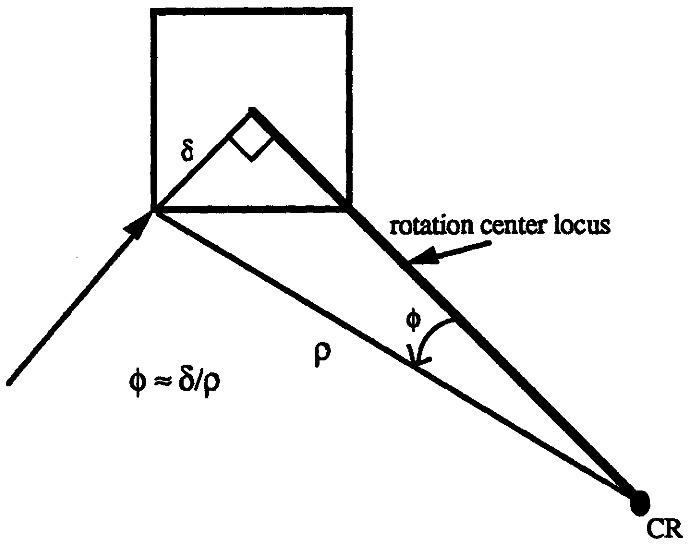

Figure 6.1 Geometric calculation of ϕ_{ss}.

We note from the above findings that our closed-loop system must generate a non-zero steady-state input. In addition, the state f must be tracked to some non-zero value which is a function of the desired radius of curvature. Figure 6.2 shows a block diagram of a closed-loop control system which fulfills these criteria.

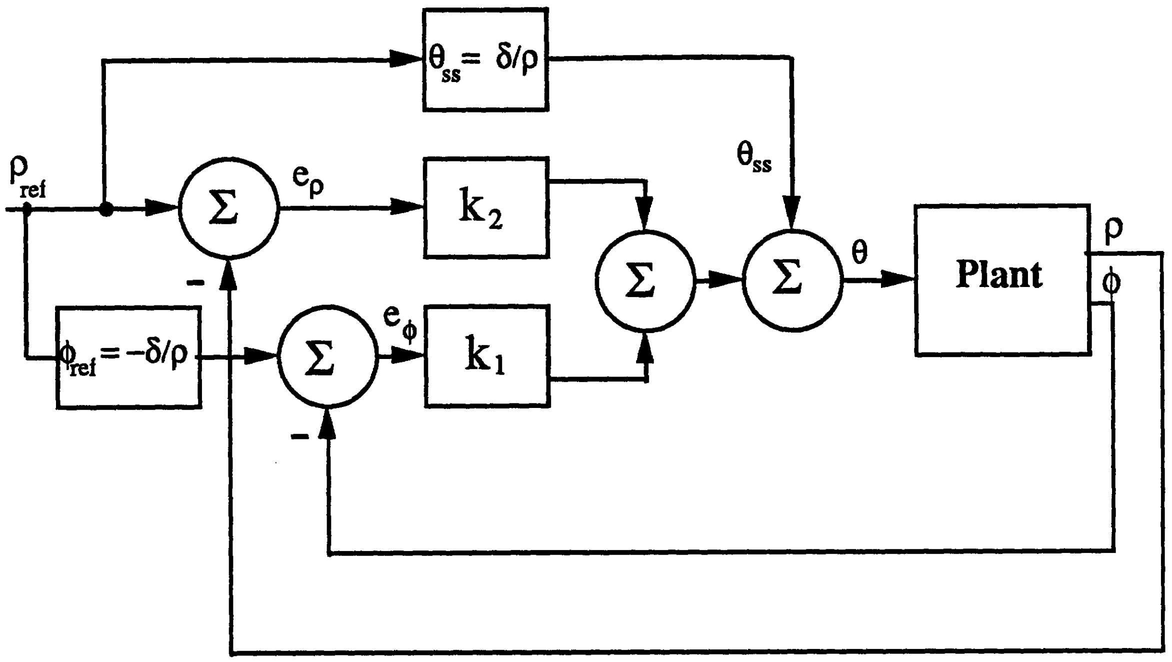

Figure 6.2 Block diagram of control system for controlling curvilinear motion.

We note immediately that the radius of curvature is the quantity we are interested in controlling; therefore, it serves as the input to the controller. From the desired curvature, we can compute the nominal value of the state ϕ as well as the steady-state input required to maintain this ϕ. The input control law according to the above diagram is

$$u = k_1 e_\phi + k_2 e_\rho + \theta_{ss} \tag{6.7}$$

Note that when the steady state errors are zero (as they should be), the plant is still driven by the non-zero input required to maintain the nominal ϕ.

In order to analyze the stability of the closed loop system, we substitute the above control law into our linearized state equations. We find that

$$\begin{bmatrix} \dot{\phi} \\ \dot{d} \end{bmatrix} = \begin{bmatrix} \frac{V_o k_1}{\delta} & \frac{-V_o}{\rho^2} + \frac{V_o k_2}{\delta} \\ V_o(1 - k_1) & -V_o k_2 \end{bmatrix} \begin{bmatrix} \phi \\ d \end{bmatrix} + \begin{bmatrix} \frac{V_o k_1}{\rho} \\ \frac{V_o \delta}{\rho}(1 - k_1) \end{bmatrix} \tag{6.8}$$

where ρ in the above equation refers to the desired value the radius of curvature and is identical to ρ_{ref} in Figure 6.2.

The matrix in the above equation which multiplies the vector of states is the closed-loop dynamics matrix. Its eigenvalues reveal the stability characteristics of the closed-loop system. Figure 6.3 shows a root locus plot of the eigenvalues of this matrix as k_1 is held constant at a value of -10.0 and k_2 is varied from -.4 to 0.0. The desired radius of curvature is specified to be 200 units.

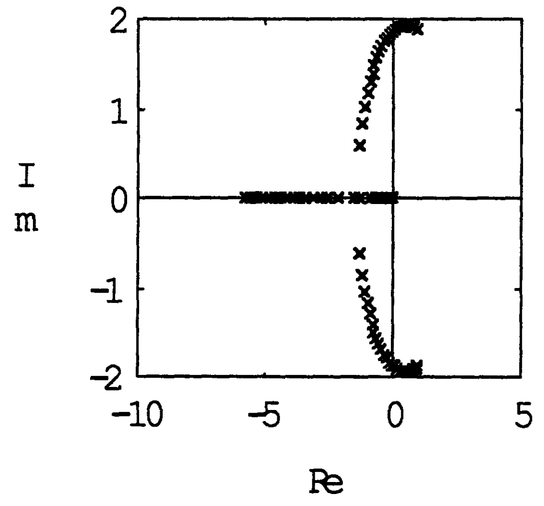

Figure 6.3 Plot of eigenvalues of closed-loop system as k_2 is varied.

When k_2 is small, both eigenvalues of the system lie on the negative real axis. This corresponds to a system with little feedback of the state d. As in our analysis of Chapter 4, this results in an overdamped system. For high values of k_2, we are effectively increasing the proportional gain of the system (while k_1, as shown in Chapter 4, can be regarded as a form of derivative feedback). The eigenvalues move away from the real axis, progressively exhibiting lower and lower damping, until both eigenvalues are in the right-half plane and the system is unstable.

The root locus plot of Figure 6.3 once again demonstrates the fact that the plant can be stabilized with just proportional feedback of the states. Interestingly, however, the closed-loop system dynamics matrix is a function of the desired radius of curvature; therefore, the eigenvalues of the closed-loop system are functions of the radius of curvature. For a control system like the one represented in Figure 6.2, this fact amounts to a system in which, for fixed gains, the dynamics are a function of the reference input.

6.4 Curvilinear Control Simulation

As in Chapter 4, we can use the state equations to simulate the closed-loop system response. The simulation is implemented identically to the simulation of straight-line motion, except that the object is now asked to track a desired curvilinear trajectory. Appendix 3 gives a listing of the code used to implement the simulation.

For the first four simulations, the desired center of rotation is the origin and the desired radius of curvature is 100 mm. The states have been given initial values of $\phi = 20°$ and $d = 40$ mm, and the magnitude of the applied velocity is 20mm/sec. The curvilinear trajectory plotted along side the actual trajectory represents the commanded trajectory. The first three simulations demonstrate typical system responses for feedback gains that correspond to different levels of damping. The fourth simulation is identical to the first except that the linearized state equations are used to perform the state integration. The final three simulations will be compared to results from experiments using the same feedback gains and initial conditions. In these simulations the states have been given initial values of $\phi = 0°$ and $d = 30$ mm. The desired center of rotation is at (100,-250) and the desired radius of curvature is 240 mm. The magnitude of the applied velocity is 10 mm/sec.

<u>Simulation 1</u> : k_1 = -10.0, k_2 = -0.1, k_I = -.001

Closed-Loop Poles: -.01, -.1697, -1.6774

Figure 6.4 shows a typical response, and Figures 6.5-6.7 show the state and input time histories for the simulation.The time scale for the simulation is 30 seconds. Note that ϕ and the input θ settle on non-zero steady-state values. Also, the effect of the integral feedback can be seen in the state d time history, as d slowly approaches its steady-state value of zero.

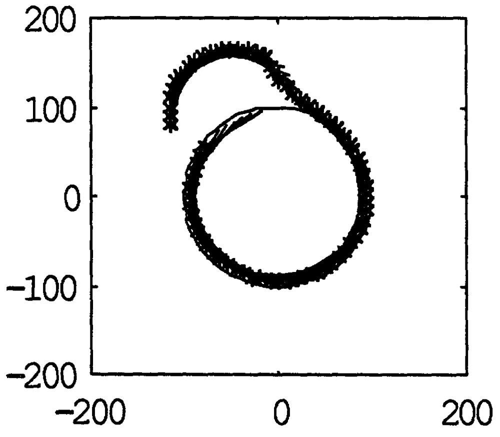

Figure 6.4. Simulation 1.

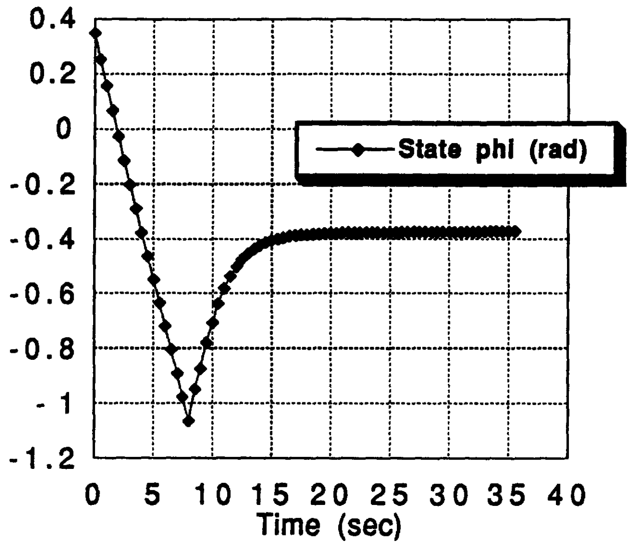

Figure 6.5. Simulation 1. Plot of state ϕ time history.

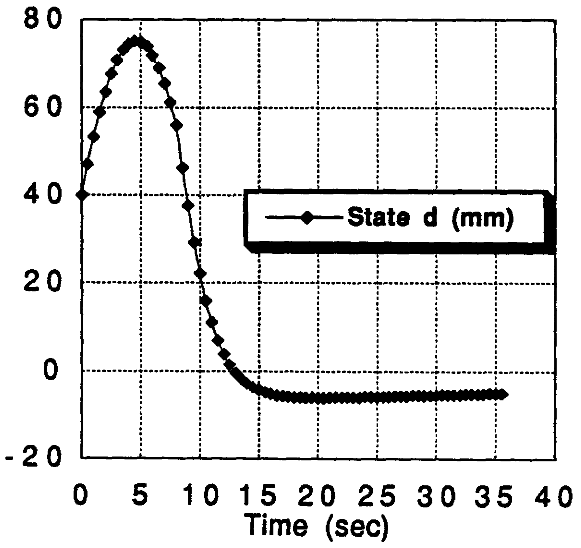

Figure 6.6. Simulation 1. Plot of state d time history.

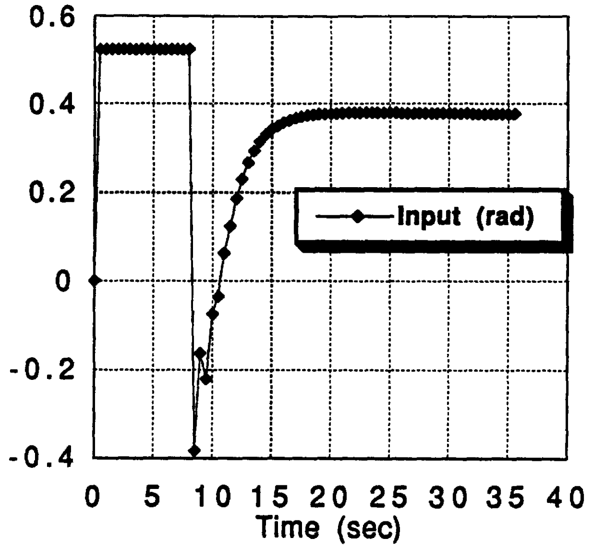

Figure 6.7. Simulation 1. Plot of input time history.

Simulation 2 : $k_1 = -5.0$, $k_2 = -0.1$, $k_I = -.001$

Closed-Loop Poles: -.01, -.2093 ± .4927i

Figure 6.8 shows an example of an underdamped response. This is confirmed by the presence of complex conjugate poles. Despite the oscillation, the system settles to steady-state in comparable time to the system in Simulation 1.

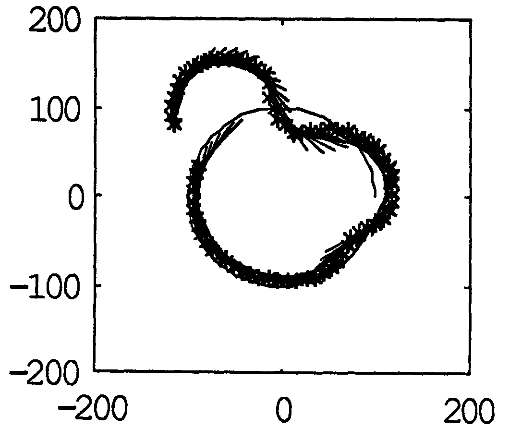

Figure 6.8. Simulation 2.

<u>Simulation 3</u> : k_1 **= -30.0,** k_2 **= -0.1,** k_I **= -.001**

Closed-Loop Poles: -.0104, -.0365, -7.5245

Figure 6.9 is an example of an overdamped response, with two poles very near the origin. Again, settling time is comparable to that of previous simulations.

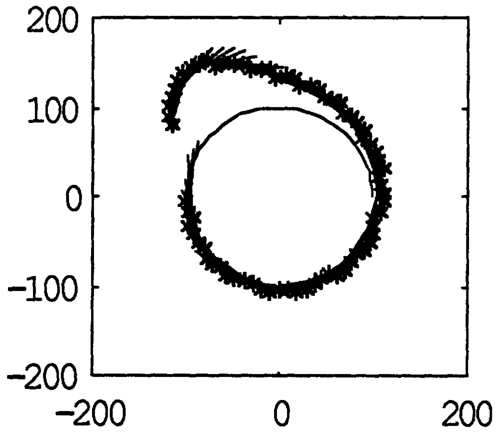

Figure 6.9. Simulation 3.

<u>Simulation 4</u> : k_1 **= -10.0,** k_2 **= -0.1,** k_I **= -.001**

Closed-Loop Poles same as those for Simulation 1.

Figure 6.10 shows the critically damped response of Simulation 1 except that the linearized state equations have been used to perform the state integration. The most apparent difference between the two simulations is that the system modelled with linearized equations is more responsive to the input. This is confirmed in the plot of the input time history, as the linearized system is in saturation for a shorter period of time than the nonlinear system. Settling time also appears smaller.

Note that the linear model exhibits oscillatory behavior. This is attributed to discretization poles resulting from the discrete time nature of the state integration. In general, though, the linearized model behaves acceptably, especially considering the large range of ϕ and θ in these simulations.

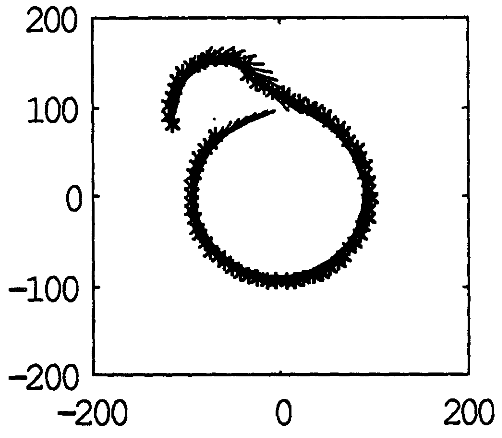

Figure 6.10. Simulation 4.

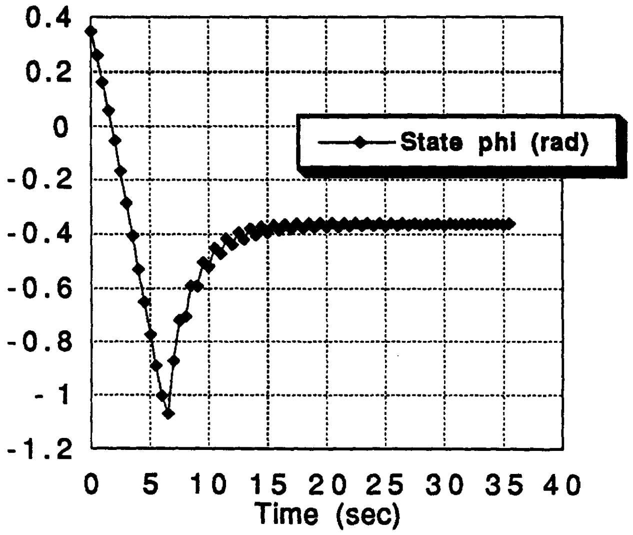

Figure 6.11. Simulation 4. Plot of state ϕ time history.

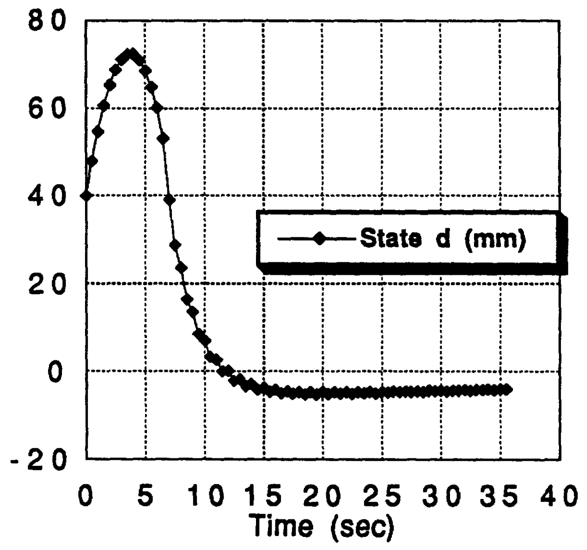

Figure 6.12 Simulation 4. Plot of state d time history.

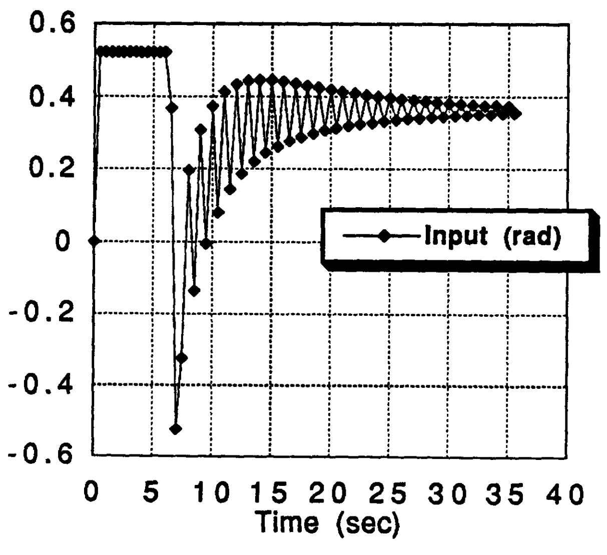

Figure 6.13 Simulation 4. Plot of input time history.

<u>**Simulation 5**</u> : $k_1 = -10.0$, $k_2 = -0.08$, $k_I = -.0007$

Closed-Loop Poles: -.0086, -.120, -1.9285

The following three simulations will be compared to experimental results. Because the specified trajectory arc is larger than for previous simulations, the system is not able to reach steady-state in the duration of the simulation. Thus, we are largely observing the transient response of the system.

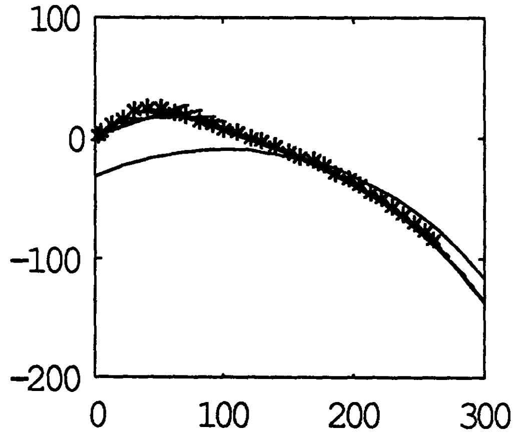

Figure 6.14 Simulation 5.

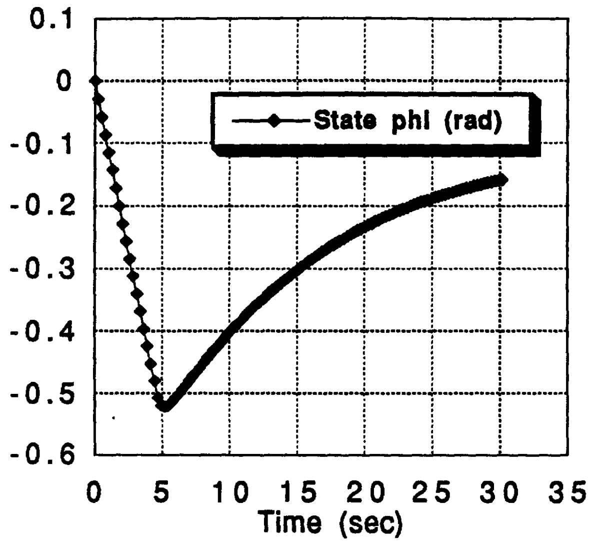

Figure 6.15 Simulation 5. Plot of state ϕ time history.

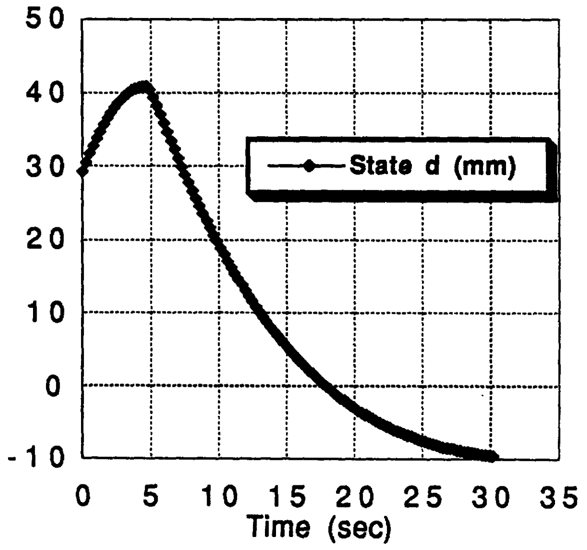

Figure 6.16 Simulation 5. Plot of state d time history.

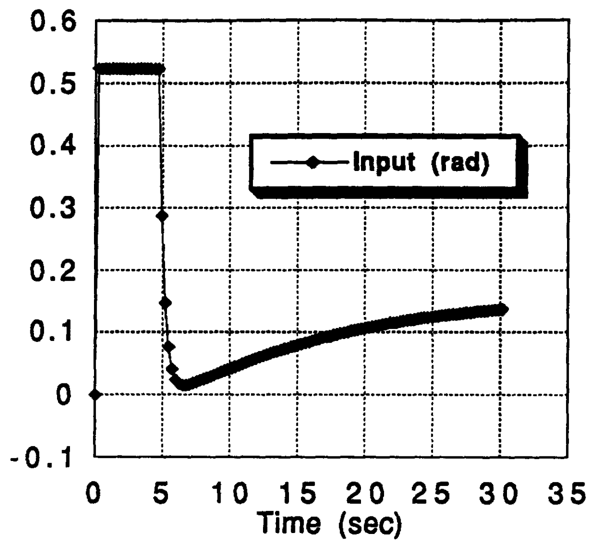

Figure 6.17 Simulation 5. Plot of input time history.

<u>Simulation 6</u> : $k_1 = -7.0$, $k_2 = -0.08$, $k_I = -.0007$

Closed-Loop Poles: -.0086, -.2442, -.9472

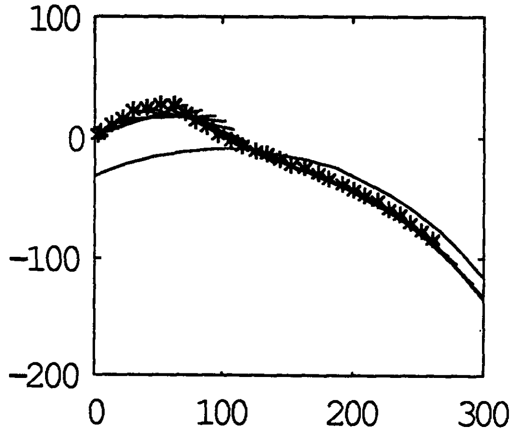

Figure 6.18 Simulation 6.

<u>Simulation 7</u> : k_1 = **-20.0,** k_2 = **-0.08,** k_I = **-.0007**

Closed-Loop Poles: -.0086, -.047, -4.858

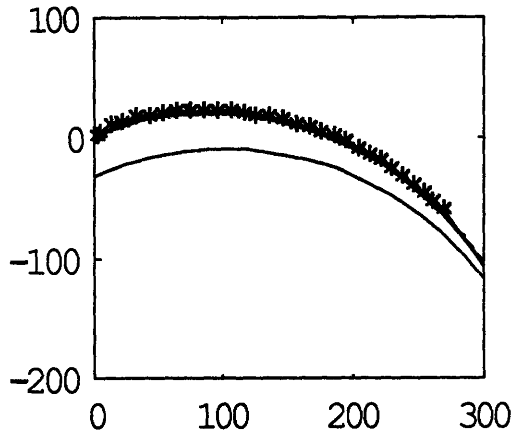

Figure 6.19 Simulation 7.

6.5 Experiments

The following series of plots show the results of the hardware simulation for the same set of controller gains as were tested in the final three software simulations. The initial state values and other system constants are the same as previous simulations. In addition, input saturation at a pushing angle of 30° in either direction is implemented so as to make sure that the flange sensor which reads the current state ϕ will always make appropriate contact with the sliding object.

<u>Experiment 1</u> : k_1 = **-10.0,** k_2 = **-0.1,** k_I = **-.001**

The results from the following three experiments can be compared to the results from Simulations 5-7. As with straight-line motion, the experimental results exhibit considerably less damping than is predicted by simulation. Some sources of discrepancy are discussed in Chapter 4.

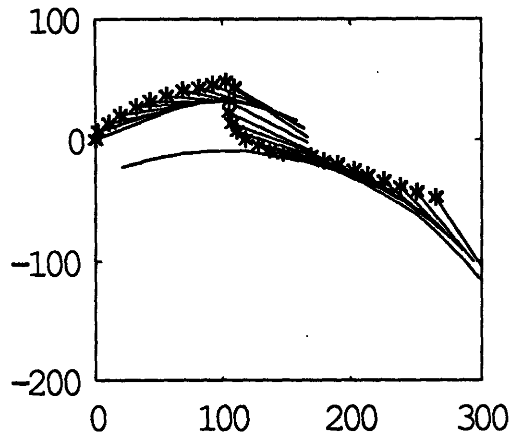

Figure 6.20 Experiment 1.

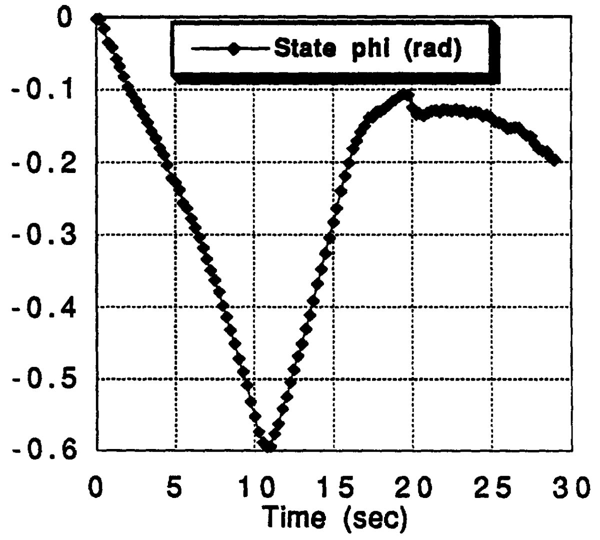

Figure 6.21 Experiment 1. Plot of state ϕ time history.

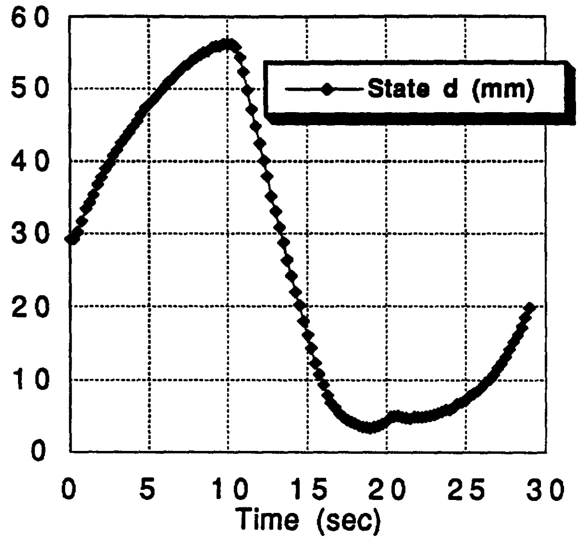

Figure 6.22 Experiment 1. Plot of state *d* time history.

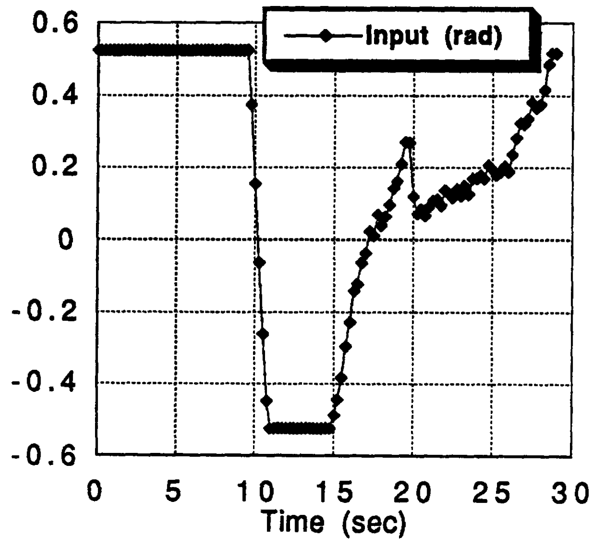

Figure 6.23 Experiment 1. Plot of input time history.

<u>**Experiment 2**</u> : k_1 = **-7.0**, k_2 = **-0.08**, k_i = **-.0007**

Underdamped Response

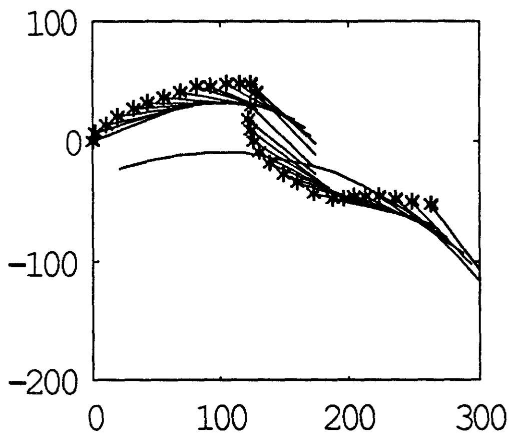

Figure 6.24 Experiment 2.

<u>Experiment 3</u> : k_1 = **-20.0,** k_2 = **-0.08,** k_i = **-.0007**

Overdamped Response

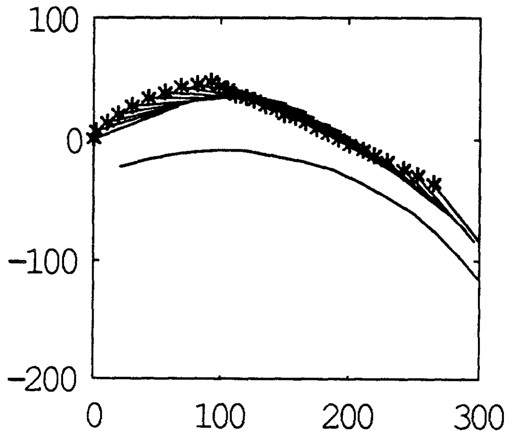

Figure 6.25 Experiment 3.

6.6 Discussion

An experimental error which is likely to account for the discrepancy between the experimental and simulated results is that the pusher which senses the object orientation is offset from the shaft of the encoder which reads this orientation. In other words, the corner which is pushed is not in line with the shaft of the encoder. The sensed orientation is therefore larger than the actual orientation, resulting in larger errors and larger inputs. The offset is less than a centimeter; however, this is significant in comparison to the length of a side of the square object being pushed.

Chapter 7
Conclusions

7.1 Controlling Basic Motions

We have demonstrated in this thesis the practicality of controlling the position and orientation of sliding objects a robotic manipulator. We began with an analysis of the planar motion of sliding objects, and proceeded to develop mathematical models of the object motion for specific cases such as tracking both straight-line and curvilinear trajectories. From these models, we were able to design closed-loop control systems for controlling the motions and, in addition, implemented the control systems with experimental hardware in order to demonstrate their practical effectiveness. From the results of the experiments, we are able to draw certain conclusions about the validity of the plant models and about the effectiveness of the controller designs.

7.1.1 Model Robustness

One of the main purposes of the modelling is to identify system states that can accurately represent the object motion. Both the software and experimental simulations show that by controlling the states derived from the modelling, we are able to achieve stable motion of the object. In addition, the model, although derived from many assumptions of linearity, is useful even for large initial conditions. This is seen by comparing simulations using both the linearized and nonlinear models. The experimental results for the straight-line motion also indicate that the motion continues to be stable in the presence of errors which we can not model, such as the non-uniformity of the pressure distribution.

7.1.2 Sources of Error

The primary source of deviation of the experimental results from the simulated results is the non-uniformity of the pressure distribution. Any irregularities in the pressure distribution can result in a drastically different locus of instantaneous centers of rotation, thereby negating the straight-line assumption made in Chapter 2 regarding this locus. Irregular pressure distributions also result in a center of friction which is no longer coincident with the center of gravity. Therefore, pushing angles which are meant to rotate the object in a desired direction can instead rotate the object opposite this direction. Much of the symmetry in the geometrical model of the object's motion is lost when the center of friction changes.

In addition, the offset of the pushing point from the shaft of the encoder which senses object orientation is likely to explain the unexpected overshoot seen in the experimental results. In fact, this error is probably more significant than error due to the nonuniformity of the pressure distribution.

7.2 Extension to General Reorientation Maneuvers

Because sliding-object motion can be expressed as a rotation about an instantaneous center of rotation, it is a straight-forward procedure to apply the control system design and analysis presented in this thesis to more general reorientation maneuvers consisting of multiple rotations about a fixed center of rotation. The major problem which arises is that there exists a finite time before the object is able to achieve the desired curvilinear trajectory. In addition, there is usually some non-zero steady-state error. The challenge, therefore, lies in controlling motions consisting of multiple rotations.

Let us consider as an example the two-step reorientation discussed in Chapter 5 which involves two tangent arcs. If the first motion does not *exactly* follow the desired arc trajectory, then we must rely on control to track the second arc. If the deviation from the first arc is too large in comparison to the size of the second arc, it may not be possible to correct the error. It is likely, then, that the procedure of defining the desired trajectory should be adaptive. In other words, the radius of the second arc is not exactly determined until the first rotation is complete.

7.3 Suggestions For Further Research

Having analyzed in this thesis some of the fundamental concepts relating to the control of sliding-object motion, it is worth discussing now some of the specific areas of the research which could benefit from further consideration.

7.3.1 Modelling

The models developed in Chapter 2 are specific to the case in which the pressure distribution along the sliding surface is uniform. While this assumption represents a large class of sliding-object motion, it is important to understand the effects of errors in this approximation. As mentioned earlier, small changes in the pressure distribution can result in significant changes in rotation center loci, rendering the geometric models of Chapter 2 inadequate. It is necessary, therefore, to extend these models to account for arbitrary pressure distributions.

7.3.2 Control

The control system designs of Chapters 4 and 6 can be extended to the control of more complicated motions involving multiple rotations. This problem involves another level of control more closely associated with adaptive trajectory planning. It will no longer be sufficient just to specify a particular curvilinear trajectory to follow. The distance to be travelled along the trajectory must also be specified.

7.3.3 Experiments

The experiments could benefit greatly from two modifications to the experimental set-up. First, the uniform pressure distribution along the contact region can be better simulated by wrapping the sliding object rather than the table with felt. In this way, any moments generated by pushing above the plane of motion will not tend to drive the sliding object into the felt. The second modification is to account for the offset of the pusher.

7.3.4 In-Grasp Reorientation

One of the applications of sliding-object motion control is the in-grasp reorientation of objects. In particular, it is worth noting the specific case of a parallel-jaw gripper, since in-grasp reorientation with these end-effectors more closely resembles the analogous problem of pushing a sliding object along a table. For this problem, we require local sensors such as tactile array sensors. Appendix 2 details the use of these sensors in this application. While these sensors can provide detailed information about the pressure distribution acting on the object, they also inherently require more time to scan, resulting in a lower bandwidth controller. Nonetheless, they are the only practical sensors for this application, even if only used to provide local position and reorientation and not force. It would be interesting to study the effects of the large scan time on control performance. In addition, there is a certain amount of uncertainty associated with extracting position and orientation information from a sensor which yields a discrete image of the sliding object. It is important to understand the effects of position sensing error on the controller performance.

The in-grasp reorientation problem also presents with it an entirely different planning issue from the traditional problem of objects sliding on a table. The range of motion of the grasped object is significantly more constrained, as is the range of pushing angles.

Appendix 1
Rotation Center Loci

In this section, we present the results of several calculations of the locus of instantaneous centers of rotation for differently shaped objects. The loci have been calculated by solving the fixed-contact quasi-static equation of Chapter 2 (equation 2.10) for a range of pushing angles and with the assumption that the pressure distribution across the region of contact is constant and uniform.

For convenience, the results have been divided by class of geometric cross-section. It is worthwhile noting the similarity in the rotation center locus of markedly different object cross-sections. In general, one finds that there exists symmetry about the line connecting the contact point and the center of mass of the object. In addition, the locus very nearly approximates a straight line as the applied velocity vector passes through the center of mass. The implications of these facts are that the motion and orientation of an object can be approximated with just knowledge of the contact point and the location of the center of mass. The information is independent of the exact cross-section. These properties are exploited in Chapter 2 throughout the derivation of the mathematical and geometric models of sliding-object motion. The straight-line motion and curvilinear motion models in these chapters can be applied to many differently shaped objects.

The following sections represent the results of the calculations. At the beginning of each section is a table itemizing the different object cross sections and the respective range of pushing angles tested for the cross sections. The figures above the tables show the location of the contact point and the relation between the pushing angle and the object cross section. Note that in the following definitions of pushing angle, we have departed from the convention defined in Chapter 2. This new definition of pushing angle is fully appropriate in the following results since it is referenced to a coordinate system common to all the cross sections. With this new definition, we can avoid explicitly identifying the line connecting the contact point and the center of mass of the object, as this line is different for each cross section. For control purposes, the convention of Chapter 2 is to be adopted.

All of the loci have been plotted in polar coordinates; therefore, the instantaneous center of rotation corresponding to a particular pushing angle can be found easily by searching radially along a direction perpendicular to the applied velocity vector.

A1.1 Rectangles

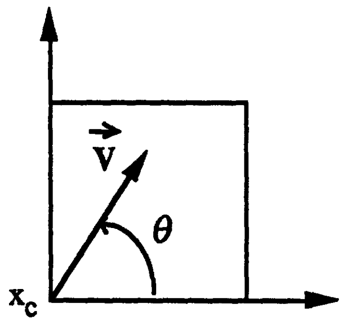

Figure A1.1 Rectangles: definition of terms.

	Dimensions	$d\theta$ (degrees)	Range of θ (degrees)
#1	1 X 1	.5	50, 180
#2	2 X 1	.5	40, 180
#3	1 X 2	.5	70, 180
#4	8 X 1	.5	20, 180
#5	1 X 8	.5	90, 180

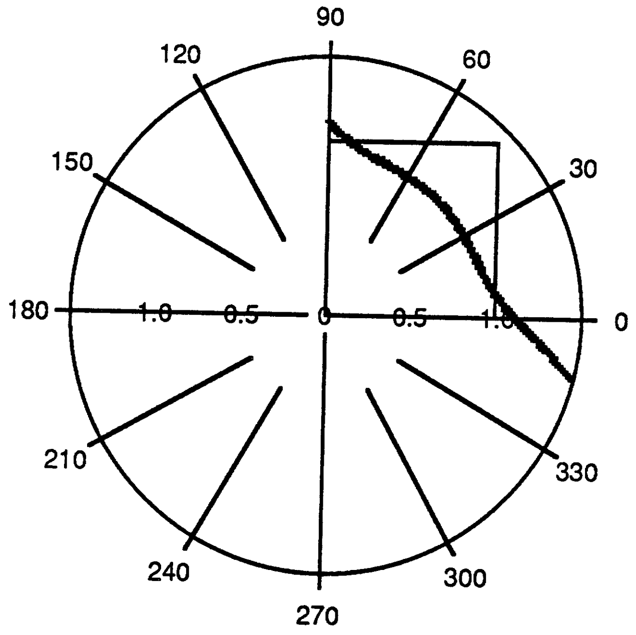

Figure A1.2 Rectangle 1.

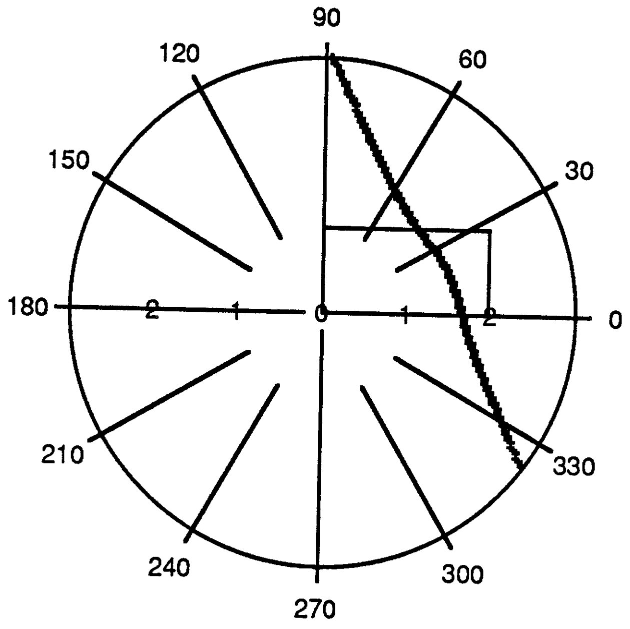

Figure A1.3 Rectangle 2.

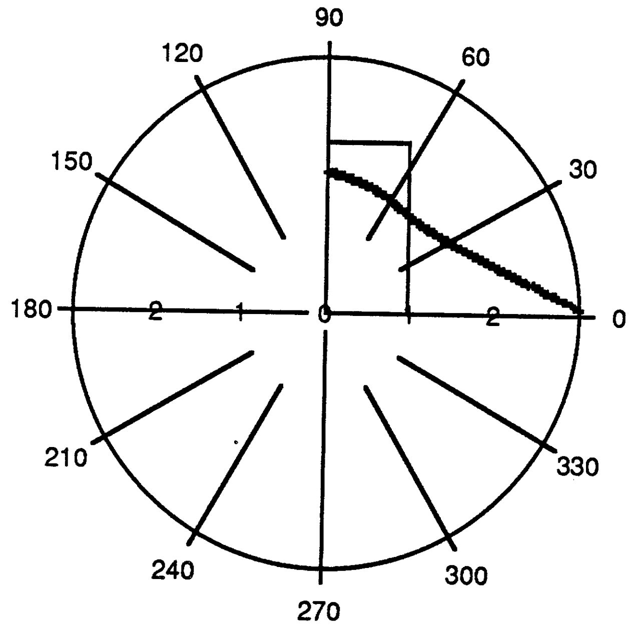

Figure A1.4 Rectangle 3.

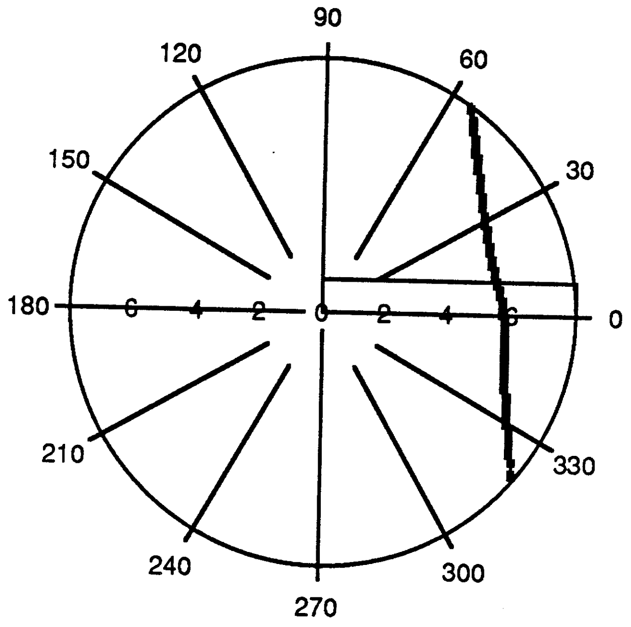

Figure A1.5 Rectangle 4.

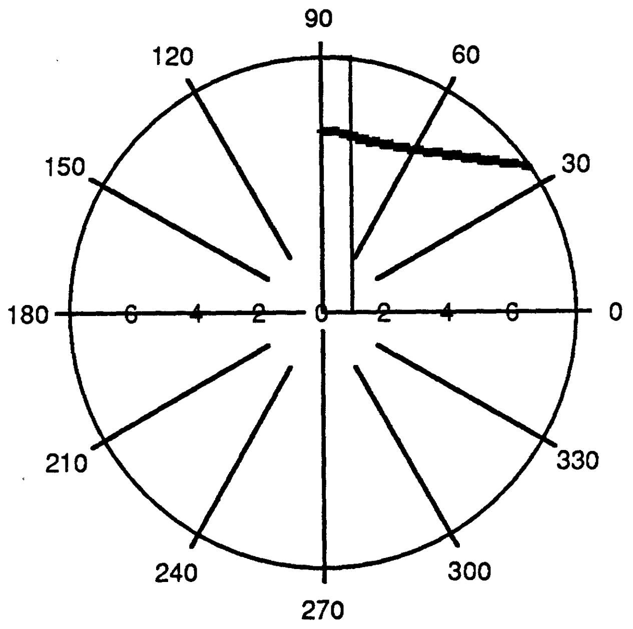

Figure A1.6 Rectangle 5.

A1.2 Triangles

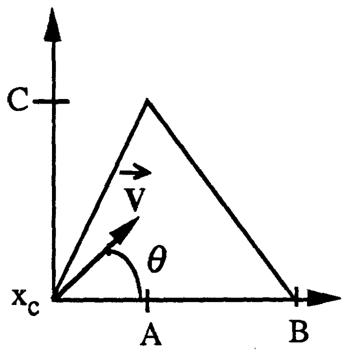

Figure A1.7 Triangles: definition of terms.

	A	B	C	dθ (degrees)	Range of θ (degrees)
#1	0.5	1.0	0.866025	.5	180, 40
#2	0.5	1.0	2.0	.5	180, 70
#3	1.0	2.0	0.5	.5	180, 20
#4	0.5	2.0	1.0	.5	180, 30
#5	0.0	2.0	1.0	.5	180, 35
#6	0.0	1.0	1.0	.5	180, 60

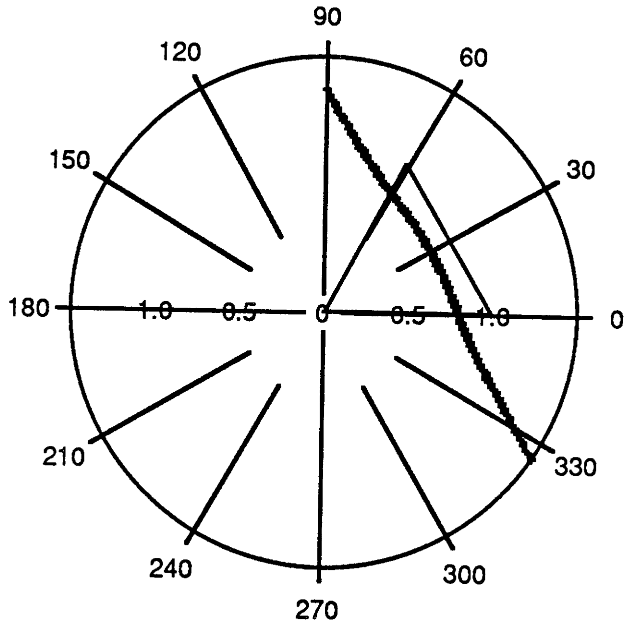

Figure A1.8 Triangle 1.

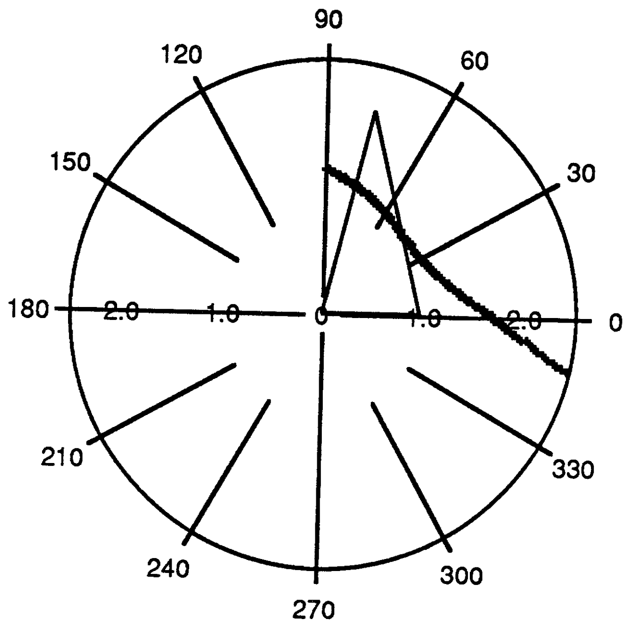

Figure A1.9 Triangle 2.

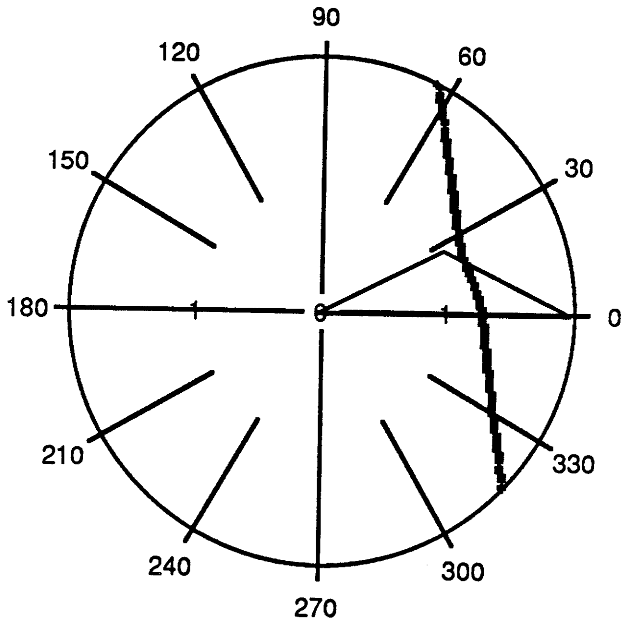

Figure A1.10 Triangle 3.

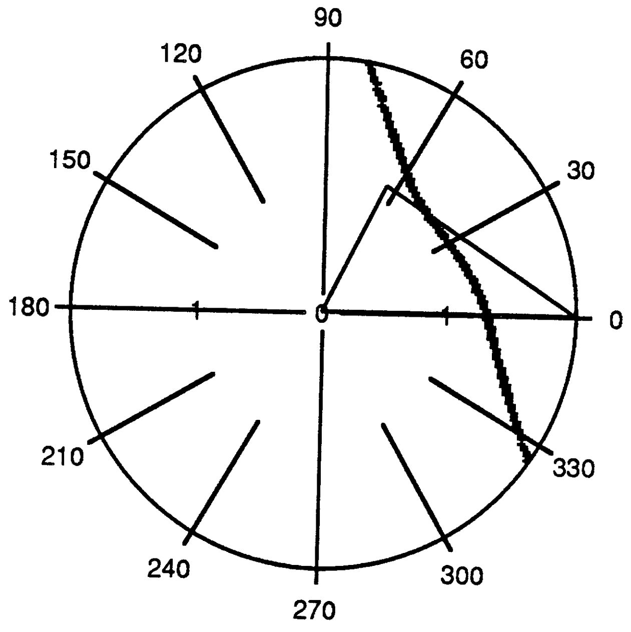

Figure A1.11 Triangle 4.

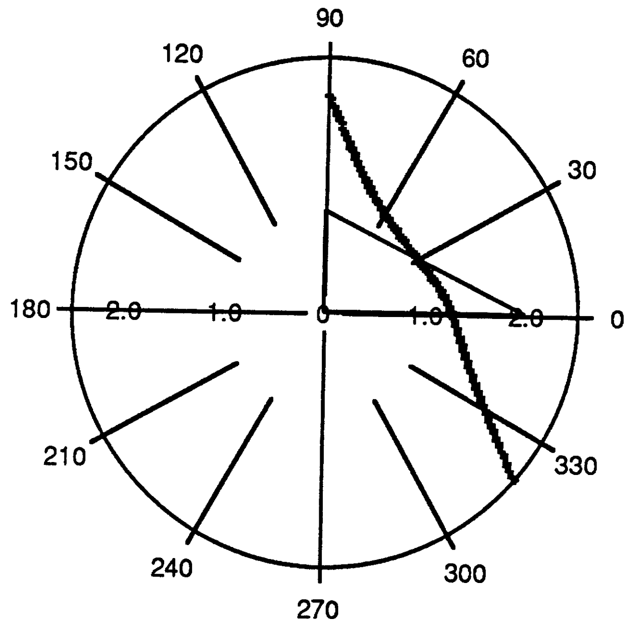

Figure A1.12 Triangle 5.

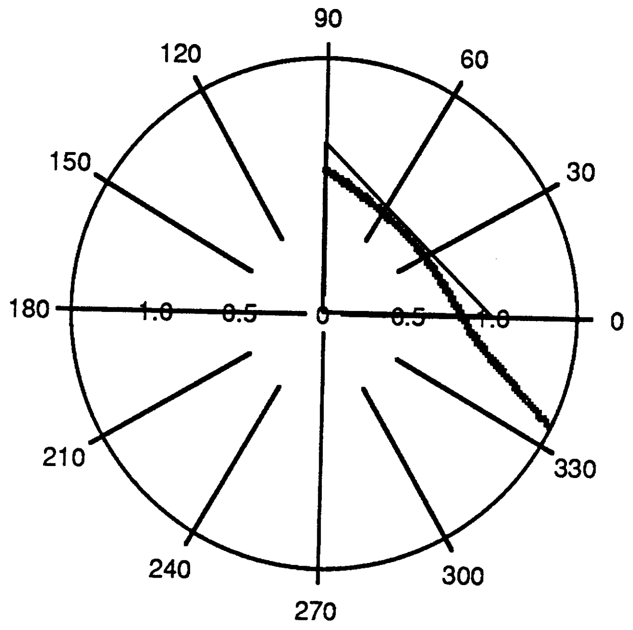

Figure A1.13 Triangle 6.

A1.3 Ellipses

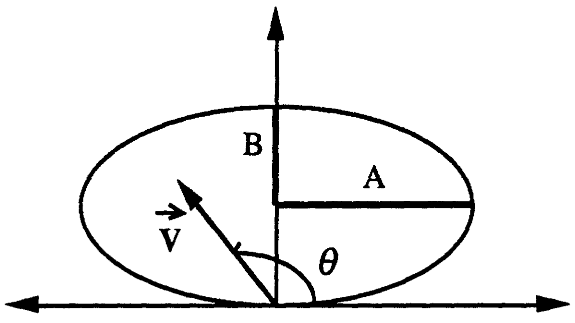

Figure A1.14 Ellipses: definition of terms.

	Center (x, y)	A	B	dθ (degrees)	Range of θ (degrees)
#1	(0.0, 0.5)	0.5	0.5	.5	180, 100
#2	(0.0, 0.5)	1.0	0.5	.5	180, 100
#3	(0.0, 1.0)	0.5	1.0	.5	180, 100
#4	(0.0, 2.0)	0.5	2.0	.5	180, 100
#5	(0.0, 0.5)	2.0	0.5	.5	180, 100

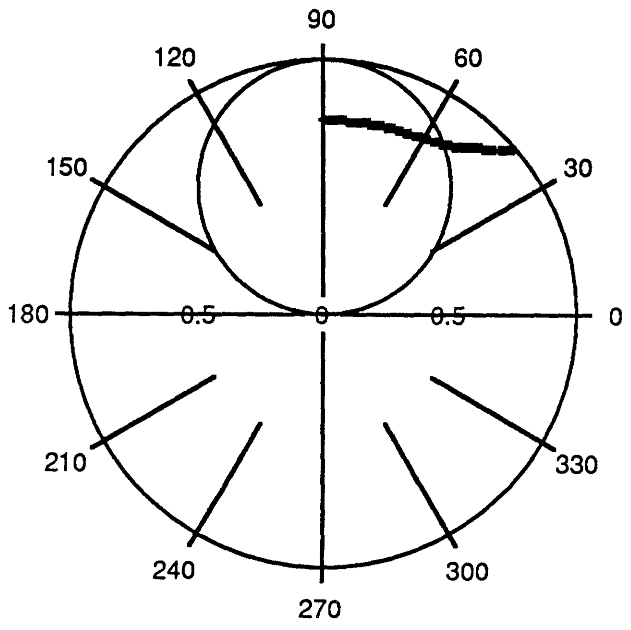

Figure A1.15 Ellipse 1.

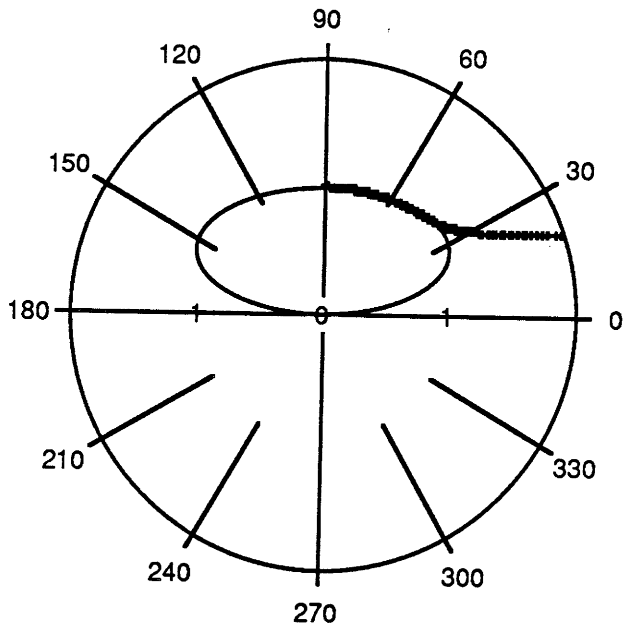

Figure A1.16 Ellipse 2.

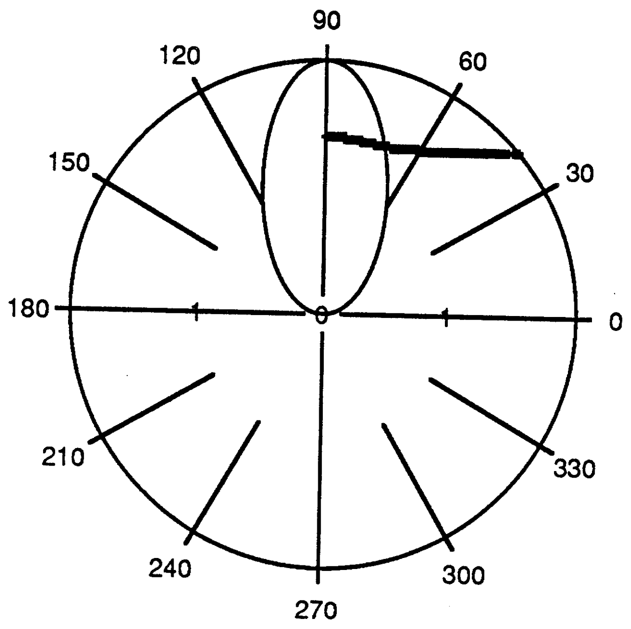

Figure A1.17 Ellipse 3.

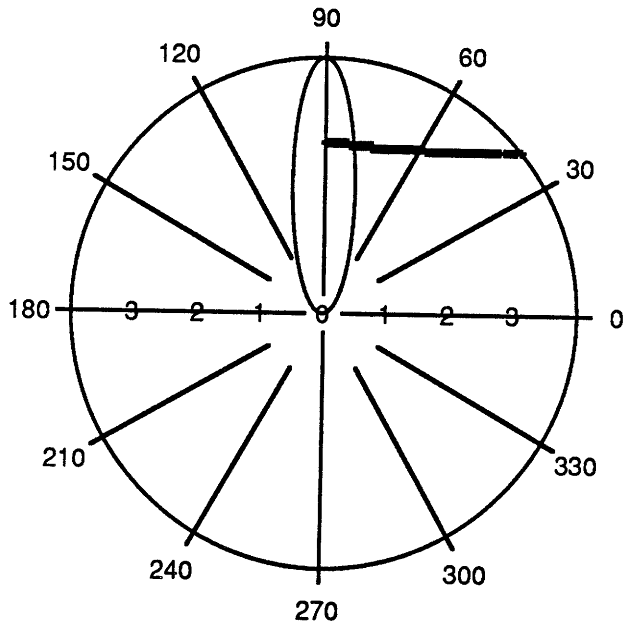

Figure A1.18 Ellipse 4.

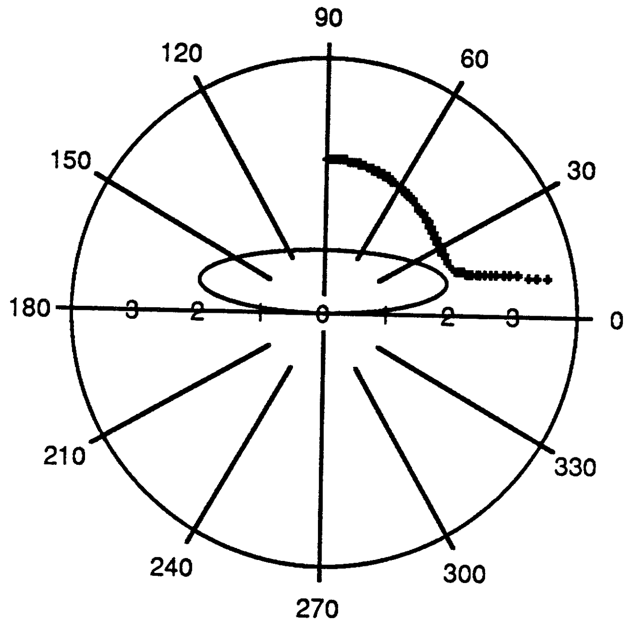

Figure A1.19 Ellipse 5.

Appendix 2
Tactile Sensing

In order to control manipulator operations involving the in-grasp reorientation of objects, it is necessary to have some form of local sensing for determining the grasped object's position and orientation with respect to the gripper. Tactile array sensors can provide this position sensing capability for a variety of different end-effectors, including multi-fingered hands as well as parallel-jaw grippers. In addition, the tactile feedback can provide force sensing to be used to control the grip forces acting on a grasped object [Fearing 1987].

Through the course of this research, significant consideration has been given to the use of tactile array sensors to be used with a parallel-jaw gripper in order to effect in-grasp reorientations. Although the ultimate focus of the thesis has been the analysis of sliding-object motion and reorientation with the general application of pushing objects on a table, it is worthwhile in this appendix to discuss the use of these sensors in the application of in-grasp reorientations and to briefly outline the results of the various studies of tactile sensors which were performed as a part of this research.

A2.1 Sensor Specifications

The tactile array sensor used in the analysis is one developed by TEKSCAN in Boston, Massachusetts.The sensor consists of a conductive gridwork of rows and columns etched on a flexible substrate [Benjamin 1985]. Between the rows and columns is a layer of ink in which are embedded conductive particles. As the ink is compressed, the conductive particles are brought closer to each other, and the local resistance through the ink is reduced. The sensor, thus, gives grey-scale tactile information—the contact pressure is deduced from changes in the local resistance, and the location of the applied force is determined from the location of the row-column intersection where the change in local resistance is observed.

The entire sensor is encased in a flexible mylar film, allowing for a desired conformity with the surface on which it is mounted. The particular array sensor used for this research consists of a network of 44 rows and 44 columns, with a spatial resolution in each dimension of 50 mil.

A2.2 Sensor Characterization

In order to perform dynamic object recognition with the tactile array sensor, it is useful to first characterize both the static and dynamic behavior of the sensor. This is accomplished by observing the sensor response to various inputs. In particular, we are concerned with the sensor response as a function of the applied load, the applied load time, and the relaxation period between successive applied loads.

In the first case, we are able to determine the Force/Resistance curve of the sensor from which we can make observations regarding the linearity of the sensor response across a range of applied loads. This information also leads to the formulation of the best operating regions for a desired output sensitivity. It is found that the sensor response to applied loads is highly non-linear. Figure A2.1 shows a typical Force/Resistance curve of the sensor. It is observed that the resistance falls off dramatically for small loads, and saturates for very large loads.

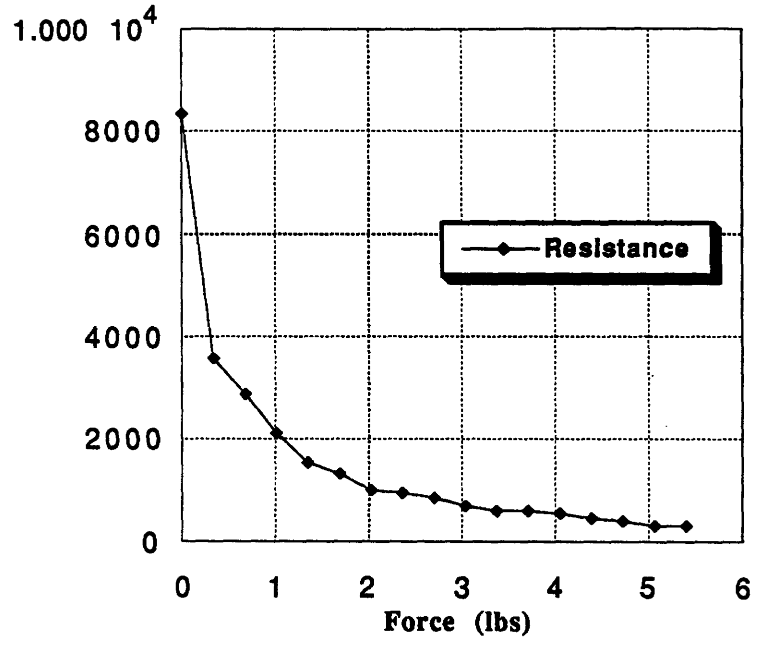

Figure A2.1 Typical Force/Resistance curve of tactile sensor.

Tests of the sensor's response as a function of the duration of the applied load show that the sensor exhibits significant "creep" in the output. In other words, the measured resistance continues to fall even after the load has been applied. In a particular test in which a constant pressure of 10 psi was applied on the sensor for a period of 15 minutes, creep in excess of 700% was observed. This behavior is obviously undesirable, as it poses a difficulty in achieving a direct relation between the applied load and the measured output. Perhaps, the dynamics of the creep can be modelled; however, this is a significant problem in and of itself.

A final test of the sensor output in response to variations in the relaxation time between applied loads indicate not only that the sensor is susceptible to permanent deformation due to moderate loads applied over a long period of time, but that the recovery of the sensor output is also a function of the relaxation period. Again, this behavior implies that there exist non-transient dynamics which must be understood in order to properly use the sensor for accurate force information.

For the purpose of simplifying the sensing issue associated with the in-grasp reorientation of objects and in order to avoid sacrificing the primary objectives of the work by attempting a full-scale characterization of the sensor behavior, it was decided that the sensor not be used for providing grey-scale force information. Instead, the sensor output is thresholded in hardware, thereby rendering the force information binary. This decision carries with it several important ramifications. First, because the sensor will now only distinguish between 'on' and 'off' contacts, it must be assumed that the pressure distribution over the sensor is uniform and constant. Only with this assumption can the center of friction be determined. In addition, force thresholding in hardware allows the tactile array sensor to be scanned with a purely digital interface. This will result in significantly higher scan rates, as the sensor output does not need to be sent through an A/D converter. Finally, data processing algorithms for determining the object's position and orientation from the sensor data will be greatly simplified.

A2.3 Scanning Electronics

Figure A2.2 shows a schematic diagram of the electronics required to scan the tactile array. The scanning is performed sequentially by using decoders to select a particular row (connected to +5 V), and then using analog multiplexers to observe the output at each column. The op-amp at the output acts like a high impedance ground, thus it plays a significant role in both isolating the selected pixel and removing signal cross-talk between adjacent pixels. The electronics are driven by a PC through a parallel I/O interface.

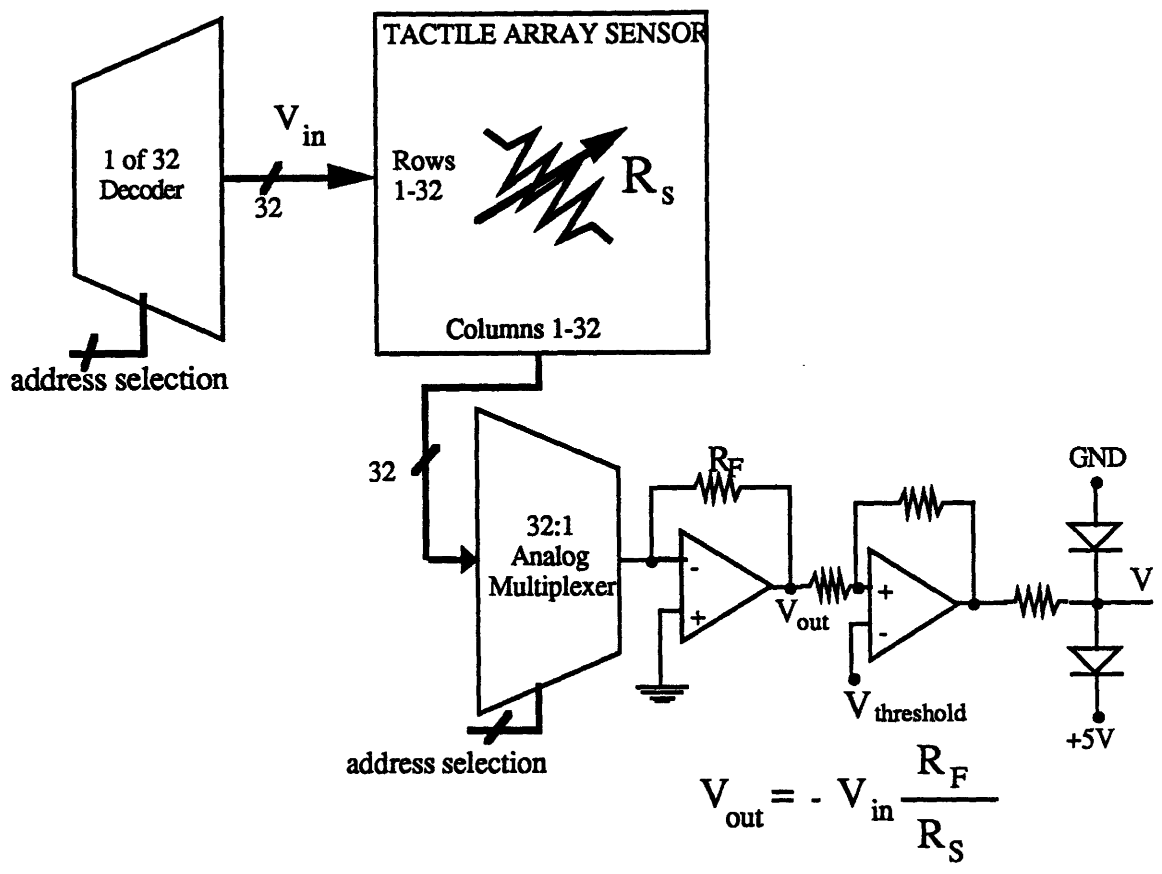

Figure A2.2 Schematic diagram of scanning electronics.

After implementing the above scanning electronics, it was found that a 32 X 32 tactile array can be scanned in 25 Hz. This scanning frequency represents a significant improvement over that achieved without hardware thresholding of the sensor output. Depending exactly on the particular I/O interface used, the improvement can be as high as two orders of magnitude.

A2.4 Tactile Data Processing

The sensor output must be manipulated to provide information about the image position and orientation. Standard image processing tools can be used for this purpose. The following are two methods for determining the imaged object's position and orientation.

Driels [Driels 1986] describes a method for determining object orientation using moment of inertias. The technique is based on the comparison of the moments and product of inertia of the image with respect to both a predefined object axes (aligned with the sensor axes) and the object's principal axes. Noting that the product of inertia with respect to the principal axes is zero, the angle, θ_p, between the principal axes and the sensor axes is given by

$$\tan(2\theta_p) = \frac{-2I_{xy}}{I_x - I_y}$$

where the moments and product of inertia are defined with respect to the sensor axes and are given by

$$I_x = \sum (x - x_{cg})^2 A$$

$$I_y = \sum (y - y_{cg})^2 A$$

$$I_{xy} = \sum (x - x_{cg})(y - y_{cg})\ A$$

In the above equations, x_{cg} and y_{cg} are the coordinates of the area centroid of the image and A is the total area of the image in pixels. For each pose of the object, we calculate the angle difference between the sensor axes and the principal axes. The change in orientation between successive poses is given by a comparison of the angle difference for the two poses.

One disadvantage of the inertia technique is that it is very sensitive to noise in the image data. For example, uncertainty near the edges of the imaged object is magnified since moments of inertia are functions of the distance from the area centroid. An alternative approach to determining the object's orientation is based on finding the edges of the imaged object, and then determining the orientation from the slopes of the lines representing the edges.

By applying a center difference mask on the image data, we can isolate the edges. The mask is simply a second-order finite difference approximation of the local derivative. Once the edges are found, we can perform a Hough transform to find the slopes of the edge lines.

The Hough transform is a useful technique for identifying features with like parameterization. For instance, all the image points on a line have the same slope and intercept. One can imagine, then, that in some parameter space consisting of slope and intercept, all of these points on the same line map to a common point which is the slope and intercept of the line. The transformation is more easily seen by analyzing the equation of a line in slope-intercept form.

$$y = \mathrm{m}x + \mathrm{b}$$

We can just as well write this equation in terms of the slope and intercept

$$b = -\mathrm{x}m + \mathrm{y}$$

Thus a single point in image space (the x-y plane) corresponds to a line in Hough or parameter space (the m-b plane) and vice versa. Note that for the special case of lines, the transformation is completely reflective.

Applying the method of Hough transform to our original problem, we see that after performing edge detection on the image, we can transform the result into Hough space to extract the slope of the lines representing the edges. For a rectangular object, the transformation will result in four points in parameter space corresponding to the slope-intercept descriptions of the four sides of the rectangle. The orientation of the object can be easily determined from the slopes of these lines. Note that because the transform weighs each point in image space evenly, inherent noise in the image data is not magnified.

Appendix 3
Experimental Hardware and Control Specifications

In this appendix, we present in greater detail the design and implementation of the experimental hardware. Of particular interest are the specifications of the motion control board used to provide low-level motor control and the software drivers associated with operating the board. The appendix also provides a detailed functional diagram of the high-level software used to fuse the multiple layers of control.

A3.1 5638 Motion Control Board

The 5638 is a full-function PC-bus compatible three-axis motion control board. At the heart of the motion control is a National Semiconductor LM628 motion control IC. This chip implements a digital programmable PID filter to generate low-level position and velocity servo control. The chip accepts encoder quadrature signals as input, and decodes these signals internally to determine position displacement. The output is a DAC signal which can be connected to a power amplifier to drive the motor. The chip also generates a trapezoidal trajectory profile given a desired acceleration and velocity.

The 5638 board, in addition to all the features of the LM628, provides an interface to the PC's interrupt lines so that interrupts can be triggered by the LM628's status bits. In addition, the board provides an auxiliary encoder counter, which is conveniently used to measure the displacement of the flange sensor described in Chapter 3.

A3.2 Software Drivers

The experimental operation, as diagrammed in Figure A3.1, is comprised of three distinct levels of control: low-level PID control of joint actuators, cartesian velocity control of joints, and high-level control of sliding-object motion. The latter two are implemented entirely in software, while the first is implemented in hardware by the 5638 board. Low-level software drivers, however, are required to program and initiate the 5638 functions. The software used to run the experiments has, therefore, a multi-layered architecture.

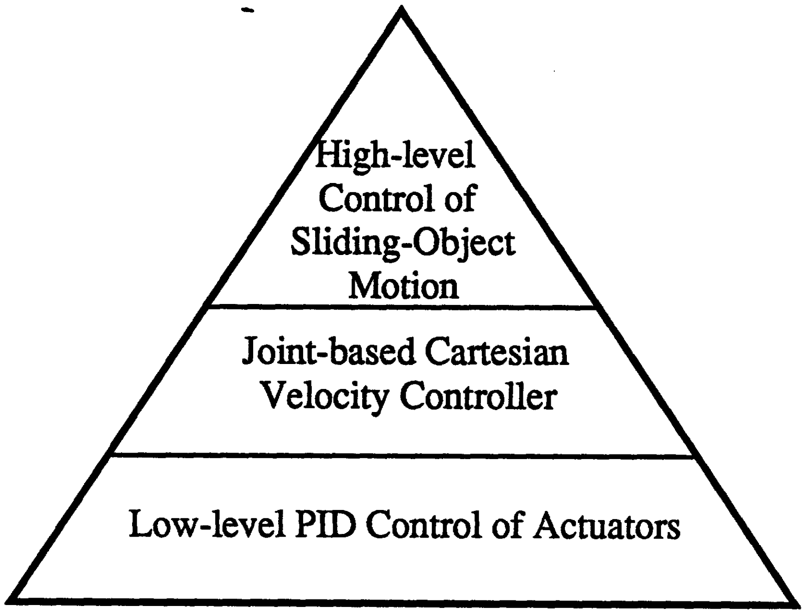
High-level Control of Sliding-Object Motion
Joint-based Cartesian Velocity Controller
Low-level PID Control of Actuators

Figure A3.1 Multi-layered control architecture.

High-level Control of Sliding-Object Motion

Figure A3.2 outlines the functions of the high-level motion control software.

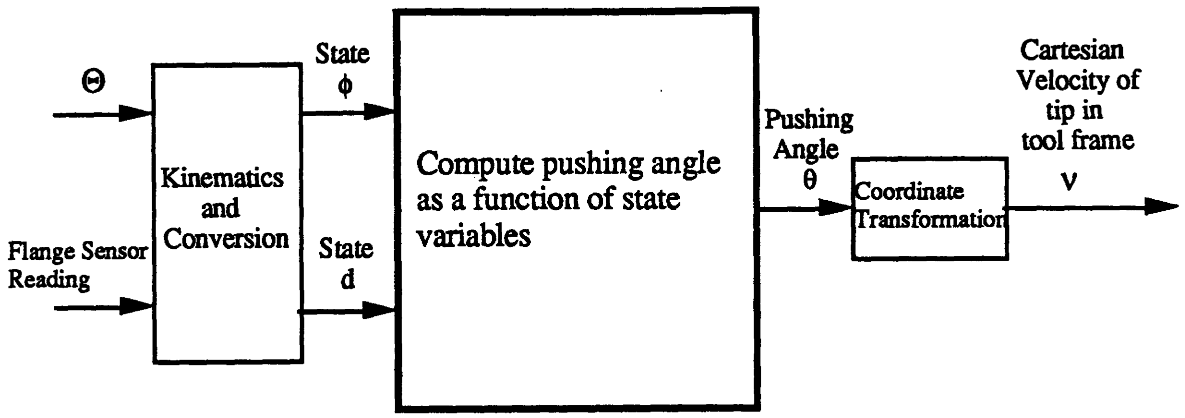
Θ
Flange Sensor Reading
Kinematics and Conversion
State ϕ
State d
Compute pushing angle as a function of state variables
Pushing Angle θ
Coordinate Transformation
Cartesian Velocity of tip in tool frame
V

Figure A3.2 Functional diagram of high-level sliding-object motion control software.

Cartesian Velocity Controller

Figure A3.3 diagrams the functions of the cartesian velocity controller.

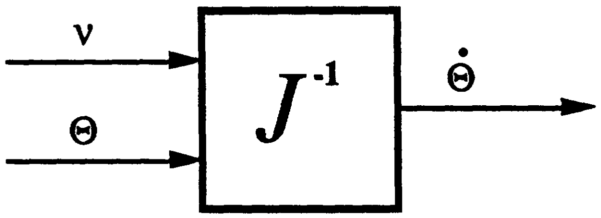

Figure A3.3 Functional diagram of joint-based cartesian velocity controller.

Low-level Control of Joint Actuators

The software which drives the 5638 board is divided into 7 files:

ins.c: Contains all low-level instructions inherent to the LM628. These subroutines write directly to the command port of the 628's.

mac.c: Contains higher-level macro definitions which call several low-level functions in order to perform a specific task, such as displaying status information or prompting the user for programmable inputs.

int.c: Contains all functions used to set-up interrupt handlers.

utl.c: Contains all low-level I/O routines for reading and writing to the command and data ports.

init.c: Contains several high-level routines for initializing the board.

inc.h: Include file for global variables and compile-time constants. Also in this file is the definition of a *joint* structure which contains all low-level control information relevant to an actuator. Each actuator has associated with it a joint structure, and all low-level subroutines accept as input a pointer to a specific joint structure. In this way, the software can be used independent of the number of actuators being controlled.

Figure A3.4 describes the functions of the low-level software drivers.

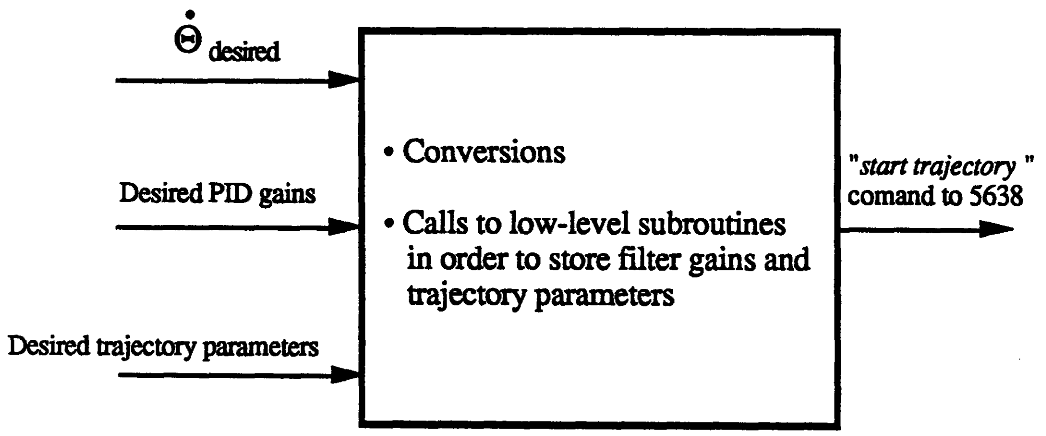
$\dot{\Theta}_{desired}$
Desired PID gains
Desired trajectory parameters
• Conversions
• Calls to low-level subroutines in order to store filter gains and trajectory parameters
"*start trajectory* " comand to 5638

Figure A3.4 Functional diagram of low-level software drivers.

The following is a program listing of the main program used to run the experiments. This program contains the controller implementation for straight-line sliding-object motion. The program calls several low-level functions which are defined categorically in the files mentioned above. All source code pertaining to low-level software drivers is located in the following directory

C:\heman\motionc

The files are named slop.XXX where XXX corresponds to the extensions defined above. The executable files for running experiments with straight-line and curvilinear motion are called *slopline.exe* and *slopcurv.exe* respectively.

Also included in the appendix is source code for the straight-line and curvilinear motion simulations.

```
/* *************                    line.c:                                   *******
********            main program containing high-level control loop for       *******
** *****            controlling straight-line motion. The program calls       *******
*******             many low-level routines defined in other files.           *******

** The low-level drivers are largely an extension and modification of the source
** code provided with the 5638 board. I am using the same library of
** LM628 functions as well as the high-level macros written by National
 ** Semiconductor. Mostly, I have modified the code to support multiple axes.
**
** latest revision:  20 August 90  (by H. Lakhani)
**
*/

#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>
#include <dos.h>
#include <signal.h>
#include <stdlib.h>
#include <malloc.h>
#include <math.h>
#include <sys/timeb.h>

#define PUBLIC
#include  "inc.h"


struct rccoord gettextposition(void)
{
        union REGS xr;
        struct rccoord screen;

        xr.h.ah = 3;
        xr.h.bh = 0;
        int86(0x10,&xr,&xr);
        screen.row = xr.h.dh + 1;
        screen.col = xr.h.dh + 1;
        return(screen);
}


void settextposition(short row, short col)
{
        union REGS xr;

        xr.h.ah = 2;
        xr.h.bh = 0;
        xr.h.dh = row-1;
        xr.h.dl = col-1;
        int86(0x10,&xr,&xr);
}

void clearscreen(void)
{
        union REGS xr;
        void settextposition();

        xr.x.ax = 0x0600;
        xr.h.bh = 7;
        xr.x.cx = 0;
        xr.x.dx = 0x1950;
        int86(0x10,&xr,&xr);
        settextposition(1,1);
}
        struct cartvel{
                float x;  /* in mm/sec       */
                float y;
        } cv, ctoolv, *cvel, *ctoolvel;

        struct jointvel     {
                float d;  /* prismatic joint (mm/sec)          */
                float theta;         /* revolute joint (tenths of radian/sec) */
        } jv, *jvel;

        struct statevars    {
                float phi;
                float d;
                float u;
         } ste, *State, st[300];

main()
{
         extern void initpgm(), init628(), checksys(), initPIC(void),install();
         void byebye(), c2jvel(), toolc2jvel();
        extern void interrupt far handler();
        void clearscreen();
         char valid, *sptr, *bufptr, c = 'y';
         int i, flag = 1, filenum, index;
         float cmdtheta, k1, k2, kint, intsum, offset, conv1, conv2, conv3;
         float dtime, temp, j1dist, th, xcx[300], xcy[300];
        long temp1;
        unsigned temp2;
         FILE *fp, *fopen();
         struct timeb ot, nt, sst, et, *otime, *ntime, *stime, *etime;
        struct path {
                char drive[_MAX_DRIVE];
                char dir[_MAX_DIR];
                char fname[_MAX_FNAME];
                char ext[_MAX_EXT];
                } *pf;


   printf( "\n" );
  clearscreen();

  Joint0 = &Jnt0;
  Joint1 = &Jnt1;
  Bothjoints = &Bothjnts;
  cvel = &cv;      /*cartesian velocity referenced to base frame  */
  jvel = &jv;
  ctoolvel = &ctoolv; /* cartesian velocity in tool frame          */
  State = &ste;
  otime = &ot;
  ntime = &nt;
  stime = &sst;
  etime = &et;

 /* Board Address, Global Address, and Interrupt Request number are
        initialized in 'slopini.h'    */

/* Set absolute position breakpoint for prismatic joint          */
  Joint1->absbrkpt = -8000;
  sbpa(Joint1);

/* install <Ctrl>C handler */
/*        if (signal(SIGINT,byebye) == SIG_ERR)
                printf( "<Ctrl>C error\n ");
*/

/* Initialization sequence   */
        initpgm();        /* initialize some program variables          */

        /* program sign-on */
         printf(" \n\n\n****  Welcome to BATTLECAT **** \n\n\n ");

        init628(Joint0); /* initialize LM628 filter; clear trajectory */
        init628(Joint1);
        checksys();       /* check out LM628 hardware initialization */
        initPIC(); /* initialize 5638 interrupt controller */
         install(handler,Intnr); /* enable host interrupt handler */

         printf( "\n "); /* esthetics */

        clearscreen();
        while(flag)       {
        /* Initialize aux. encoder and prompt user to initialize the state
                of the object      */
        clearscreen();

        /* Prompt for number of data file   */
         printf( "Enter number of data file (1-8)==>  ", &filenum);
         scanf( "%d ", &filenum);
        if(filenum ==1)
                fp = fopen( "linesim1.dat ",  "w ");
        else if(filenum == 2)
                fp = fopen( "linesim2.dat ",  "w ");
        else if(filenum == 3)
                fp = fopen( "linesim3.dat ",  "w ");
        else if(filenum == 4)
                fp = fopen(" linesim4.dat" , " w" );
        else if(filenum == 5)
                fp = fopen( "linesim5.dat ",  "w ");
        else if(filenum == 6)
                fp = fopen( "linesim6.dat ",  "w ");
        else if(filenum == 7)
                fp = fopen( "linesim7.dat ",  "w " );
        else if(filenum == 8)
                fp = fopen( "linesim8.dat ",  "w ");
        else    fp = fopen( "linesim.dat ",  "w ");

         printf( "RESET AUX ENCODER TO HOME POSITION AND PRESS ANY KEY\n ");
```

```
while(!kbhit())    {}
getch();
 outp(AUX_ENC_RESET, 0x00);

 printf( pLACE OBJECT IN DESIRED ORIENTATION AND THEN PRESS ANY KEY\n ");
while(!kbhit())    {
          rdauxenc();
           settextposition(10,1);
           printf( "Auxenc = %d\r ", Auxenc);
           settextposition(11,1);
          printf( "Object orientation = %f degrees\r ", Auxenc*(360.0/CPR2));
}
getch();
State->phi = (float)(Auxenc*2*PI/CPR2);

printf( "\n ");

printf( "Enter perpendicular deviation from desired trajectory (mm) ==>  ");
scanf( "%f ", &offset);
State->d = offset;

printf( "\nState->phi = %f,\t State->d = %f\n ", State->phi, State->d);"

/* prompt for feedback gains          */
printf( "Enter gain Kphi==>  ");"
scanf( "%f ", &k1);
printf( "Enter gain Kd==>  ");"
scanf( "%f ", &k2);
printf( "Enter gain Ki==>  ");"
scanf( "%f ", &kint);

intsum = 0.0;

/* Write filter and trajectory data to joints    */
Joint0->kp = 750;          /* Joint 0        */
Joint0->ki = 10;
Joint0->ilimit = 50;
lkp(Joint0);
lki(Joint0);
lilimit(Joint0);
vmode(Joint0);
Joint0->cmdacc = 20;
lacc(Joint0);
Joint0->cmdvel = 0;
lvel(Joint0);
udf(Joint0);

Joint1->kp = 750;          /* Joint 1        */
Joint1->ki = 10;
Joint1->ilimit = 50;
lkp(Joint1);
lki(Joint1);
lilimit(Joint1);
vmode(Joint1);
rev(Joint1);
Joint1->cmdacc = 20;
lacc(Joint1);
Joint1->cmdvel = 0;
```

```
 lvel(Joint1);
 udf(Joint1);

 printf( "HIT ANY KEY TO START CONTROL LOOP...\n ");"
 while(!kbhit())     {}
 getch();

 "/* Since we are dealing with absolute trajectory values, I'll"
   just send the 'stt' command directly to the Global address port.
The normal stt() subroutine in ins.c checks for relative
   parameters and then subsequently updates the absolute values  */
 "wrcmd(Bothjoints, STT);"

 clearscreen();

 printf( "HIT ANY KEY TO END SEQUENCE...\n ");"

 index = 0;

 st[index].phi = State->phi;
 st[index].d = State->d;
 xcx[index] = 0.0;
 xcy[index] = State->d;

 ftime(stime);
 ftime(otime);

 while(!kbhit())     {

 index++;
 /* Compute input (pushing angle) as a functin of current state              */
 cmdtheta = -k1*(State->phi) - k2*(State->d) + intsum;
 if(fabs(cmdtheta) > INPUTSAT)     {         /* Saturation        */
           if (cmdtheta > 0.0)
                     cmdtheta = (float)(INPUTSAT);
           else cmdtheta = -(float)(INPUTSAT);
 }
 st[index-1].u = cmdtheta;

 /* Compute cartesian tip velocity */
 ctoolvel->x = VO*cos(State->phi + cmdtheta);
 ctoolvel->y = VO*sin(State->phi + cmdtheta);

 /* multiply by inverse Jacobian to get joint velocities           */
 toolc2jvel();

 /* Convert prismatic joint velocity to motor shaft velocity      */
 conv1 = CPR*LEADSCREW/(10*2.54);

 jvel->d *= conv1;            /* counts/sec       */

 /* Convert revolute joint velocity to counts/sec         */
 conv2 = CPR/(2*PI);
 jvel->theta *= conv2;      /* (counts/sec)     */

 if(jvel->theta > 0)          {
           fwd(Joint0);
 }
else	{
		rev(Joint0);
		jvel->theta *= -1.0;
}

conv3 = 341*MICROSEC*65536;
Joint0->cmdvel = (long int)((jvel->theta)*conv3);
lvel(Joint0);

Joint1->cmdvel = (long int)((jvel->d)*conv3);
lvel(Joint1);

wrcmd(Bothjoints,STT);

/* compute new state values		*/
rdauxenc();
rdrp(Joint0);
rdrp(Joint1);
State->phi = (float)(2.0*PI*((float)(Auxenc)/CPR2 + (float)(Joint0->realp)/CPR));


/* Use kinematics to determine state 'd' from joint angles		*/
th = (float)((Joint0->realp)*2.0*PI/CPR);
j1dist = (Joint1->realp)*25.4/(CPR*LEADSCREW);

State->d = offset + RADIUS*sin(th);
xcy[index] = State->d;
xcx[index] = -j1dist + RADIUS*(1 - cos(th));

/* Comput integral sum	*/
		ftime(ntime);

		temp1 = ntime->time - otime->time;
		if (ntime->millitm > otime->millitm) {
				temp2 = ntime->millitm - otime->millitm;
		}
		else {
				temp2 = 1000 - (otime->millitm - ntime->millitm);
				temp1--;
		}

		dtime = temp1 + temp2/1000.0;

if (fabs(State->d) > 1.0)
		intsum += -kint*dtime*State->d;
else intsum += 0.0;

otime->time = ntime->time;
otime->millitm = ntime->millitm;

st[index].phi = State->phi;
st[index].d = State->d;

}
getch();

panic(Joint0);
panic(Joint1);
```

```
        ftime(etime);

        st[index].u = st[index-1].u;
        clearscreen();
        settextposition(2,1);
        printf( "state.phi = %f\tstate.d = %f\n ", State->phi, State->d);"
        settextposition(10,1);
        printf( "HIT ANY KEY TO RETURN TO HOMEPOSITION...\n ");"
        while(!kbhit())     {}
        getch();

        rtnhome();

        /* Save data to file            */
        settextposition(11,1);
        printf( "SAVING DATA....  ");"

        fprintf(fp,  "%d\n ", index);
        fprintf(fp,  "%f\t%f\t%f\n ", k1, k2, kint);
        temp = 0.0;
        temp1 = etime->time - stime->time;
        dtime = (float)(temp1)/(float)(index);
        for (i = 0; i <= index; i++)            {
                fprintf(fp,  "%f\t%f\t%f\t%f\n ", temp, st[i].phi, st[i].d,st[i].u);
                fprintf(fp,  "%f, %f\n ", xcx[i], xcy[i]);
                temp += dtime;
        }
        fclose(fp);
        printf( "DONE\n ");

        settextposition(22,1);
        printf( "Another try?(y/n)... ");
        c = getchoice();
        if (c == 'n')
                flag = 0;

}


        byebye();

}


void byebye(void)
{

        extern void cleanup(int num);
        extern char getchoice();
         void off(), rdstatus();
        char c = 'n';

        /* ignore further <Ctrl>C interrupts            */
        rdstatus(Joint0);
        rdstatus(Joint1);
        if (!(Joint0->status & 0x80) && !(Joint1->status & 0x80)) {
                 settextposition(22,1);
                printf( "Keep motors on? (y/n)... ");
                c = getchoice(c);
                if (c != 'y')
                        off(Joint0);
                        off(Joint1);
                printf( "\n ");
        }
        cleanup(Intnr);

        /* Restore DOS default <Ctrl>C handler      */
/*      signal(SIGINT, SIG_DFL);
*/
        exit(0);
}


void c2jvel()
{
        float invjac[2][2], theta;
        extern struct cartvel *cvel;
        extern struct jointvel *jvel;

        rdrp(Joint0);
        theta = (Joint0->realp)*2*PI/CPR;
        printf( "theta = %f\n ", theta);

        invjac[0][0] = 1.;
        invjac[0][1] = tan(theta);
        invjac[1][0] = 0.;
        invjac[1][1] = 1/(RADIUS*cos(theta));

        jvel->d = (invjac[0][0])*(cvel->x) + (invjac[0][1])*(cvel->y);
        jvel->theta = (invjac[1][0])*(cvel->x) +(invjac[1][1])*(cvel->y);
}

void toolc2jvel()
{
        float invjac[2][2], theta2;
        extern struct cartvel *ctoolvel;
        extern struct jointvel *jvel;

        rdrp(Joint0);
        theta2 = (Joint0->realp)*2.0*PI/CPR;

        invjac[0][0] = 1.0/cos(theta2);
        invjac[0][1] = 0.0;
        invjac[1][0] = tan(theta2)/RADIUS;
        invjac[1][1] = 1.0/RADIUS;

        jvel->d = (invjac[0][0])*(ctoolvel->x) + (invjac[0][1])*(ctoolvel->y);
        jvel->theta = (invjac[1][0])*(ctoolvel->x) + (invjac[1][1])*(ctoolvel->y);

}


rtnhome()       /* Returns the mechanism to its 'home' position      */
{
        pmode(Joint0);
        Joint0->cmdpos = 0;
        lpos(Joint0);
        Joint0->cmdvel = 100000;
        lvel(Joint0);

        pmode(Joint1);
        Joint1->cmdpos = 0;
        lpos(Joint1);
        Joint1->cmdvel = 200000;
        lvel(Joint1);

        wrcmd(Bothjoints, STT);

}
```

```
/*************************************************************************
                                        linesim.c

                source code for simulating straight-line control of sliding objects.
*************************************************************************/

#include <stdio.h>
#include <math.h>

#define TIME 31.0
#define DT 0.23
#define VO 10.0
#define DELTA 35
#define SIDE 50
#define SCALE 50
#define D 20.0              /* initial d        */
#define PHI 20.0            /* initial phi = degrees */
#define INPUTSAT 0.5236         /* 30 degrees       */

struct state  {
        float phi;
        float d;
} State[1000];

float Input, K1, K2,Ki,Intsum;


main()
{

        int i, sgnw = 1, data1, data2, data3;
        float phiref, dref, phi, d, phierr, derr;
        float command, theta, radius, w, dx, dy;
        float rx, ry;
        float time[1000], aa[1000], bb[1000], cc[1000], dd[1000];
        float phider(), dder();
        struct point {
                float x;  /* global          */
                float y;
                float xloc;        /* local  */
                float yloc;
        }a, c, p, cr, bl,br,tr,tl;

/* 'a' is contact point = bottom left corner of box.
                                        'c' is the top right corner of box.
                                        The line through a-p is perpendicular to l2          */

                struct rect           {
                struct point topleft;
                struct point btmright;
        } box;

        struct line          {
                float slope;
                float intercept;
        } l1, l2, v, r, loci;
```

```
FILE *fp1,*fp2, *fopen();

fp1 = fopen("rksim3m", "w");
fp2 = fopen("rksim3k", "w");

printf("Enter gain K1==> ");
scanf("%f", &K1);
printf("Enter gain K2==> ");
scanf("%f", &K2);
printf("Enter gain Ki==> ");
scanf("%f", &Ki);

/* Initialize state              */
State[0].phi = PHI*PI/180.0;          /* phi     */
State[0].d = D;

time[0] = 0.0;

/* Write initial values to matlab file          */
fprintf(fp1,phi(1) = %f;\td(1) = %f;\tt(1) = %f;\n", State[0].phi,State[0].d,time[0]);
fprintf(fp1,"u(1) = %f;\n", Input);
fprintf(fp2,"%f\t%f\t%f\t%f\n", time[0], State[0].phi,State[0].d,Input);


/* define coordinates of box. The local coordinatees are referenced to an axis whose origin
          is the pushing point          */
bl.x = 0.0;
bl.y = State[0].d;
br.xloc = SIDE*cos(PI/4);
br.yloc = -SIDE*sin(PI/4);
br.x = br.xloc*cos(State[0].phi) - br.yloc*sin(State[0].phi);
br.y = br.xloc*sin(State[0].phi) + br.yloc*cos(State[0].phi)+bl.y;
tr.xloc = SIDE*sqrt(2.0);
tr.yloc = 0.0;
tr.x = tr.xloc*cos(State[0].phi) - tr.yloc*sin(State[0].phi);
tr.y = tr.xloc*sin(State[0].phi) + tr.yloc*cos(State[0].phi)+bl.y;
tl.xloc = SIDE*cos(PI/4);
tl.yloc = SIDE*sin(PI/4);
tl.x = tl.xloc*cos(State[0].phi) - tl.yloc*sin(State[0].phi);
tl.y = tl.xloc*sin(State[0].phi) + tl.yloc*cos(State[0].phi)+bl.y;

/* Write data to matlab file          */
fprintf(fp1,"x(1) = %f;\ty(1) = %f;\n", bl.x, bl.y);
fprintf(fp1,plot(x(1),y(1),'*');\n");
fprintf(fp1,"x(2) = %f;\ty(2) = %f;\n", tr.x, tr.y);
fprintf(fp1,plot(x,y,'-');\n");

Intsum = 0.0;
data1 = 1;
data2 = 1;

/* Begin control loop      */
for (i = 0;i<=TIME/DT;i++)          {
          time[i+1] = time[i] + DT;


          if (fabs(State[i].d) > .2)     /* Some value which we will consider to be close
                                                  enough to zero     */
```

```
                 Intsum += -Ki*State[i].d;
                else Intsum += 0.0;

                Input = K1*(Phiref-State[i].phi) - K2*State[i].d + Intsum;

                if(fabs(Input) > INPUTSAT) {        /* Input saturation  */
                  if(Input > 0.0)
                    Input = INPUTSAT;
                  else Input = -INPUTSAT;
                }

                /* nonlinear model             */
/*              phider = -VO*sin(Input)/DELTA);
*/
                /* linear model    */
                phider =  - VO*Input/DELTA ;

                /* Compute new phi             */
                aa[i] = DT*phider;
                bb[i] = DT*phider;
                cc[i] = DT*phider;
                dd[i] = DT*phider;
                State[i+1].phi = State[i].phi + (aa[i] + 2*bb[i] + 2*cc[i] + dd[i])/6;

                /* nonlinear model             */
/*              dder = VO*sin(State[i+1].phi + Input);
*/
                /* linear model    */
                dder = VO*(State[i+1].phi + Input);

                aa[i] = DT*dder;
                bb[i] = DT*dder;
                cc[i] = DT*dder;
                dd[i] = DT*dder;
                State[i+1].d = State[i].d + (aa[i] + 2*bb[i] + 2*cc[i] + dd[i])/6;

                fprintf(fp1,phi(%d) = %f;\td(%d) = %f;\tt(%d) = %f;\n",
                        i+2,State[i+1].phi,i+2,State[i+1].d, i+2,time[i+1]);
                fprintf(fp1,"u(%d) = %f;\n", i+2,Input);

                fprintf(fp2,"%f\t%f\t%f\t%f\n", time[i+1], State[i+1].phi,State[i+1].d,Input);


                /* define coordinates of box */
                bl.x += VO*cos(Input + State[i+1].phi)*DT;
                bl.y = State[i+1].d;
                br.xloc = SIDE*cos(PI/4);
                br.yloc = -SIDE*sin(PI/4);
                br.x = br.xloc*cos(State[i+1].phi) - br.yloc*sin(State[i+1].phi)+bl.x;
                br.y = br.xloc*sin(State[i+1].phi) + br.yloc*cos(State[i+1].phi)+bl.y;
                tr.xloc = SIDE*sqrt(2.0);
                tr.yloc = 0.0;
                tr.x = tr.xloc*cos(State[i+1].phi) - tr.yloc*sin(State[i+1].phi)+bl.x;
                tr.y = tr.xloc*sin(State[i+1].phi) + tr.yloc*cos(State[i+1].phi)+bl.y;
                tl.xloc = SIDE*cos(PI/4);
                tl.yloc = SIDE*sin(PI/4);
                tl.x = tl.xloc*cos(State[i+1].phi) - tl.yloc*sin(State[i+1].phi)+bl.x;
                tl.y = tl.xloc*sin(State[i+1].phi) + tl.yloc*cos(State[i+1].phi)+bl.y;
```

```
		data3 = data1 + data2;

		if (data3 > 1)		{		/* Only write out every fourth data point
							(for plotting purposes) */
		fprintf(fp1,"x(1) = %f;\ty(1) = %f;\n", bl.x, bl.y);
		fprintf(fp1,plot(x(1),y(1),'*');\n");
		fprintf(fp1,"x(2) = %f;\ty(2) = %f;\n", tr.x, tr.y);
		fprintf(fp1,plot(x,y,'-');\n");
		}

		if (data1 < 0)
			data2 *= -1;
		data1 *= -1;
	}

	fclose(fp1);
	fclose(fp2);

}
```

```
/************************************************************************
                                        curvesim.c

                Source code for simulating curvilinear motion of sliding objects.

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

#include <stdio.h>
#include <math.h>

#define TIME 30.0
#define DT 0.26
#define VO 10.0
#define DELTA 35.0
#define SIDE 50.0
#define SCALE 50.0
#define CRX 100.0
#define CRY -250.0
#define RHO 240.0
#define STARTPSI 111
#define DD 29.25            /* initial d          */
#define PHI 0.0             /* initial phi in degrees */
#define ROOT2 1.4142136
#define INPUTSAT 0.5236          /* 30 degrees       */
#define LINEAR 1


float Input,K1, K2, Ki, Phiref, Intsum;
struct state {
        float phi;
        float d;
} State[1000];

main()
{

        int i, data1, data2, data3;
        float dpsi, rotangle;
        float time[1000], aa[1000], bb[1000], cc[1000], dd[1000];
        float phider, dder;
        struct point {
                float x;  /* global          */
                float y;
                float xloc;        /* local  */
                float yloc;
        }a, b, c, d;        /* 'a' is contact point = bottom left corner of box. 'c' is the top right
                                corner of box. The line through a-p is perpendicular to l2        */

        struct polarpoint {
          float r;
          float psi;
        } apolar;
        struct rect         {
                struct point topleft;
                struct point btmright;
        } box;
```

```
struct line         {
            float slope;
            float intercept;
} l1, l2, m;

FILE *fp1, *fp2, *fopen();

fp1 = fopen("curve4m", "w");
fp2 = fopen("curve4k", "w");

/* prompt for feedback gains          */
printf("Enter gain K1--> ");
scanf("%f", &K1);
printf("Enter gain K2-->  ");
scanf("%f", &K2);
printf("Enter gain Ki-->  ");
scanf("%f", &Ki);

printf("%f\t%f\t%f\n", K1, K2, Ki);

/* initialize states          */
State[0].phi = PHI*PI/180;
State[0].d = DD;

Phiref = -DELTA/RHO;

apolar.r = RHO+State[0].d;
apolar.psi = STARTPSI*PI/180;
a.x = apolar.r*cos(apolar.psi)+CRX;
a.y = apolar.r*sin(apolar.psi)+CRY;


/* Compute the slope of the line, m,connecting point 'a' to the desired
   center of rotation. From this and 'phi' we get the
   orientation of the box */
m.slope = (a.y - CRY)/(a.x-CRX);
l2.slope = -1/m.slope;
l1.slope = tan(State[0].phi) +l2.slope;
l1.slope = atan(l1.slope); /* convert to radians */
rotangle = -(State[0].phi + apolar.psi- PI/2);
c.xloc = SIDE*ROOT2;
c.yloc = 0.0;
c.x = c.xloc*cos(rotangle) + c.yloc*sin(rotangle) + a.x;
c.y = -c.xloc*sin(rotangle) + c.yloc*cos(rotangle) + a.y;
b.xloc = DELTA;
b.yloc = -DELTA;
b.x = b.xloc*cos(rotangle) + b.yloc*sin(rotangle) + a.x;
b.y = -b.xloc*sin(rotangle) + b.yloc*cos(rotangle) + a.y;
d.xloc = DELTA;
d.yloc = DELTA;
d.x = d.xloc*cos(rotangle) + d.yloc*sin(rotangle) + a.x;
d.y = -d.xloc*sin(rotangle) + d.yloc*cos(rotangle) + a.y;

time[0] = 0.0;
printf(phiref = %f\tphi = %f\td = %f\n", Phiref, State[0].phi, State[0].d);

fprintf(fp1,"x(1) = %f;\ty(1) = %f;\n", a.x, a.y);
fprintf(fp1,plot(x(1),y(1),'*');\n");
fprintf(fp1,"x(2) = %f;\ty(2) = %f;\n", c.x, c.y);
fprintf(fp1,plot(x,y,'-');\n");

fprintf(fp1,phi(1) = %f;\td(1) = %f;\tt(1) = %f;\n",State[0].phi,State[0].d,time[0]);
fprintf(fp1,"u(1) = 0.0;\n");

fprintf(fp2,"%f\t%f\t%f\n",time[0],State[0].phi,State[0].d);

Intsum = 0.0;
data1 = 1;
data2 = 1;

/* Begin control loop        */
for (i = 0;i<=TIME/DT;i++){
        time[i+1] = time[i] + DT;

        if (fabs(State[i].d) > .2)     /* Some value which we will consider to be
                                            close enough to zero        */
          Intsum += -Ki*State[i].d;
        else Intsum += 0.0;

        Input = K1*(Phiref-State[i].phi) - K2*State[i].d + Intsum + DELTA/RHO;

        if(fabs(Input) > INPUTSAT)          { /* Input saturation at 45 degrees */
          if(Input > 0.0)
            Input = INPUTSAT;
          else Input = -INPUTSAT;
        }

        /* nonlinear model           */
/*      phider = VO*(cos(State[i].phi + Input)/(RHO+State[i].d) - sin(Input)/DELTA);
*/
        /* linear model    */
        phider = -VO*State[i].d/(RHO*RHO) - VO*Input/DELTA + VO/RHO;

        /* Compute new phi          */
        aa[i] = DT*phider;
        bb[i] = DT*phider;
        cc[i] = DT*phider;
        dd[i] = DT*phider;
        State[i+1].phi = State[i].phi + (aa[i] + 2*bb[i] + 2*cc[i] + dd[i])/6;

        /* nonlinear model           */
/*      dder = VO*sin(State[i+1].phi + Input);
*/
        /* linear model    */
        dder = VO*(State[i+1].phi + Input);

        aa[i] = DT*dder;
        bb[i] = DT*dder;
        cc[i] = DT*dder;
        dd[i] = DT*dder;
        State[i+1].d = State[i].d + (aa[i] + 2*bb[i] + 2*cc[i] + dd[i])/6;

        fprintf(fp1,phi(%d) = %f;\td(%d) = %f;\tt(%d) = %f;\n", i+2, State[i+1].phi,
                i+2,State[i+1].d, i+2,time[i+1]);
        fprintf(fp1,"u(%d) = %f;\n", i+2,Input*180/PI);

        fprintf(fp2,"%f\t%f\t%f\t%f\n",time[i+1],State[i+1].phi,State[i+1].d, Input);

        dpsi = VO*cos(State[i+1].phi + Input)*DT/(RHO+State[i+1].d);

        apolar.r = RHO+State[i+1].d;
        apolar.psi -= dpsi;
        a.x = apolar.r*cos(apolar.psi)+CRX;
        a.y = apolar.r*sin(apolar.psi)+CRY;

        /* Compute the slope of the line, m, connecting point 'a' to the desired
           center of rotation. From this and 'phi'  we get the orientation of the box */
        m.slope = (a.y - CRY)/(a.x-CRX);
        l2.slope = -1/m.slope;
        l1.slope = tan(State[i+1].phi) +l2.slope;
        l1.slope = tan(l1.slope); /* convert to radians */
        rotangle = -(State[i+1].phi + apolar.psi- PI/2);
        c.x = c.xloc*cos(rotangle) + c.yloc*sin(rotangle) + a.x;
        c.y = -c.xloc*sin(rotangle) + c.yloc*cos(rotangle) + a.y;
        b.x = b.xloc*cos(rotangle)+b.yloc*sin(rotangle)+a.x;
        b.y = -b.xloc*sin(rotangle)+b.yloc*cos(rotangle)+a.y;
        d.x = d.xloc*cos(rotangle)+d.yloc*sin(rotangle)+ a.x;
        d.y = -d.xloc*sin(rotangle)+d.yloc*cos(rotangle)+a.y;

        data3 = data1 + data2;
        if (data3 > 1) {    /* write out every fourth data point (for plotting purposes)      */
        fprintf(fp1,"x(1) = %f;\ty(1) = %f;\n", a.x, a.y);
        fprintf(fp1,plot(x(1),y(1),'*');\n");
        fprintf(fp1,"x(2) = %f;\ty(2) = %f;\n", c.x, c.y);
        fprintf(fp1,plot(x,y,'-');\n");
        }

        if (data1 < 0)
                data2 *= -1;
        data1 *= -1;

     }
   fclose(fp1);
   fclose(fp2);
}
```

References

Benjamin, Michael and Robert M. Podoloff, "A Tactile Sensor for Analyzing Dental Occlusion," SENSORS, vol. 6, no. 3, 1985.

Craig, John J., *Introduction to Robotics, Mechanics, and Control*, Addison-Wesley, Reading, Massachusetts, 1986.

Driels, M.R., "Pose Estimation Using Tactile Sensor Data for Assembly Operations," *Proc. IEEE International Conference on Robotics and Automation*, pp. 1255-1261, 1986.

Fearing, R.S., "Implementing a Force Strategy for Object Reorientation," *Proc. IEEE International Conference on Robotics and Automation*, pp. 96-102, 1986.

Fearing, R.S., "Some Experiments With Tactile Sensing During Grasping," *Proc. IEEE International Conference on Robotics and Automation*, pp. 1637-1643, 1987.

Friedland, Bernard, *Control System Design: An Introduction to State-Space Methods*, McGraw Hill, New York 1886.

Mason, Mathew T., *Robot Hands and the Mechanics of Manipulation*, The MIT Press, Cambridge, Massachusetts, 1985.

Salisbury, J. Kenneth, *Robot Hands and the Mechanics of Manipulation*, The MIT Press, Cambridge, Massachusetts, 1985.

Strang, Gilbert, *Introduction to Applied Mathematics*, Wellesley-Cambridge Press, Wellesley, Massachusetts, 1986.