HYBRID CONSTRAINT SPACE POSITION/FORCE CONTROL

By

Roger Wong

B. A. Sc. (Mechanical Engineering) University of British Columbia

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

MECHANICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

June 1996

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of MECHANICAL ENGINEERING
FACULTY OF APPLIED SCIENCE
The University of British Columbia
Vancouver, Canada

Date August 14, 1996

Abstract

This thesis documents the conceptual development of the author's contribution, the hybrid constraint space position/force controller for robotic manipulator control in constrained environments. This method is built upon a constraint space dynamic model, where the model parameters are displacement along the constraint trajectory and normal force between the manipulator end-effector and environment. This dynamic model is constructed by transforming conventional joint space manipulator dynamics into their constraint space equivalents through the application of mapping functions, which relate differential displacements and velocities in the constraint space coordinate system to the joint space coordinate system. Conventional PD controllers may then be applied to the simplified dynamic structure of the constraint space equations of motion, in order to produce a vector of manipulator joint torques which will satisfy both position and force requirements along the environmental constraint. Actuator constraints and momentum compensating techniques are also used to ensure that the position and force control problems are completely decoupled from one another. This modelling technique is then applied to the control problem for a two degree of freedom prismatic robot as an illustrative example. Simulation of this specific controller is carried out with respect to three different constraint surfaces, a planar, a concave circular and a convex circular environment. The results of these simulations show that the hybrid constraint space controller provides excellent position and force trajectory tracking for the planar case study. This thesis is intended to be the forerunner to future work which will develop in detail, the application of hybrid constraint space control to highly nonlinear manipulator/constraint models.

Table of Contents

Abstract ii

List of Figures vi

List of Tables ix

Nomenclature x

Acknowledgements xii

1 Introduction 1

1.1 Opening Remarks 1

1.2 Motivation 2

1.3 Contributions of the Thesis 3

1.4 Outline of the Thesis 4

2 Literature Review 5

2.1 Stiffness Control 6

2.2 Hybrid Position/Force Control 8

2.3 Hybrid Impedance Control 11

2.4 Reduced State Position/Force Control 12

2.5 Summary 14

3 Hybrid Constraint Space (HCS) Control 16

3.1 Definition of a Constraint Space Coordinate System 19

3.2 Derivation of the Conventional Joint Space Dynamics 20
3.3 Derivation of Mapping Functions 21
3.4 Formulation of the Constraint Space Dynamics 22
3.5 Actuator Constraints Required for Maintaining End-Effector Trajectory . 24
3.6 Derivation of a Position Control Law 28
3.7 Derivation of a Force Control Law 29
3.8 Superposition 30

4 Derivation of an HCS Controller for a 2DOF Prismatic Robot 31
4.1 Definition of a Constraint Space Coordinate System 31
4.2 Derivation of the Conventional Joint Space Dynamics 32
4.3 Derivation of Mapping Functions 33
4.4 Formulation of the Constraint Space Dynamics 34
4.5 Actuator Constraints Required for Maintaining End-Effector Trajectory . 36
4.6 Derivation of a Position Control Law 38
4.7 Derivation of a Force Control Law 39
4.8 Superposition 41

5 Simulation Results for a 2DOF Prismatic Robot with HCS Control 42
5.1 Planar Constraint Surface 42
5.2 Concave Circular Constraint Surface 46
5.3 Convex Circular Constraint Surface 52
5.4 Comparative Discussion 59

6 Conclusions & Summary 61
6.1 Conclusions 61
6.2 Summary 61

6.3 Future Work 62

Bibliography **63**

Appendix A Software Coding for Simulations Performed in Matlab **65**

List of Figures

1.1 Research Tree in the Field of Robotics 2

2.2 Stiffness Control:SDOF Manipulator 6

2.3 Hybrid Position/Force Control: 2DOF Manipulator 9

3.4 Procedural Flow Chart for a Hybrid Constraint Space Controller 17

3.5 General Constraint Space Coordinate System 19

3.6 nDOF Manipulator Constrained to Arbitrary Surface 20

3.7 Generalized Constraint Forces Q_s and Q_n 25

3.8 Momentum Compensation for Curved Surfaces 27

4.9 2DOF Planar Robot with Arbitrary Constraint 32

5.10 Planar Constraint Surface 43

5.11 Task Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Planar Constraint Surface with Position Control 44

5.12 Constraint Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Planar Constraint Surface with Position Control 45

5.13 Contact Force Simulation Results for the 2DOF Prismatic Robot Constrained to a Planar Constraint Surface with Position Control 45

5.14 Task Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Planar Constraint Surface with Position/Force Control 47

5.15 Constraint Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Planar Constraint Surface with Position/Force Control . . . 47

5.16 Contact Force Simulation Results for the 2DOF Prismatic Robot Constrained to a Planar Constraint Surface with Position/Force Control . . . 48
5.17 Concave Circular Constraint Surface 49
5.18 Task Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Concave Circular Constraint Surface with Position Control 50
5.19 Constraint Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Concave Circular Constraint Surface with Position Control 51
5.20 Contact Force Simulation Results for the 2DOF Prismatic Robot Constrained to a Concave Circular Constraint Surface with Position Control 51
5.21 Task Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Concave Circular Constraint Surface with Position/Force Control . . 52
5.22 Constraint Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Concave Circular Constraint Surface with Position/Force Control 53
5.23 Contact Force Simulation Results for the 2DOF Prismatic Robot Constrained to a Concave Circular Constraint Surface with Position/Force Control 53
5.24 Convex Circular Constraint Surface 54
5.25 Task Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Convex Circular Constraint Surface with Position Control 55
5.26 Constraint Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Convex Circular Constraint Surface with Position Control . 56
5.27 Contact Force Simulation Results for the 2DOF Prismatic Robot Constrained to a Convex Circular Constraint Surface with Position Control . 56
5.28 Task Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Convex Circular Constraint Surface with Position/Force Control . . 57

5.29 Constraint Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Convex Circular Constraint Surface with Position/Force Control 58
5.30 Contact Force Simulation Results for the 2DOF Prismatic Robot Constrained to a Convex Circular Constraint Surface with Position/Force Control 58

List of Tables

5.1 Simulation Parameters for the 2DOF Prismatic Robot with HCS Control Constrained to a Planar Surface 43

5.2 Simulation Parameters for the 2DOF Prismatic Robot with HCS Control Constrained to a Concave Circular Surface 50

5.3 Simulation Parameters for the 2DOF Prismatic Robot with HCS Control Constrained to a Convex Circular Constraint Surface 54

Nomenclature

The following symbolic convention is with respect to the author's own material contained within Chapter 3, Chapter 4 and Chapter 5.

s	Constraint Space Position Coordinate
n	Constraint Space Force Coordinate
N_r	Contact Force at the End-Effector
$\underline{q}$	Manipulator Joint Space
L	Manipulator Lagrangian Potential Function
$\underline{Q}$	Manipulator Joint Torques
M	General Mass Matrix
C	General Coriolis/Centripetal Matrix
G	General Gravity Matrix
f_i	Differential Kinematic Mapping Function
g_i	Geometric Mapping Function
Q_s	Constraint Space Position Generalized Force
Q_n	Constraint Space Force Generalized Force
m_s	Constraint Space Equivalent Mass Term
c_s	Constraint Space Equivalent Coriolis/Centripetal Term
g_s	Constraint Space Equivalent Gravity Term
F	Q_s as a Function of Manipulator Joint Torques
h_i	Joint Space Acceleration as a Function of $\dot{s}$ and $\ddot{s}$
h_i^*	Joint Space Acceleration as a Function of $\ddot{s}$

F_i	Mass Coefficient for ith Manipulator Joint Equation of Motion
F_{ij}	Kinematic Manipulator Joint Torque Ratio Constraint
Q_{imc}	Momentum Compensation for ith Manipulator Joint Torque
Q_{ikin}	Position Control Component of Joint Torques
k_{pv}	Position Controller Velocity Gain
k_{pp}	Position Controller Position Gain
k_{fv}	Force Controller Velocity Gain
k_{fp}	Force Controller Position Gain
k_e	Environmental Stiffness
f_m	Measured Contact Force
Q_{inor}	Force Control Component of Joint Torques
H_i	Component of ith Joint Torque Required for Force Control
Q_{iact}	Superposed Position/Force ith Joint Torque

Acknowledgements

First and foremost, I would like to recognize the guidance and support I have received from my advisor, Dr. D. B. Cherchas throughout the last two years.

I would also like to thank Mai Lai and John for their patience, love, and encouragement. To my friends and the entire department of mechanical engineering here at U.B.C., thank you for listening.

Finally, for NSERC and ASI B.C., thank you for the financial assistance and sponsorship which made this project possible.

Chapter 1

Introduction

1.1 Opening Remarks

Research in the area of robotics is typically driven by two concerns; the desire to expand robotics applications and the desire to improve robot performance. The former usually involves interdisciplinary collaboration, and consists of automating industrial or commercial processes through the application of existing robot technology. The latter however, is primarily concerned with advances in manipulator technology itself. Together, the two fuel a technological revolution which has produced marvels ranging from automated assembly lines in the automotive industry to state-of-the-art robot surgeons in the operating room.

It is difficult to say which of these two research groups has wagered more heavily in the flourishing growth of the technology, for both have played an integral role. With the visionary research done in the area of applications, robotics continues to justify and magnify the funding which it receives from the various segments of industry, commerce and government. On the other hand, with the innovative research performed in the area of fundamentals, robotics is constantly increasing its adaptability and suitability as a transferable technology. The only certainty is continued success for the field in the years to come.

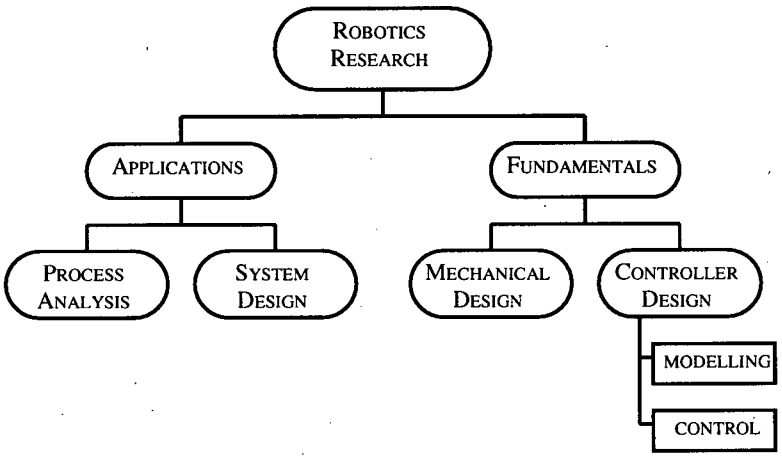

Figure 1.1: Research Tree in the Field of Robotics

1.2 Motivation

The author's work has the singular objective of advancing robot performance through the introduction of a new manipulator controller designed especially for use in constrained environments, thereby falling within the category of robotics research dedicated to fundamentals. A constrained environment is where the manipulator end-effector is restricted to movement along some arbitrary work space surface, a situation typically characterized by simultaneous position and force requirements. An industrial milling operation is a good example of a constrained environment.

Do constrained environments warrant such attention? Absolutely. The majority of industrial or commercial processes which may benefit through the application of robotic automation usually consist of constrained environments. Common but essential tasks such as assembly, cutting or grinding all require the tool to follow a specific path on the workpiece surface while simultaneously maintaining a prescribed normal interaction

force with that same surface. Hence, improvements in the area of constrained manipulator modelling and control will most certainly lead to advanced applications for robotic manipulators in even the most demanding industrial or commercial setting.

1.3 Contributions of the Thesis

Conventional approaches to the problem of constrained manipulator modelling and control are often devised in the joint space coordinate system of the robot in question. This joint space formulation is typically used because in actuality, the manipulator joints are the recipients of the control signals, and it is convenient to have the system dynamic model and the physical reality share a common coordinate system. However, when a manipulator is operating within a constrained environment, the joint space formulation is no longer appropriate, for two reasons. First, the joint space formulation is usually redundant. Second, the joint space formulation is inconsistent with the system control parameters, which usually consist of constraint surface variables such as displacement along the constraint surface or the normal force of interaction between the manipulator end-effector and the constraint surface. The major theoretical contribution of *hybrid constraint space control (HCS control)*, is a conceptual approach which addresses these two issues by solving the manipulator modelling and control problem for constrained environments with a constraint space dynamic model.

1.4 Outline of the Thesis

The purpose of this thesis, is to illustrate in detail the concept of hybrid constraint space control. Towards that end, the thesis will be structured as follows. The first chapter hereafter, entitled *Literature Review* will document the conceptual origins of hybrid constraint space control by surveying past research relevant to the work. The chapter entitled *Hybrid Constraint Space Control* will then discuss the method in a generalized fashion, with respect to arbitrary manipulators constrained to arbitrary surfaces. Next, the chapter entitled *Derivation of an HCS Controller for a 2DOF Prismatic Robot* will illustrate the general concepts discussed previously with a specific example, a two degree of freedom prismatic robot. The controller devised in this chapter will then be simulated for varying constraint environments within the following chapter, entitled *Simulation Results for a 2DOF Prismatic Robot with HCS Control.* Finally, conclusions regarding the work, and suggestions for future study will be presented through the chapter entitled *Conclusions*, to be followed by a bibliography of reference work and a list of supporting appendices.

Chapter 2

Literature Review

Hybrid constraint space control is unique because it employs a constraint space dynamic model. In other words, the model parameters deal exclusively with constraint conditions, such as end-effector displacement along the constraint surface or normal force levels between the manipulator end-effector and the environment. In hindsight, the employment of a constraint based model seems completely reasonable, since a process involving a constrained environment is almost always governed by constraint conditions. If so, why did researchers first begin with joint space formulations for their manipulator models?

When the first industrial manipulators were designed, robotic automation was only being applied to positioning processes such as spray-painting and welding. Because joint space solutions to these positioning problems were readily available through inverse kinematics, researchers came to perceive the joint space formulation as an undisputedly practical procedure. Consequently, when robotic automation began its integration with force control applications, researchers persisted in applying joint space formulations.

Gradually however, the preference for joint space approaches weakened as researchers began to recognize the important role played by the constraint surface in position/force control problems for constrained environments. The purpose of this literature review is to highlight the key conceptual developments of this progression, which ultimately lead to the development of hybrid constraint space control. Towards that end, this chapter

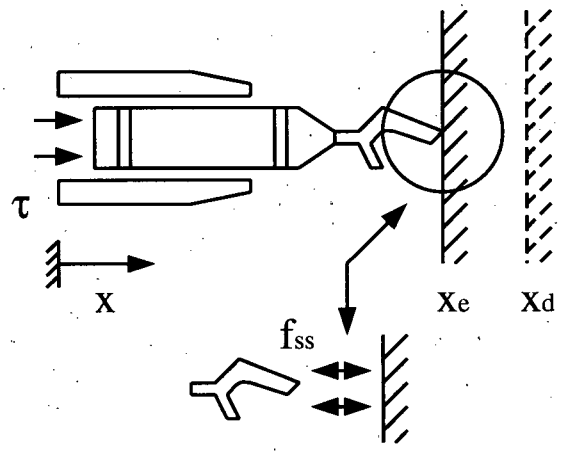

Figure 2.2: Stiffness Control:SDOF Manipulator

will include brief discussions on *stiffness control*, *hybrid position/force control*, *hybrid impedance control* and *reduced state position/force control*, as well as closing remarks regarding the influence of these methods in the author's formulation of hybrid constraint space control.

2.1 Stiffness Control

The concept of *stiffness control* was introduced to the literature by Salisbury and Craig in 1980, *Salisbury and Craig (1980)*, and is important because it represented a turning point in the direction of robotics research at the time. Previously, researchers had been primarily concerned with the robot position control problem. Stiffness control was the first method to formally address the force control problem, and consequently, the predescessor to modern constrained environment control strategies.

The concept of stiffness control may be clearly illustrated by considering the force control problem for a single degree of freedom manipulator as shown within Figure 2.2.

In this example, the manipulator is required to establish a contact force of f_d with a rigid environment. Assuming the presence of a simple PD control algorithm and steady-state conditions, the control force τ, which is equal to the contact force, will depend only upon the position error. However, the high environmental stiffness will prevent the control force from producing any significant surface deformation. As a result, the position error will persist, and the steady-state contact force will obey the following expression.

$$f_{ss} \cong k_p(x_d - x_e) \tag{2.1}$$

Note that the term k_p represents the controller position gain.

By considering Equation 2.1 and its similarity to *Hooke's Law*, the concept of stiffness control may be clearly summarized. By varying the proportional control gain k_p, which represents the effective stiffness of the closed-loop system, f_{ss} may be tuned to match any reasonable value of f_d. This statement may be generalized to include higher order manipulators, as the principles of operation remain unchanged.

Although the stiffness controller is important in that it represents the first successful attempt at robot manipulator force control, it is quite limited with respect to applicable environments. For instance, the controller is only capable of set-point force control, unless proportional control gains are altered in mid-operation, which is bad practice and cumbersome as a tool. In addition, the issue of simultaneous position/force control is not addressed by this method. In short, the stiffness controller is a pure force controller employing a joint space dynamic model. As such, its limitations rendered it inadequate for dealing with constrained environments.

2.2 Hybrid Position/Force Control

The ability to enforce simultaneous position/force control is an important requirement for any successful manipulator controller in a constrained environment. Consequently, research efforts following the development of the stiffness controller sought exclusively to attain this goal. The concept of *hybrid position/force control*, introduced by Raibert and Craig in 1981, *Raibert and Craig (1981)*, accomplished this end through task space decoupling. Raibert and Craig postulated that if the position and force control problems were decoupled, then independent, dedicated controllers could be designed for each subtask. These controllers would then act in parallel, with the net effect being a system under simultaneous position/force control.

In order to illustrate the operating principles of hybrid position/force control, consider the following dynamics formulation for an arbitrary n-link manipulator.

$$\underline{\tau} = M(\underline{q})\ddot{\underline{q}} + C(\underline{q}, \dot{\underline{q}})\dot{\underline{q}} + G(\underline{q}) + \underline{\tau}_{con}(\underline{f}) \tag{2.2}$$

Note that the vectors $\underline{\tau}$, $\underline{q}$ and $\underline{f}$ represent respectively, the joint torques, the joint displacements and the end-effector contact forces. Furthermore, the vector $\underline{\tau}_{con}$ represents contact forces at the end-effector transformed into equivalent disturbances for the joint degrees of freedom. Now, suppose it is known that a vector of desired joint trajectories $\ddot{\underline{q}}_d$, $\dot{\underline{q}}_d$ and $\underline{q}_d$ will produce the desired end-effector position and contact force histories. Then, a computed torque controller for the vector of joint torques $\underline{\tau}$ would take the following form.

$$\underline{\tau} = M(\underline{q})(\ddot{\underline{q}}_d + \underline{k}_v(\dot{\underline{q}}_d - \dot{\underline{q}}) + \underline{k}_p(\underline{q}_d - \underline{q})) + C(\underline{q}, \dot{\underline{q}})\dot{\underline{q}} + G(\underline{q}) + \underline{\tau}_{con} \tag{2.3}$$

Note again, that the control formulation of Equation 2.3 is with respect to the joint coordinate system $\underline{q}$.

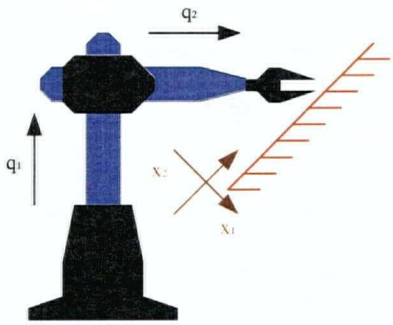

Figure 2.3: Hybrid Position/Force Control: 2DOF Manipulator

A hybrid position/force controller for the dynamics system of Equation 2.2 would differ from the controller of Equation 2.3 in that the hybrid controller would be based upon a decoupled task space coordinate system $\underline{x}$ as opposed to the joint coordinate system $\underline{q}$. A decoupled task space coordinate system is one where the basis vectors of the coordinate system are aligned in directions of pure position or force control. An example of a decoupled task space coordinate system for a two degree of freedom manipulator is shown within Figure 2.3.

In order to formulate a controller in terms of the decoupled task space, it is necessary to incorporate task space components into Equation 2.2 through a transforming function from task space to joint space.

$$\underline{x} = h(\underline{q}) \tag{2.4}$$

In addition, the first and second time derivatives of the task space basis vectors are also required.

$$\dot{\underline{x}} = J(\underline{q})\dot{\underline{q}} \qquad \ddot{\underline{x}} = J(\underline{q})\ddot{\underline{q}} + \dot{J}(\underline{q})\dot{\underline{q}} \tag{2.5}$$

Note that the term $J(\underline{q})$ is the manipulator Jacobian. Substitution of Equation 2.5 into

Equation 2.2 will yield the following expression.

$$\underline{\tau} = M(\underline{q})J^{-1}(\underline{q})(\ddot{\underline{x}} - \dot{J}(\underline{q})\dot{\underline{q}}) + C(\underline{q}, \dot{\underline{q}})\dot{\underline{q}} + G(\underline{q}) + \underline{\tau}_{con} \tag{2.6}$$

Note that Equation 2.6 may be used as the basis for another computed torque controller where the desired trajectories are now $\ddot{\underline{x}}_d$, $\dot{\underline{x}}_d$ and $\underline{x}_d$ as opposed to $\ddot{\underline{q}}_d$, $\dot{\underline{q}}_d$ and $\underline{q}_d$.

$$\begin{aligned} \underline{\tau} \;=\; & M(\underline{q})J^{-1}(\underline{q})(\ddot{\underline{x}}_d + \underline{k}_v(\dot{\underline{x}}_d - \dot{\underline{x}}) + \underline{k}_p(\underline{x}_d - \underline{x}) - \dot{J}(\underline{q})\dot{\underline{q}}) + \\ & C(\underline{q}, \dot{\underline{q}})\dot{\underline{q}} + G(\underline{q}) + \underline{\tau}_{con} \end{aligned} \tag{2.7}$$

Clearly, Equation 2.7 represents PD control with respect to decoupled position and force subtasks, since the vector $\underline{x}$ is a decoupled task space coordinate system. The remaining terms with respect to the joint space may be fed-forward from sensory data.

However, Equation 2.7 should not be confused with a constraint space dynamic model. The dynamic model of Equation 2.6 upon which the computed torque controller of Equation 2.7 is built, is actually a joint space dynamic model. Task space kinematic quantities are merely introduced into the joint space model so that a controller may receive task space references as input. This method operates by aligning the task space with constraint space conditions. The constraint space is not actually incorporated into the system dynamic model. As a result, complicated constraint trajectories will introduce difficulties for the method, as these constraints may no longer be aligned with a single, stationary task space coordinate system. The controller would require a position and time varying Jacobian function to relate the constantly changing task space system.

Nevertheless, hybrid position/force control successfully implements simultaneous position/force control for simple, constrained environments. Furthermore, by decoupling the control problem into purely position or force subtasks, the control of either is clearly isolated. Because of these qualities, hybrid position/force control, even today, remains a cornerstone for robot force control theory.

2.3 Hybrid Impedance Control

With the introduction of hybrid position/force control as a viable solution to control problems for constrained environments, the applicability of robot manipulators to industry increased significantly. As a result, subsequent work in the area sought to extend the method to its limits, focussing on the various applications of hybrid position/force control, as opposed to other, more effective alternatives. As a result, the next major advance in robot force control theory did not appear in the literature until 1987, *Hogan (1987)*, in the form of *hybrid impedance control* by Hogan.

Where hybrid position/force control was built upon distinct separations between the position and force subtasks, hybrid impedance control postulated that for any given task space direction, position and force were inseparable quantities. Instead of defining a task space direction as position or force exclusive, Hogan proposed that control for each task space direction should be derived on the basis of the desired dynamic behaviour for the components of position and force in that direction. Because the control parameters were relationships between position and force, Hogan's approach earned the name impedance control.

In practice, the hybrid impedance controller may often be implemented with a computed torque controller framework similar to that of the hybrid position/force controller as shown within Equation 2.7. Instead of using a PD controller represented by the terms $\underline{\ddot{x}}_d$, $\underline{\dot{x}}_d$ and $\underline{x}_d$, the desired task space response is derived from inverse Laplace transforms of the desired transfer function response for position and force in that particular direction. Hence, the control laws for position and force predominant directions are respectively, as follows.

$$\underline{\tau} = M(\underline{q})J^{-1}(\underline{q})(L^{-1}\{s(\dot{x}_d(s) - Z^{-1}(s)f(s))\} - \dot{J}(\underline{q})\dot{\underline{q}}) +$$

$$C(\underline{q},\dot{\underline{q}})\dot{\underline{q}} + G(\underline{q}) + \underline{\tau}_{con} \tag{2.8}$$

$$\begin{aligned} \underline{\tau} \;=\; & M(\underline{q})J^{-1}(\underline{q})(L^{-1}\{sZ^{-1}(s)(f_d(s) - f(s))\} - \dot{J}(\underline{q})\dot{\underline{q}}) + \\ & C(\underline{q},\dot{\underline{q}})\dot{\underline{q}} + G(\underline{q}) + \underline{\tau}_{con} \end{aligned} \tag{2.9}$$

The transfer functions in Equation 2.8 and Equation 2.9 are chosen by applying the *duality principle*, which assigns complementary transfer functions to the manipulator closed loop model depending upon the environmental impedance in that particular task space direction.

Nevertheless, hybrid impedance control is also based upon a joint space dynamic model. Like hybrid position/force control, it employs a task space transformation to incorporate constraint conditions into the system dynamics. Consequently, the method also suffers from the limitations of a stationary task space when applied to complicated constraint trajectories. However, the value of hybrid impedance control, and the reason for its inclusion in this discussion, is the landmark it represents for conceptual developments in the area of robot force control. Previously, little concern was given to the role of the environment in the determination of control laws for a given manipulator task. Hybrid impedance control represents the first environment sensitive control algorithm.

2.4 Reduced State Position/Force Control

Following the development of hybrid impedance control, researchers began to seriously consider the role of the environment in the formulation of constrained environment control strategies. Introduced to the literature in 1988, *reduced state position/force control* by McClamroch and Wang, *McClamroch and Wang (1988)*, was a novel approach to the position/force control problem, differing substantially from the joint space methodologies

of earlier work such as stiffness control.

McClamroch and Wang postulated that when a manipulator is brought into contact with a constraint surface, its degrees of positional freedom are reduced while degrees of force freedom are introduced into the system. Consequently, the design of a controller for a given manipulator and environment configuration should be based upon a reduced control coordinate set, representing the true degrees of freedom of the manipulator and constraint pairing. However, McClamroch and Wang did not identify a procedure for isolating this reduced control coordinate set. Their work proved the existence of reduced coordinates, and made reference to them in the formulation of the following computed torque controller.

Consider the previous n-link manipulator, and assume that a functional relationship exists between the manipulator joint space $\underline{q}$ and the reduced control coordinates $\underline{z}$.

$$\underline{q} = k(\underline{z}) \tag{2.10}$$

Furthermore, taking the first and second time derivatives of Equation 2.10 will yield the following expressions.

$$\dot{\underline{q}} = K(\underline{z})\dot{\underline{z}} \qquad \ddot{\underline{q}} = K(\underline{z})\ddot{\underline{z}} + \dot{K}(\underline{z})\dot{\underline{z}} \qquad K(\underline{z}) = \frac{dk(\underline{z})}{d\underline{z}} \tag{2.11}$$

In addition, the vector τ_{con} may be related to a vector of force degrees of freedom $\underline{\lambda}$, representing the contact forces between the manipulator and the environment.

$$\tau_{con} = J^{-1}(\underline{z})\underline{\lambda} \tag{2.12}$$

Note that the degrees of freedom for $\underline{z}$ combined with $\underline{\lambda}$ equal the original joint degrees of freedom. Substituting the results of Equation 2.11 and Equation 2.12 into the n-link manipulator dynamics will yield a reduced state dynamics formulation.

$$\underline{\tau} = M^*(K(\underline{z})\ddot{\underline{z}} + \dot{K}(\underline{z})\dot{\underline{z}}) + C^*(\underline{z}, \dot{\underline{z}})K(\underline{z})\dot{\underline{z}} + G^*(\underline{z}) + J^{-1}(\underline{z})\underline{\lambda} \tag{2.13}$$

Applying PD control terms, the equivalent computed torque controller may then be devised.

$$\begin{aligned} \underline{\tau} &= M^*(\underline{z})K(\underline{z})(\ddot{\underline{z}}_d + \underline{k}_v(\dot{\underline{z}}_d - \dot{\underline{z}}) + \underline{k}_p(\underline{z}_d - \underline{z})) + M^*(\underline{z})\dot{K}(\underline{z})\dot{\underline{z}} + \\ & C^*(\underline{z}, \dot{\underline{z}})K(\underline{z})\dot{\underline{z}} + G^*(\underline{z}) + J^{-1}(\underline{\lambda}_d + \underline{k}_f(\underline{\lambda}_d - \underline{\lambda})) \end{aligned} \tag{2.14}$$

Unlike the previous three controllers discussed, the reduced state position/force controller operates with respect to an optimal coordinate system $\underline{z}$, which fully recognizes the fact that the manipulator and the constraint are an integrated system with degrees of freedom which are different from the manipulator's alone. However, McClamroch and Wang have not identified a procedure for identifying Equation 2.10, which is the crux of the method, if it is to have any practical application. Despite this setback, reduced state position/force control is still a major achievement, for it represents the first fully documented attempt at building a constraint space dynamic model for manipulator control in a constrained environment.

2.5 Summary

This literature review has considered four different controllers in an attempt to chart the progression of concepts which ultimately lead to the author's contribution, hybrid constraint space control. The first method considered, stiffness control, was a joint space, set-point force controller, therefore, not actually a true constrained environment controller. Nevertheless, stiffness control was included in the discussion, as it represented the first attempt at force control. The second, hybrid position/force control, was yet another joint space formulation, although this method was capable of simultaneous position/force control. In addition, this hybrid method made use of a decoupling task space coordinate system in order to separate the position and force problems into independent

subtasks. However, the method employed stationary task space systems, making it inappropriate for complicated constraint trajectories. The third, hybrid impedance control, was also a joint space formulation, but was also the first method to recognize the influence of the environment in controller design. Finally, there was reduced state position/force control, which sought formulation of the system dynamics in an optimal reduced control coordinate system, markedly different from the previous joint space methods.

Hybrid constraint space control goes one step further than reduced state position/force control by actively defining the optimal control coordinates. In addition, this optimal control coordinate system is constraint based, as well as being decoupling in nature. Hence, hybrid constraint space control is actually the culmination of concepts from hybrid position/force control and reduced state position/force control, with the end result being a distinctively, constraint based approach to system modelling and control for constrained environments.

Chapter 3

Hybrid Constraint Space (HCS) Control

Hybrid constraint space modelling is a new approach in formulating control strategies for constrained robot manipulators. The technique essentially combines the concept of task space decoupling from hybrid position/force control with the concept of optimal state representation from reduced state position/force control, in order to introduce a unique constraint space coordinate system in which to model the system dynamics. There are two distinct advantages in employing hybrid constraint space control. First, the method completely decouples the position and force constraints into independent subtasks. Second, the control equations are explicitly expressed in terms of constraint parameters of practical concern, such as displacement along the constraint surface and normal interaction force between the manipulator end-effector and environmental constraint surface. As a result, the constrained system is decoupled and represented with an optimal, non-redundant model. Because of these qualities, hybrid constraint space control has the potential to become a truly viable alternative to existing methods for constrained manipulator control.

The formulation of a hybrid constraint space controller consists of eight distinct steps. This procedure is illustrated pictorally in Figure 3.4. In short, hybrid constraint space control is a modelling technique where the system dynamic model is constructed with respect to a constraint based coordinate system. Therefore, the first step is the definition of

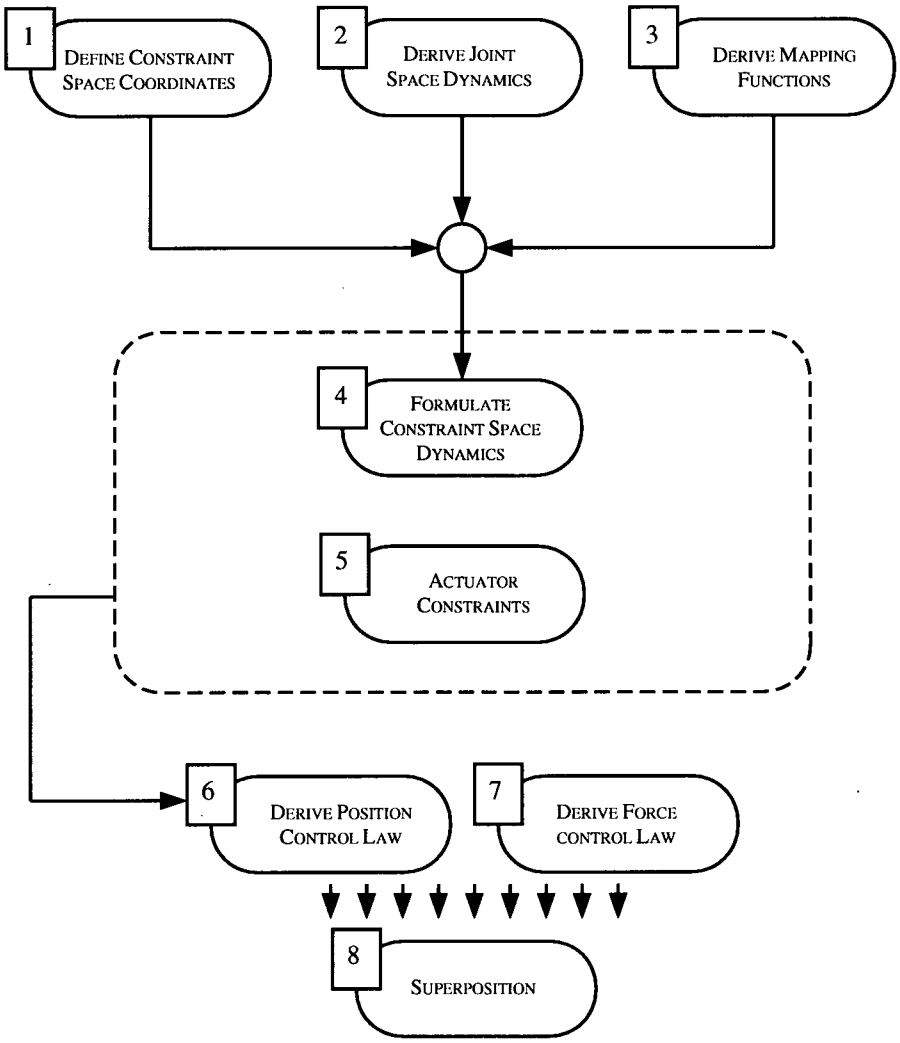

Figure 3.4: Procedural Flow Chart for a Hybrid Constraint Space Controller

such a coordinate system with respect to the constraint surface of the particular problem. Having defined a constraint space, the next immediate goal becomes the formulation of a system dynamic model based upon these coordinates.

In order to construct a constraint space dynamic model, it is first necessary to construct a manipulator joint space dynamic model as a starting point. Next, kinematic relationships are formulated between the constraint space degrees of freedom and the joint space degrees of freedom. These relationships or mapping functions are then used to transform the manipulator joint space dynamic model into an equivalent, but constraint space based dynamic model. The end-result is a manipulator dynamic model based upon the previously defined constraint space coordinate system. It is important to note that this constraint space dynamic model is a model where the manipulator end-effector implicitly travels along the constraint trajectory, provided that certain kinematic constraints and momentum compensating terms are applied to the manipulator joint torques, which is the fifth step.

Through the application of these first five steps, the manipulator end-effector has been constrained to a trajectory consistent with the environmental constraint surface. As a result, the position and force problems have become decoupled. Furthermore, the position and force control actuator requirements may be linearly superposed to achieve simultaneous position and force control. Therefore, the sixth and seventh steps involve the application of respectively, appropriate position and force control laws. Finally, the eighth step consists of the summation of the required manipulator actuator forces from the separate position and force subtasks into a unifying actuator force vector responsible for hybrid position and force control.

In order to clarify this procedure, each of these eight steps will now be discussed

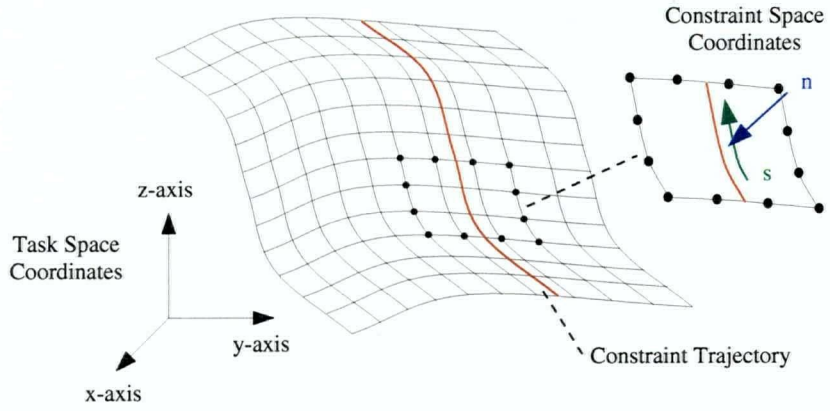

Figure 3.5: General Constraint Space Coordinate System

in detail with respect to a general n-degree of freedom manipulator constrained to an arbitrary surface in 3-space.

3.1 Definition of a Constraint Space Coordinate System

The first step in devising a hybrid constraint space controller is the definition of a constraint space coordinate system. This constraint space coordinate system is unique and consists of a single positional degree of freedom, s, along the constraint trajectory together with a single force degree of freedom, n, normal to the constraint trajectory. For the purposes of illustration, consider the constraint trajectory on the arbitrary curved constraint surface shown within Figure 3.5.

Consider now, the case where an arbitrary n-degree of freedom manipulator is constrained to the previously shown constraint surface, a situation illustrated in Figure 3.6. For all intents and purposes, the desired kinematics and dynamics of the constrained system may be completely described by s and n alone. This premise holds true, because for

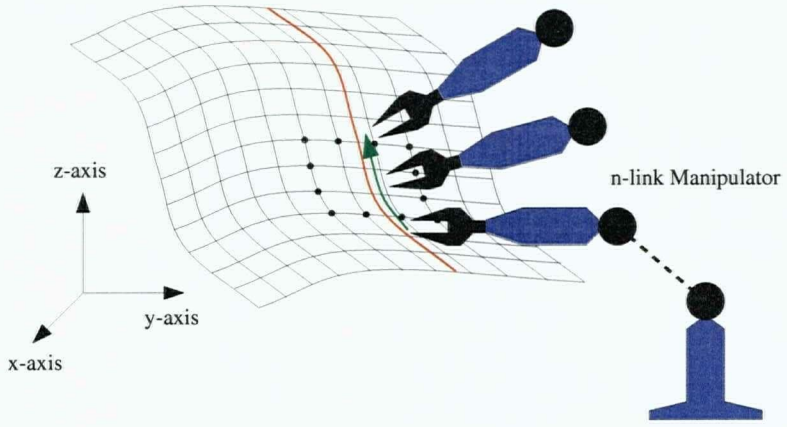

Figure 3.6: nDOF Manipulator Constrained to Arbitrary Surface

practical applications such as cutting or grinding, the control variables relate primarily to conditions on the workpiece. Since the workpiece surface is represented by the constraint surface, a process where the system consists of a manipulator constrained to a work surface may be completely controlled through the specification of the variables s and n. In other words, the coordinates of the constraint space coordinate system contain all the information required to specify interactive behaviour between the manipulator end-effector and the constraint surface. It is this property of the constraint space coordinate system which makes it an optimal reference frame for modelling the system dynamics.

3.2 Derivation of the Conventional Joint Space Dynamics

Having identified a constraint space coordinate system, it now becomes necessary to construct a model of the constrained manipulator dynamics with respect to these coordinates. The starting point in the construction of this model is the conventional joint space dynamics formulation of the manipulator. More specifically, the Lagrangian potential function and the equations of motion for the manipulator with respect to its joint

vector coordinates are required. Referring to the n-degree of freedom manipulator of Figure 3.6, the potential function and equations of motion take the following form.

$$L = L(\underline{q}, \underline{\dot{q}}) \tag{3.1}$$

$$\underline{Q} = M(\underline{q})\underline{\ddot{q}} + C(\underline{q}, \underline{\dot{q}})\underline{\dot{q}} + G(\underline{q}) \tag{3.2}$$

Once this potential function has been derived, it may be transformed into an equivalent constraint space representation through mapping functions, and subsequently used to produce the constraint space equations of motion through the application of Lagrange's equation. The joint space equations of motion on the other hand, will be used as the basis for kinematic constraints, to be discussed later.

3.3 Derivation of Mapping Functions

In order to transform the Lagrangian potential function into its constraint space equivalent, mapping functions between the joint space and the constraint space coordinate system are required. These mapping functions may be derived using differential kinematics, and take the following form.

$$\begin{array}{ccccc} \dot{q}_1 & = & f_1(s, \dot{s}) & = & \frac{dq_1}{ds}\dot{s} \\ \dot{q}_2 & = & f_2(s, \dot{s}) & = & \frac{dq_2}{ds}\dot{s} \\ \vdots & & \vdots & & \vdots \\ \dot{q}_n & = & f_n(s, \dot{s}) & = & \frac{dq_n}{ds}\dot{s} \end{array} \tag{3.3}$$

These results require an exact knowledge of the differential relation between a differential displacement along the constraint trajectory and subsequent differential displacements of the manipulator joints. For reasonable constraint trajectories which may be represented with analytic functions, these differential relations are easily derived.

However, in the case of highly nonlinear robots, differential kinematics alone are insufficient to produce the constraint space equations of motion. Geometric functions must also be derived.

$$\begin{aligned} q_1 &= g_1(s) \\ q_2 &= g_2(s) \\ \vdots &= \quad \vdots \\ q_n &= g_n(s) \end{aligned} \tag{3.4}$$

Nevertheless, it is not the intent of this work to deal with the cases of highly nonlinear robots. It is the intent of the work to illustrate the concept of hybrid constraint space control. Hence, it is sufficient to say at this point, that differential kinematics alone are adequate for deriving the mapping functions of example manipulators treated onwards. Nonlinear robots will complicate the derivation of the mapping functions, but will not limit the applicability of the method.

One further point of notice, is the case of redundancy. For a constraint trajectory in 2-space, only two degrees of manipulator freedom are required. Likewise, for a constraint trajectory in 3-space, only three degrees of manipulator freedom are required. As a result, the method assumes that only the bare minimum of the required joints are active. The remaining joints are locked, and removed from consideration. Therefore, the mathematical developments shown previously, should be restructured so that only active joint degrees of freedom are present. Note that these statements have been made ignoring orientation requirements.

3.4 Formulation of the Constraint Space Dynamics

Once the mapping functions have been identified, the constraint space dynamics formulation may be attained. The mapping functions are first used to transform the Lagrangian

potential function from joint space to constraint space coordinates.

$$L = L(\underline{q}, \underline{\dot{q}}) = L(s, \dot{s}) \tag{3.5}$$

The constraint space potential function may then be used through Lagrange's method to provide the constraint space dynamics.

$$Q_s = \frac{d}{dt}\left(\frac{\delta L}{\delta \dot{s}}\right) - \frac{\delta L}{\delta s} \tag{3.6}$$

$$Q_s = m_s(s)\ddot{s} + c_s(s, \dot{s})\dot{s} + g_s(s) \tag{3.7}$$

Note that the constraint space dynamics consist of a single equation of motion in s. This phenomena holds true, because for a constrained manipulator end-effector, the only two relevant parameters are the location of the end-effector along the constraint trajectory and the magnitude of the normal force of interaction between the end-effector and the constraint surface.

Since s represents motion along the constraint trajectory, the normal force involves no dynamics in the ideal case of a rigid environmental constraint. The normal force is simply a static force balance.

$$Q_n = N_r \tag{3.8}$$

Having formulated these two equations, which represent the constraint space dynamics, it is now necessary to define expressions which relate the generalized forces Q_s and Q_n to the joint actuator torque vector $\underline{Q}$. Again, $\underline{Q}$ in this context, refers to the active manipulator joints.

The relationship between Q_s and $\underline{Q}$ may be derived through the principle of virtual work, stated as follows.

$$Q_s = \frac{\delta W}{\delta s} \tag{3.9}$$

Given a differential displacement along s, δW is the dot product between the active joint torque vector and the differential displacements of the active joints.

$$\delta W = Q_1 \delta q_1 + Q_2 \delta q_2 + \cdots + Q_n \delta q_n \tag{3.10}$$

Since the differential relations between s and the active joint vectors are known, the following expression may be formed.

$$Q_s = F(Q_1, Q_2 \cdots Q_n) \tag{3.11}$$

However, this single equation is insufficient to provide a constraining solution to the active joint torque vector $\underline{Q}$. The results of the virtual work formulation only guarantee the magnitude of the equivalent force at the end-effector acting tangential to the constraint trajectory. Additional expressions which ensure kinematic consistency with the constraint trajectory are also required.

The relationship between Q_n and $\underline{Q}$ may be dealt with later, as hybrid constraint space control completely decouples the effects of Q_s and Q_n. Once kinematic constraints have been introduced to provide a set of actuator joint torques which ensure desired positional action, the required torques to satisfy normal force considerations may be included through linear superposition.

3.5 Actuator Constraints Required for Maintaining End-Effector Trajectory

Having constructed a formulation of the constraint space dynamics, it is useful to interpret the physical meaning of the generalized forces Q_s and Q_n. Consider Figure 3.7 where these two generalized forces are shown.

For a minimally active joint manipulator constrained to this surface, the joint motion will be unique, given that the joint servo motors are unpowered, and that a tangential

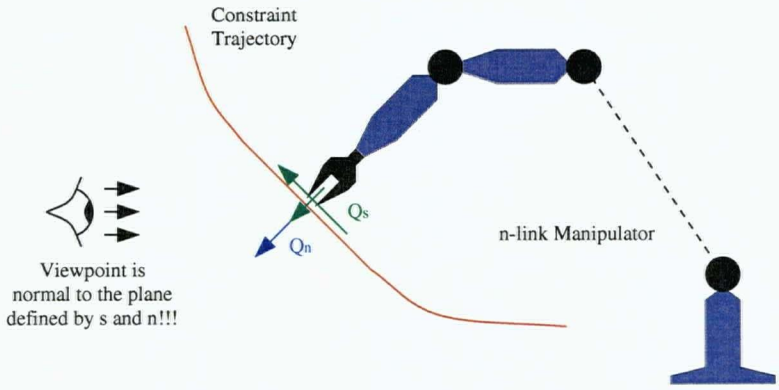

Figure 3.7: Generalized Constraint Forces Q_s and Q_n

force Q_s together with a normal force Q_n are applied to the manipulator end-effector. From a cursory inspection, it would appear that Q_s is responsible for controlling motion in the constraint trajectory direction, while Q_n is responsible for providing the desired normal force of interaction. This statement is mostly true.

The force Q_s is used to control position and the force Q_n is used to control normal force, but Q_n is also used to introduce kinematic constraints as well as momentum compensation. Consequently, it is now necessary to illustrate the derivation of the kinematic constraints which help define the position controlling component of Q_n. In order to define the kinematic constraints on the active joint vector, the existing differential kinematics must be differentiated once further with respect to time.

$$\begin{array}{rcccl} \ddot{q}_1 & = & \frac{d}{dt} f_1(\dot{s}) & = & h_1(\dot{s}, \ddot{s}) \\ \ddot{q}_2 & = & \frac{d}{dt} f_2(\dot{s}) & = & h_2(\dot{s}, \ddot{s}) \\ \vdots & & \vdots & & \vdots \\ \ddot{q}_n & = & \frac{d}{dt} f_n(\dot{s}) & = & h_n(\dot{s}, \ddot{s}) \end{array} \tag{3.12}$$

Note that these second time derivatives will have two components. The first component will be with respect to $\dot{s}$ while the second component will be with respect to $\ddot{s}$. The

first term is a coriolis and centripetal term, and is nonlinear. The second term deals with tangential acceleration, and is linear. Since the kinematic constraint is concerned with maintaining a desired tangential acceleration on the constraint surface, only the tangential components are used at this point.

$$\begin{aligned} \ddot{q}_1 &= h_1^*(\ddot{s}) \\ \ddot{q}_2 &= h_2^*(\ddot{s}) \\ \vdots & \quad \vdots \\ \ddot{q}_n &= h_n^*(\ddot{s}) \end{aligned} \tag{3.13}$$

Returning to the joint space equations of motion derived previously, these second time derivatives may be substituted. Given that the manipulator is linear and that only minimally active joints are present, the joint space equations of motion will consist of either two equations for a planar trajectory or three equations for a 3-space trajectory. Assuming a constraint trajectory in 3-space, the equations of motion will be linear and take the following form.

$$\begin{aligned} Q_1 &= F_1(m_1, m_2)\ddot{s} \\ Q_2 &= F_2(m_1, m_2)\ddot{s} \\ Q_3 &= F_3(m_1, m_2)\ddot{s} \end{aligned} \tag{3.14}$$

The terms m_1 and m_2 represent the local slopes to define the orientation of the tangent plane to the constraint surface at any given point. Note that these slopes are themselves, analytic functions dependent solely upon s. Regardless, these three equations of motion may be used to form a pair of force ratios, and together with the previous virtual work expression, used to solve a 3×3 system of equations to provide unique values for the active joint actuator forces.

$$\begin{aligned} Q_s &= F(Q_1, Q_2, Q_3) \\ \frac{Q_1}{Q_2} &= F_{12}(m_1, m_2) \end{aligned}$$

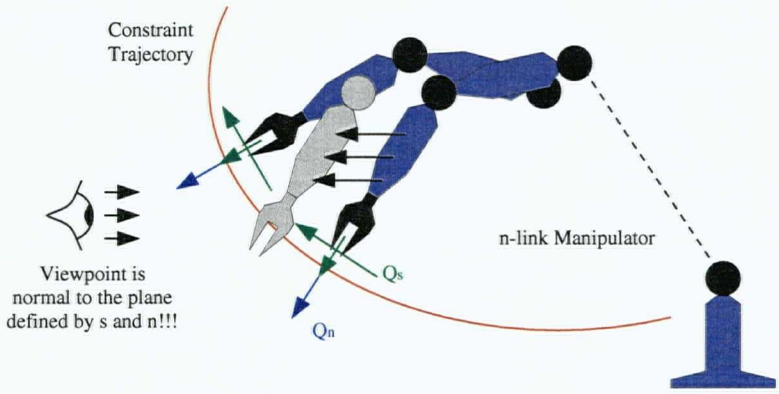

Figure 3.8: Momentum Compensation for Curved Surfaces

$$\frac{Q_2}{Q_3} = F_{23}(m_1, m_2) \tag{3.15}$$

With a unique solution for the active joint actuator forces already derived, it would seem that all the physical aspects of the constraint space method have been addressed. However, consider the case of a n-degree of freedom manipulator traversing a curved constraint surface, shown in detail within Figure 3.8.

It is clear that equations which focus on the desired end-effector tangential force along with a kinematic component of the normal force will always produce acceleration parallel to the constraint trajectory at any point. However, when rounding a curved path, the manipulator will have momentum which is normal to the constraint surface. If this momentum is not compensated, the end-effector will drift into the constraint surface. When drifting occurs, the kinematic and static force components of Q_n will have overlapping effects, with the consequence being a coupled position and force problem. In order to avoid this situation, momentum compensation must be included in the required values of the active joint torque actuators.

The computation of the required amounts of compensation are simple, and involve

employing the nonlinear terms of the second time derivatives computed for the kinematic constraints. These nonlinear terms may be substituted into the joint space equations of motion to directly give the required values of momentum compensation.

$$\begin{aligned} Q_{1mc} &= G_1(m_1, m_2)\dot{s} \\ Q_{2mc} &= G_2(m_1, m_2)\dot{s} \\ Q_{3mc} &= G_3(m_1, m_2)\dot{s} \end{aligned} \tag{3.16}$$

These terms may then be added to the previous solutions of Equation 3.15 to provide a set of active joint torque actuator forces which give exact end-effector position control.

$$\begin{aligned} Q_{1kin} &= Q_1 + Q_{1mc} \\ Q_{2kin} &= Q_2 + Q_{2mc} \\ Q_{3kin} &= Q_3 + Q_{3mc} \end{aligned} \tag{3.17}$$

Note that for non-curved surfaces, the momentum compensation terms are zero, and do not factor into the model.

3.6 Derivation of a Position Control Law

The discussion up until this point has assumed that a value for the generalized constraint space force Q_s has been available for the virtual work expression, which is then used to aid in the solution of the active joint torque forces. In order to actually generate an appropriate value of Q_s depending upon the status of the end-effector compared to its desired status upon the constraint trajectory, it is necessary to implement a position control law. Fortunately, the dynamics for the coordinate s consist of a single equation of motion. Because of this simplicity, which is a natural consequence of employing the decoupling action of the hybrid constraint space controller, any number of conventional control laws may be employed. The author has chosen to apply computed torque control,

Lewis, Abdallah, and Dawson (1993). Consequently, the controller for Q_s may be formed as follows.

$$Q_s = m_s(s)(\ddot{s}_d + k_{pv}(\dot{s}_d - \dot{s}) + k_{pp}(s_d - s)) + c_s(s, \dot{s})\dot{s} + g_s(s) \tag{3.18}$$

This controller is essentially a PD law with respect to the linear portion of the dynamics and a feed-forward mechanism with respect to the nonlinear portion. Note that sensory equipment can readily provide the required states s and $\dot{s}$ which are exactly measured along the constraint trajectory. Other control laws may be used, as long as they provide satisfactory convergence. The aim of this section is simply to provide a functioning and consistent method for generating Q_s.

3.7 Derivation of a Force Control Law

As mentioned previously, the generalized force Q_n in actuality, consists of two components. The first component deals with kinematic constraints and momentum compensation, and has had its effects implicitly incorporated into the active joint torque forces. Because of this incorporation, the position and force control problems may be completely decoupled. As a result, there is no control required for the second component of the generalized force Q_n in the ideal case of a rigid environment. The second component may simply be set equal to the desired contact force.

$$Q_n = f_d \tag{3.19}$$

However, in the case of a non-rigid environment, albeit a very stiff environment, a compliant force control law of the following form may be applied.

$$Q_n = f_d - k_{fv}k_e\dot{\delta} + k_{fp}(f_d - f_m) \tag{3.20}$$

Note that k_e represents the environmental stiffness, $\dot{\delta}$ represents the rate of change of deformation and f_m represents the desired normal force.

Regardless of the choice for either Equation 3.19 or Equation 3.20 it is necessary to compute the required active joint torque forces which when superposed with the kinematic requirements, produces the desired normal force of interaction. These requirements may be found again, through the principles of virtual work.

$$\begin{aligned} Q_{1nor} &= H_1(Q_n) \\ Q_{2nor} &= H_2(Q_n) \\ Q_{3nor} &= H_3(Q_n) \end{aligned} \tag{3.21}$$

Another method is to simply treat the active joint actuators as a coordinate space, and to find the representation of the normal force vector with respect to this joint space.

3.8 Superposition

With the position and normal components completely isolated, the final step for implementation consists of the linear superposition of terms.

$$\begin{aligned} Q_{1act} &= Q_{1kin} + Q_{1nor} \\ Q_{2act} &= Q_{2kin} + Q_{2nor} \\ Q_{3act} &= Q_{3kin} + Q_{3nor} \end{aligned} \tag{3.22}$$

Note that this development has not considered the effects of end-effector orientation. With various applications, the angle of approach is very important, and to address this concern, it is sufficient to say that angle of approach will only introduce additional kinematic constraints. Otherwise, the method of hybrid constraint space control is universally applicable.

Chapter 4

Derivation of an HCS Controller for a 2DOF Prismatic Robot

In order to illustrate the conceptual discussion from the previous chapter, consider the planar robot shown within Figure 4.9. This planar robot is shown with an arbitrary constraint surface, as the purpose of this section is the development of a hybrid constraint space controller for a general constraint.

4.1 Definition of a Constraint Space Coordinate System

The definition of a constraint space coordinate system for the arbitrary constraint shown within Figure 4.9 is very simple. The coordinate s is essentially the line integral along the constraint curve. The normal force coordinate n is essentially the local normal direction to s for any given value of the position coordinate. It is interesting to note that this definition differs from hybrid position/force control in that the the decoupling coordinate system is a moving coordinate system. The position coordinate s is a displacement along the constraint surface, not a displacement in some prespecified, fixed axis direction. In the same vein, the force coordinate n is the local normal to s, and as such, is also not in a prespecified, fixed axis direction. Furthermore, the method is different from reduced state position/force control in that an explicit and unique formulation for selecting the optimal system coordinates has been employed.

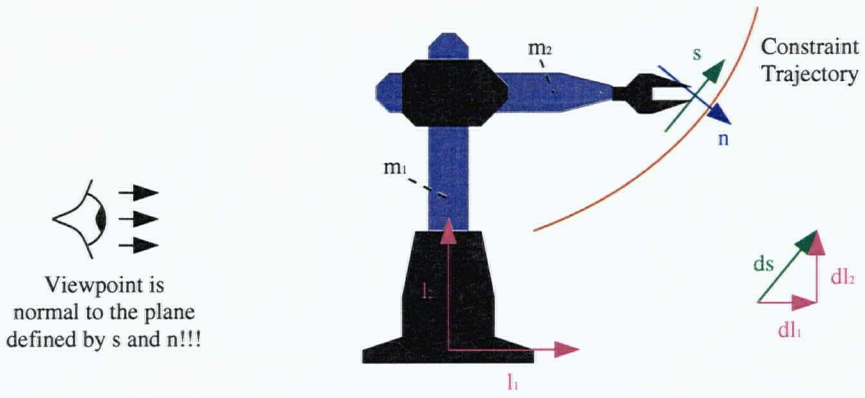

Figure 4.9: 2DOF Planar Robot with Arbitrary Constraint

4.2 Derivation of the Conventional Joint Space Dynamics

Having identified the constraint space coordinates s and n, it is now necessary to formulate the constraint space dynamics in terms of these two variables. The first step however, is the derivation of the joint space dynamics. With respect to the planar manipulator in question, the Lagrangian potential function may be written in terms of the kinetic energies of the two linkages.

$$L = KE_1 + KE_2 \tag{4.1}$$

$$KE_1 = \frac{1}{2}m_1\dot{l}_1^2 \qquad KE_2 = \frac{1}{2}m_2(\dot{l}_1^2 + \dot{l}_2^2) \tag{4.2}$$

$$L = \frac{1}{2}(m_1 + m_2)\dot{l}_1^2 + \frac{1}{2}m_2\dot{l}_2^2 \tag{4.3}$$

In addition, the joint space equations of motion should also be derived at this point, as they are required for the kinematic constraints.

$$\frac{d}{dt}\left(\frac{\delta L}{\delta \dot{\underline{q}}}\right) - \frac{\delta L}{\delta \underline{q}} = \underline{Q} \tag{4.4}$$

$$\underline{q} = \begin{bmatrix} l_1 \\ l_2 \end{bmatrix} \qquad \underline{Q} = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} \tag{4.5}$$

$$\begin{aligned} F_1 &= (m_1 + m_2)\ddot{l}_1 \\ F_2 &= m_2\ddot{l}_2 \end{aligned} \tag{4.6}$$

Note that these results have been derived ignoring the effects of gravity. Furthermore, Equation 4.3 and Equation 4.6 correspond to respectively Equation 3.1 and Equation 3.2 from the previous chapter.

4.3 Derivation of Mapping Functions

Having derived the joint space dynamics formulation, it is necessary to find mapping functions from joint space to constraint space. Differential kinematics may be employed to derive these relations. Consider the general differential displacements shown within Figure 4.9. With respect to this illustration, several expressions may be found.

$$\theta = tan^{-1}m \tag{4.7}$$

Note that the term m represents the local slope of the constraint surface. With respect to a planar trajectory, a single value of slope is all that is required to define the orientation of the constraint. Using this slope, differential relations between s and the joint vector may be found.

$$\frac{dl_1}{ds} = sin(tan^{-1}m) \qquad \frac{dl_2}{ds} = cos(tan^{-1}m) \tag{4.8}$$

Before the actual computation of the first time derivatives, recall the definition of the time derivative, being the sum of all partials, the partials consisting of derivatives taken with respect to all parameters which the function consists.

$$f = f(x_1, x_2 \cdots x_n) \tag{4.9}$$

$$\frac{df}{dt} = \frac{df}{dx_1}\frac{dx_1}{dt} + \frac{df}{dx_2}\frac{dx_2}{dt} + \cdots + \frac{df}{dx_n}\frac{dx_n}{dt} \tag{4.10}$$

Because the constraint space coordinate system uniquely defines the constrained system in an optimal fashion, for constrained motion, the joint vector must be a function of the constraint space coordinates. Note that there has been a state reduction as a consequence of constraining the end-effector to the constraint surface.

$$l_1 = l_1(s) \qquad l_2 = l_2(s) \tag{4.11}$$

As a result, it may be said that joint space representation for a constrained system is redundant.

Using these results, the differential kinematic relations may be defined as follows.

$$\begin{aligned} \dot{l}_1 &= \frac{dl_1}{dt} = \frac{dl_1}{ds}\frac{ds}{dt} = \sin(\tan^{-1} m)\dot{s} \\ \dot{l}_2 &= \frac{dl_2}{dt} = \frac{dl_2}{ds}\frac{ds}{dt} = \cos(\tan^{-1} m)\dot{s} \end{aligned} \tag{4.12}$$

The mapping functions for this manipulator and constraint system are then easily summarized.

$$\begin{aligned} \dot{l}_1 &= \sin(\tan^{-1} m)\dot{s} \\ \dot{l}_2 &= \cos(\tan^{-1} m)\dot{s} \end{aligned} \tag{4.13}$$

Note that Equation 4.13 corresponds to Equation 3.3. Furthermore, there is no equivalent to Equation 3.4 because this prismatic robot is linear.

4.4 Formulation of the Constraint Space Dynamics

Having derived the mapping functions, it is now possible to construct the constraint space dynamics through substitution. Recall the joint space Lagrangian potential function.

$$\begin{aligned} L &= \frac{1}{2}(m_1 + m_2)\dot{l}_1^2 + \frac{1}{2}m_2\dot{l}_2^2 \\ &= \frac{1}{2}(m_1 + m_2)\sin^2(\tan^{-1} m)\dot{s}^2 + \frac{1}{2}m_2\cos^2(\tan^{-1} m)\dot{s}^2 \end{aligned} \tag{4.14}$$

$$L = \frac{1}{2}(m_1 \sin^2(\tan^{-1} m) + m_2)\dot{s}^2 \tag{4.15}$$

The constraint space Lagrangian has now been formed, and the constraint space equation of motion may be found by applying Lagrange's equation with respect to the constraint degree of freedom s.

$$Q_s = \frac{d}{dt}\left(\frac{\delta L}{\delta \dot{s}}\right) - \frac{\delta L}{\delta s} \tag{4.16}$$

For clarity, some intermediate computations are now shown.

$$\frac{\delta L}{\delta \dot{s}} = (m_1 \sin^2(\tan^{-1} m) + m_2)\dot{s} \tag{4.17}$$

$$\frac{d}{dt}\left(\frac{\delta L}{\delta \dot{s}}\right) = (m_1 \sin^2(\tan^{-1} m) + m_2)\ddot{s} + 2m_1 \sin(\tan^{-1} m)\cos(\tan^{-1} m)\frac{1}{1+m^2}\dot{m}\dot{s} \tag{4.18}$$

$$\dot{m} = \frac{dm}{ds}\dot{s} \tag{4.19}$$

$$\frac{\delta L}{\delta s} = m_1 \sin(\tan^{-1} m)\cos(\tan^{-1} m)\frac{1}{1+m^2}\dot{m}\dot{s} \tag{4.20}$$

Summarizing these results produces the constraint space equation of motion.

$$Q_s = (m_1 \sin^2(\tan^{-1} m) + m_2)\ddot{s} + m_1 \sin(\tan^{-1} m)\cos(\tan^{-1} m)\frac{1}{1+m^2}\dot{m}\dot{s} \tag{4.21}$$

Note that Equation 4.21 corresponds to Equation 3.7. This constraint space equation of motion may now be used as the basis for a control algorithm governing the kinematic behaviour of the manipulator end-effector as it traverses the constraint surface.

However, it must be possible to define Q_s as a function of the joint actuator forces, or in other words, a relation must be found between F_1 and F_2 which ensures that a generalized force of Q_s is present in the direction of s. This relation may be derived through the principles of virtual work.

$$Q_s = \frac{\delta W}{\delta s} \tag{4.22}$$

$$\begin{aligned} Q_s &= \frac{F_1 \delta l_1 + F_2 \delta l_2}{\delta s} \\ &= \frac{F_1 \sin(\tan^{-1} m)\delta s + F_2 \cos(\tan^{-1} m)\delta s}{\delta s} \\ Q_s &= F_1 \sin(\tan^{-1} m) + F_2 \cos(\tan^{-1} m) \end{aligned} \tag{4.23}$$

Equation 4.23 is the analog to Equation 3.11 from the previous chapter. Note that this relationship does not uniquely define F_1 and F_2, since the relation only specifies what F_1 and F_2 must be as a pair to ensure an end-effector equivalent force of Q_s in the direction of s.

4.5 Actuator Constraints Required for Maintaining End-Effector Trajectory

Since the relationship derived from virtual work for the constraint force Q_s does not uniquely define F_1 and F_2, it becomes necessary to provide another relationship. Recall the previous unconstrained joint space dynamics.

$$\begin{aligned} F_1 &= (m_1 + m_2)\ddot{l}_1 \\ F_2 &= m_2\ddot{l}_2 \end{aligned} \tag{4.24}$$

If the second time derivatives were computed with respect to the constraint space coordinate time derivatives, and substituted into the equations of motion, a second kinematic constraining equation may be formed.

$$\ddot{l}_1 = \ddot{l}_1(\dot{s}, \ddot{s}) = \cos(\tan^{-1} m)\frac{1}{1+m^2}\dot{m}\dot{s} + \sin(\tan^{-1} m)\ddot{s} \tag{4.25}$$

$$\ddot{l}_2 = \ddot{l}_2(\dot{s}, \ddot{s}) = -\sin(\tan^{-1} m)\frac{1}{1+m^2}\dot{m}\dot{s} + \cos(\tan^{-1} m)\ddot{s} \tag{4.26}$$

Note that Equation 4.25 and Equation 4.26 are equivalent to Equation 3.12. Using knowledge from particle differential kinematics, it is known that the first terms are normal accelerations while the latter are tangential accelerations. Considering solely the tangential accelerations the second time derivative relations are as follows.

$$\ddot{l}_1 = \sin(\tan^{-1} m)\ddot{s} \tag{4.27}$$

$$\ddot{l}_2 = \cos(\tan^{-1} m)\ddot{s} \tag{4.28}$$

These equations may then be substituted into the joint space dynamics, and divided to form a new relation.

$$\frac{F_1}{F_2} = \frac{(m_1 + m_2)\sin(\tan^{-1} m)\ddot{s}}{m_2 \cos(\tan^{-1} m)\ddot{s}} = \frac{(m_1 + m_2)m}{m_2} \tag{4.29}$$

Now, assuming the presence of some form of controller providing values for Q_s, the following system of equations may be solved for F_1 and F_2, the required joint actuator forces for desired motion along the constraint surface.

$$Q_s = F_1 \sin(\tan^{-1} m) + F_2 \cos(\tan^{-1} m) \tag{4.30}$$

$$\frac{F_1}{F_2} = \frac{(m_1 + m_2)m}{m_2} \tag{4.31}$$

Note that the system of equations in the variables F_1 and F_2 shown by Equation 4.30 and Equation 4.31 are equivalent to the general expression from Equation 3.15.

Although a pair of equations have been derived for the solution of F_1 and F_2, these equations do not take into account the need for momentum compensation if the end-effector is swinging around a curved surface. Consider this description of the mechanics involved. The constraint space relationship derived from virtual work will ensure a certain tangential force Q_s in the direction of s. If the end-effector is lagging behind, Q_s may be increased, if is leading ahead, Q_s may be reduced. But, with the two actuators F_1 and F_2, there exist any infinite pair of compensation which can produce Q_s. Hence, a second constraint is required, that is the acceleration of the joints l_1 and l_2 must be such that the acceleration of the end-effector is in the direction of s. However, when the end-effector is tracing a curved surface, normal acceleration is also required.

In attempting to decouple the position and force problems completely, it is desirable to compute the required normal acceleration to maintain end-effector tracking, and compensate, so that the end-effector will follow the constraint surface exactly, without the

need for contact force to maintain tracking. The normal acceleration has already been computed previously when considering the kinematic constraints.

$$\ddot{l}_1 = \cos(\tan^{-1} m)\frac{1}{1+m^2}\dot{m}\dot{s} \tag{4.32}$$

$$\ddot{l}_2 = -\sin(\tan^{-1} m)\frac{1}{1+m^2}\dot{m}\dot{s} \tag{4.33}$$

Substituting these acceleration components into the joint space equations of motion will yield the following.

$$F_{1mc} = (m_1 + m_2)\cos(\tan^{-1} m)\frac{1}{1+m^2}\dot{m}\dot{s} \tag{4.34}$$

$$F_{2mc} = -m_2 \sin(\tan^{-1} m)\frac{1}{1+m^2}\dot{m}\dot{s} \tag{4.35}$$

Note that Equation 4.34 and Equation 4.35 are equivalent to Equation 3.16. Likewise, Equation 4.36 and Equation 4.37 are equivalent to Equation 3.17. These compensating forces may be added to the results of the systems solution to generate the required actuator forces for position control.

$$F_{1kin} = F_1 + F_{1mc} \tag{4.36}$$

$$F_{2kin} = F_2 + F_{2mc} \tag{4.37}$$

Note that for non-curved surfaces, $\dot{m} = 0$ and no compensation is required.

4.6 Derivation of a Position Control Law

The previous discussion has shown how the actuator forces may be computed, given the factors Q_s, s, $\dot{s}$, m and $\dot{m}$. The slope and slope time derivatives depend on the constraint geometry and are simple to compute as a function of end-effector location. The position and velocity along the constraint surface, may be measured. However, a control law is

required to compute the desired values of Q_s. For this task, it is necessary to return to the constraint space dynamics formulation derived earlier.

$$Q_s = (m_1 \sin^2(\tan^{-1} m) + m_2)\ddot{s} + m_1 \sin(\tan^{-1} m)\cos(\tan^{-1} m)\frac{1}{1+m^2}\dot{m}\dot{s} \qquad (4.38)$$

The computed torque controller can then be applied to this expression.

$$\begin{aligned} Q_s &= (m_1 sin^2(tan^{-1}m) + m_2)(\ddot{s}_d + k_{pv}(\dot{s}_d - \dot{s}) + k_{pp}(s_d - s)) + \\ &\quad (m_1 sin(tan^{-1}m)cos(tan^{-1}m)\frac{\dot{m}}{1+m^2})\dot{s} \qquad (4.39) \\ Q_s &= Q_s(s, \dot{s}, m, \dot{m}) \qquad (4.40) \end{aligned}$$

Note that Equation 4.39 is equivalent to Equation 3.18. Furthermore, the required system data shown in summary form by Equation 4.40 are easily attainable through sensors at the end-effector and an a priori knowledge of the constraint surface. Hence, this controller, which uses PD control on the linear dynamics and feed-forward control on the non-linear dynamics, is readily transferrable to most practical, constrained environments.

4.7 Derivation of a Force Control Law

As mentioned previously, the position control law in conjunction with the kinematic constraints and momentum compensation, has been formulated such that the end-effector exactly traces the constraint surface. The end-effector may not initially be at the correct point on the surface, but it will always remain on the constraint surface while it attempts to track the desired trajectory. Furthermore, this tracking has been achieved with kinematic constraints and momentum compensation, such that the end-effector contacts the constraint surface without any actual interaction force. Therefore, for force control, the desired force may simply be treated as a linear superposition to the position controlling

actuator forces. In the case of an ideal, rigid environment, no force control law is required. The joints may simply apply a desired contact force, which when superimposed with the position control components of the joint torques, produce an equal and opposite reaction from the environment, hence, the static force balance shown in Equation 3.19. However, when a non-rigid environment is encountered, a force control law is required, and will take exactly the form of Equation 3.20.

$$Q_n = f_d - k_{fv} k_e \dot{\delta} + k_{fp}(f_d - f_m) \tag{4.41}$$

Equation 4.41 does not differ from Equation 3.20 because these equations do not contain dynamic terms which are manipulator dependent, such as Equation 4.21.

The relationship between the actuator forces F_1 and F_2, and the desired contact force may be derived through virtual work.

$$Q_n = \frac{\delta W_n}{\delta n} \tag{4.42}$$

$$Q_n = \frac{F_1 \delta l_1 + F_2 \delta l_2}{\delta n} \tag{4.43}$$

$$\delta l_1 = -\delta n \cos(\tan^{-1} m) \qquad \delta l_2 = \delta n \sin(\tan^{-1} m) \tag{4.44}$$

$$Q_n = -F_1 \cos(\tan^{-1} m) + F_2 \sin(\tan^{-1} m) \tag{4.45}$$

Combining this result with the virtual work expression for Q_s will yield the following system of equations.

$$Q_s = F_1 \sin(\tan^{-1} m) + F_2 \cos(\tan^{-1} m) \tag{4.46}$$

$$Q_n = -F_1 \cos(\tan^{-1} m) + F_2 \sin(\tan^{-1} m) \tag{4.47}$$

This system may be rewritten with the constraint space forces as the independent quantities.

$$F_1 = Q_s \sin(\tan^{-1} m) - Q_n \cos(\tan^{-1} m) \tag{4.48}$$

$$F_2 = Q_s \cos(\tan^{-1} m) + Q_n \sin(\tan^{-1} m) \tag{4.49}$$

The components of F_1 and F_2 responsible for the normal force of interaction may then be simply found.

$$F_{1nor} = -Q_n \cos(\tan^{-1} m) \tag{4.50}$$

$$F_{2nor} = Q_n \sin(\tan^{-1} m) \tag{4.51}$$

Note that Equation 4.50 and Equation 4.51 are equivalent to the general expression of Equation 3.21.

4.8 Superposition

Having computed both the kinematic and normal force controlling components of the actuator forces F_1 and F_2, these components may be linearly superposed, since the hybrid constraint space controller is a linearly decoupling mechanism.

$$F_{1sup} = F_{1kin} + F_{1nor} \tag{4.52}$$

$$F_{2sup} = F_{2kin} + F_{2nor} \tag{4.53}$$

Finally, note that Equation 4.52 and Equation 4.53 are equivalent to Equation 3.22.

Chapter 5

Simulation Results for a 2DOF Prismatic Robot with HCS Control

In the previous chapter, a hybrid constraint space controller was developed for a two degree of freedom prismatic robot. This controller will now be used to govern the same robot as it is constrained to three different environmental surfaces: a planar surface, a concave circular surface and a convex circular surface. The computer simulations used to generate results for this section are written with respect to *Matlab m-file* and *s-function* format.

5.1 Planar Constraint Surface

The first simulation environment to be considered is the simple planar constraint surface, where the constraint trajectory is a straight line, as shown within Figure 5.10. The physical parameters used for this simulation are summarized within Table 5.1. The term k_e represents the environmental stiffness, and is required since the manipulator end-effector is assumed to be making contact with the constraint surface. Furthermore, these simulations assume zero vector initial conditions.

The first simulation with respect to the planar constraint, is the case of pure position control. The results for this simulation are shown in Figure 5.11, Figure 5.12 and Figure 5.13. Figure 5.11 shows a superposition of the constraint surface in task space

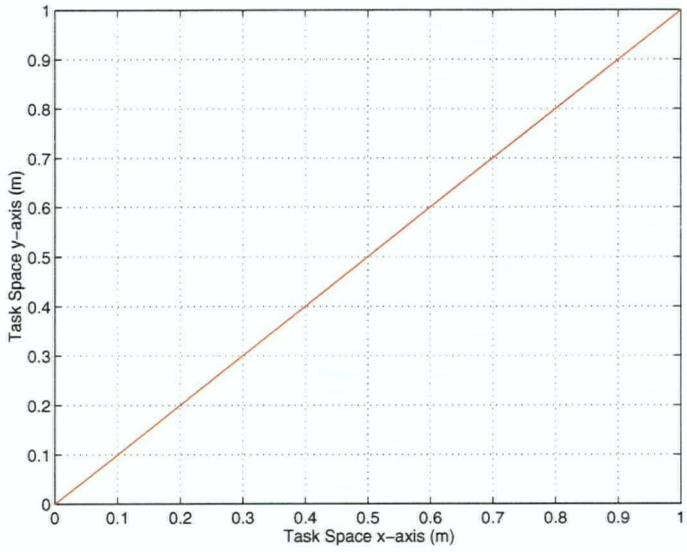

Figure 5.10: Planar Constraint Surface

m_1	m_2	k_e	k_{pv}	k_{pp}	k_{fv}	k_{fp}	$f(x)$
2 kg	1 kg	1000 N/m	5	10	0.075	0.001	$y = x$

Table 5.1: Simulation Parameters for the 2DOF Prismatic Robot with HCS Control Constrained to a Planar Surface

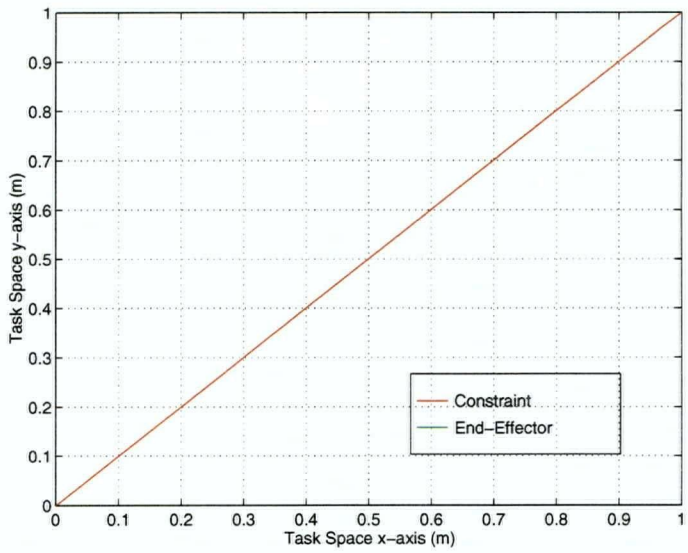

Figure 5.11: Task Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Planar Constraint Surface with Position Control

with the controlled end-effector position trajectory in task space. Clearly, the hybrid constraint space controller is capable of enforcing a trajectory that is consistent with the environmental constraint surface in question. However, maintaining a trajectory on the constraint surface alone is insufficient. The end-effector must also maintain the desired kinematic response on the constraint surface. The constraint space simulation results are shown within Figure 5.12 and demonstrate convergence between desired and actual constraint space position within two seconds. Finally, the hybrid constraint space controller must be capable of enforcing this position control without violating the boundaries of the environmental constraint surface. This condition is necessary because hybrid constraint space control is a position/force decoupling controller, such that the end-effector trajectory is maintained without the need for contact forces from the environment. Figure 5.13 confirms this condition by demonstrating that the contact force is zero for the duration of this first simulation.

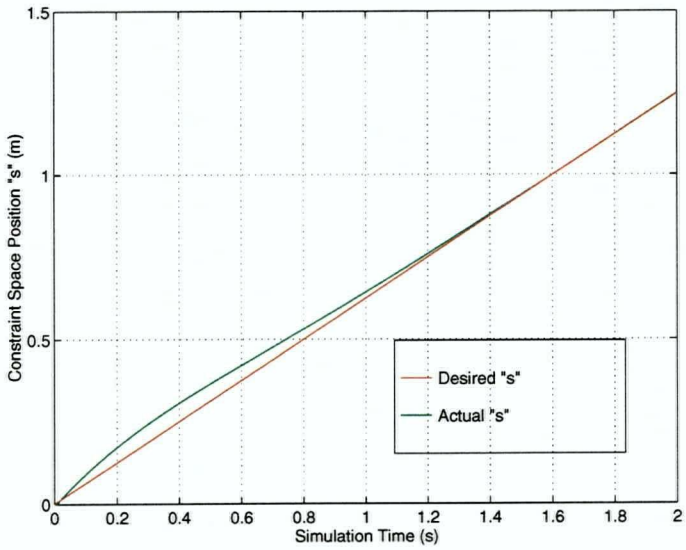

Figure 5.12: Constraint Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Planar Constraint Surface with Position Control

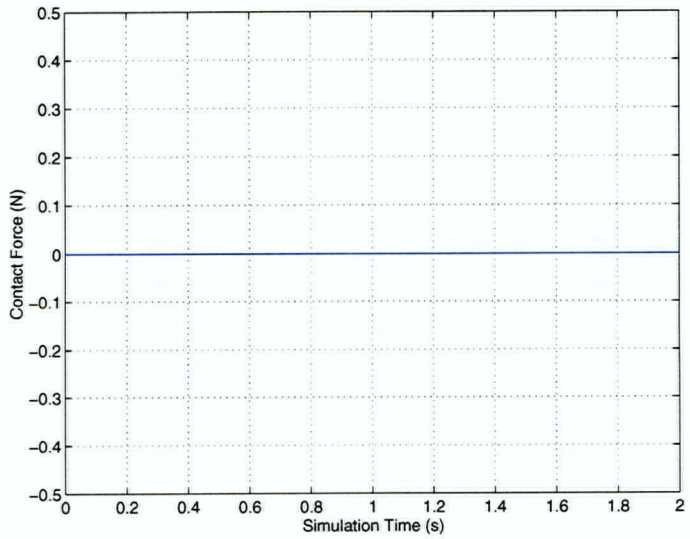

Figure 5.13: Contact Force Simulation Results for the 2DOF Prismatic Robot Constrained to a Planar Constraint Surface with Position Control

Having confirmed the effectiveness of the hybrid constraint space controller for the case of pure position control, consider now the case of position/force control for the planar constraint surface. The results for this simulation are shown in Figure 5.14, Figure 5.15 and Figure 5.16. Figure 5.14 shows a superposition of the constraint surface with the controlled end-effector trajectory in task space, and again, the two are consistent. The environment has been modelled as a hard surface with high stiffness, so that no significant deformation occurs as the result of commanding a contact force between the end-effector and the constraint surface. Figure 5.15 confirms the decoupling nature of the hybrid constraint space controller as the constraint space kinematic response is unchanged from the case of pure position control. Finally, the contact force is shown within Figure 5.16, illustrating convergent behaviour towards the desired value, well within the simulation time of two seconds.

The simulation results for the planar constraint show that the hybrid constraint space controller performs very well, providing decoupled and convergent control for both position and force trajectories on the constraint surface. However, the flexibility and usefulness of the hybrid constraint space controller are much more evident when the technique is applied to curved constraint surfaces.

5.2 Concave Circular Constraint Surface

The second simulation environment to be considered in this chapter is the concave circular constraint surface. Again, the constraint trajectory is a line, but a circular one, as shown within Figure 5.17. The physical parameters used for this simulation are summarized within Table 5.2. Once more, zero initial conditions are assumed. In addition, the hybrid constraint space controller for the simulations in this and the following section have made

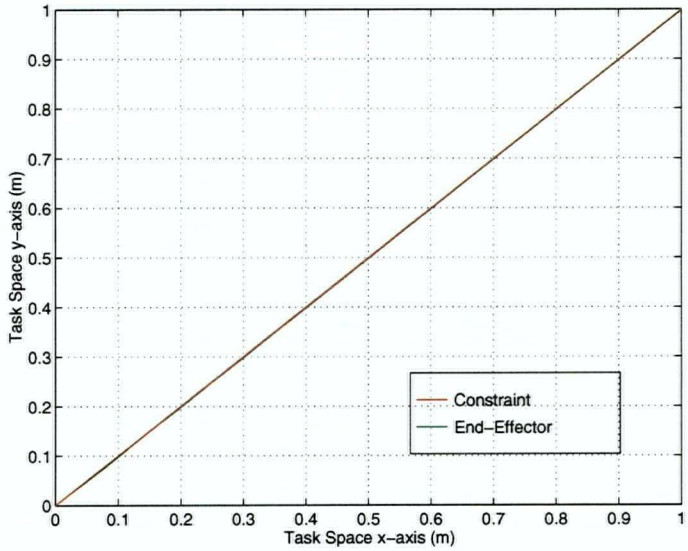

Figure 5.14: Task Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Planar Constraint Surface with Position/Force Control

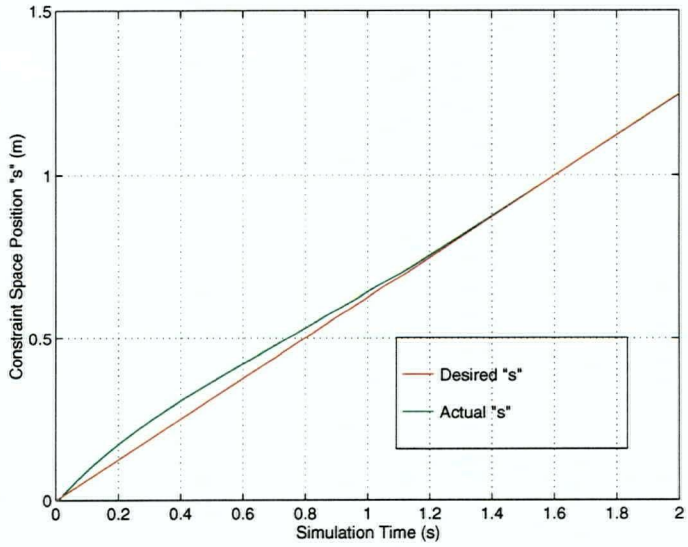

Figure 5.15: Constraint Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Planar Constraint Surface with Position/Force Control

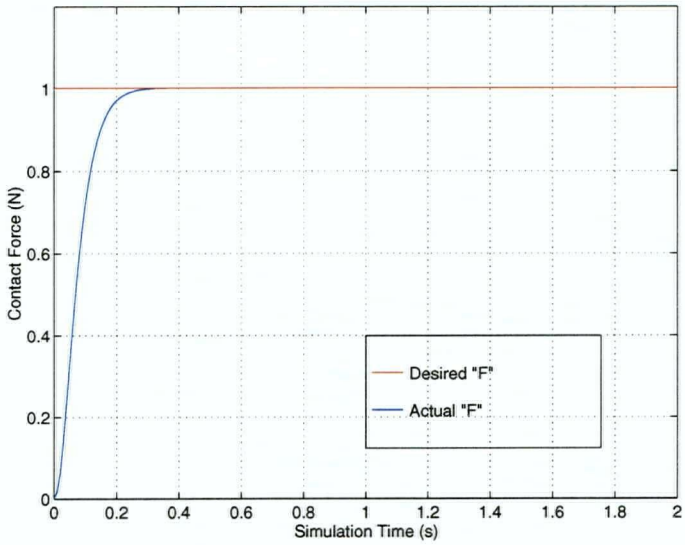

Figure 5.16: Contact Force Simulation Results for the 2DOF Prismatic Robot Constrained to a Planar Constraint Surface with Position/Force Control

use of momentum compensation, since the constraint surfaces exhibit curvature.

The case of pure position control for the circular concave constraint is considered first, with the simulation results shown within Figure 5.18, Figure 5.19 and Figure 5.20. Figure 5.18 shows the superposition of the constraint surface with the task space end-effector trajectory. Similar to the previous planar constraint case, the hybrid constraint space controller is capable of maintaining the end-effector on the constraint surface. Note though, that this simulation is conducted away from the singular region of infinite slope where the task space x-axis goes to one. In practice, a hybrid constraint space controller may be used to trace an entire circle, but the controlling equations must be restructured to avoid singularities associated with infinite slope. In the case of this two degree of freedom prismatic robot, the restructuring may be performed by switching the joint degrees of freedom with one another, such that a near infinite slope becomes a near zero slope. Figure 5.19 shows the kinematic response in constraint space coordinates,

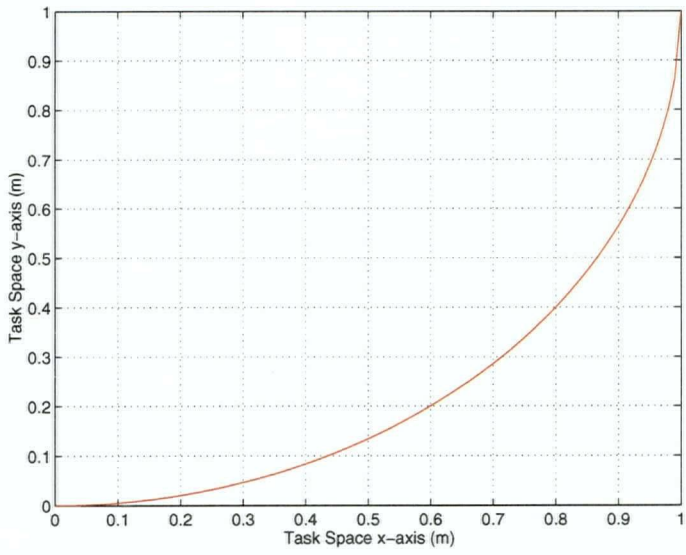

Figure 5.17: Concave Circular Constraint Surface

and again, the hybrid constraint space controller enforces convergence with the desired trajectory. With respect to the decoupling issue, Figure 5.20 shows a zero contact force for the duration of this pure position control simulation.

Consider now a second simulation with the concave circular constraint surface, where position/force control is required. The results for this trial are shown within Figure 5.21, Figure 5.22 and Figure 5.23. Figure 5.21 is a superposition of the constraint and the controlled end-effector trajectory, and shows a complete correlation between the two. Any deformation into the environment surface due to an applied normal force is once more negligible, as the constraint surface is modelled with a high stiffness. Figure 5.22 shows a constraint space coordinate kinematic response which is convergent, and identical to the corresponding pure position control plot, demonstrating again, the decoupling nature of the hybrid constraint space controller. Finally, Figure 5.23 illustrates the contact force history for this simulation, which is very similar to the one for the planar constraint dealt

m_1	m_2	k_e	k_{pv}	k_{pp}	k_{fv}	k_{fp}	$f(x)$
2 kg	1 kg	1000 N/m	5	10	0.1	0.001	$y = 1 - \sqrt{1 - x^2}$

Table 5.2: Simulation Parameters for the 2DOF Prismatic Robot with HCS Control Constrained to a Concave Circular Surface

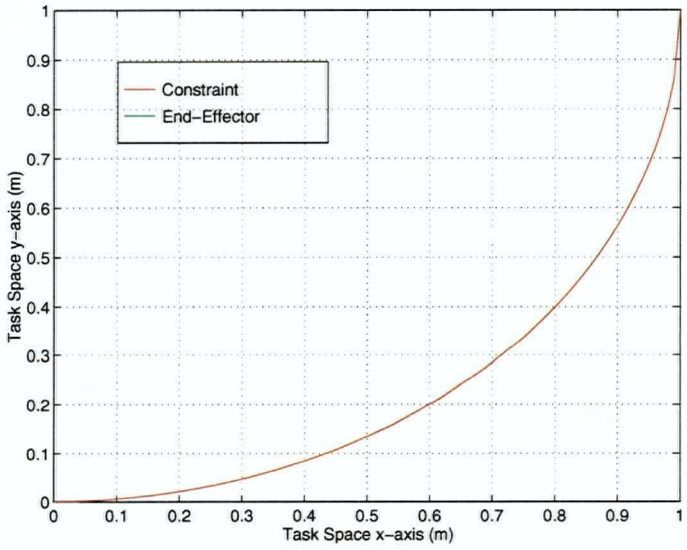

Figure 5.18: Task Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Concave Circular Constraint Surface with Position Control

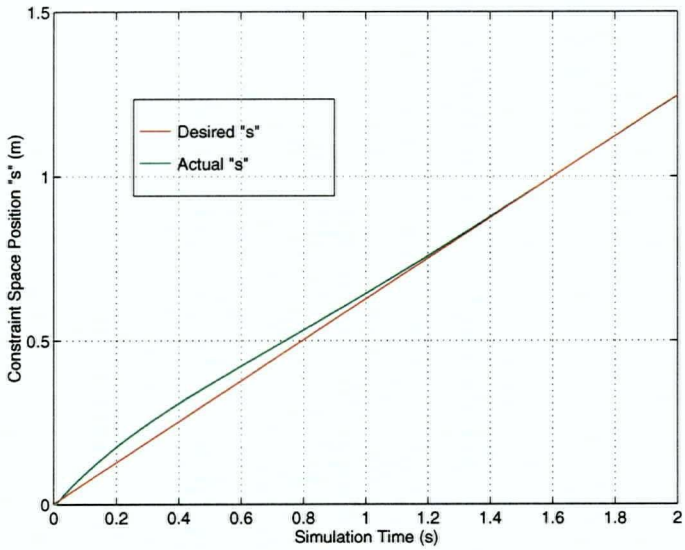

Figure 5.19: Constraint Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Concave Circular Constraint Surface with Position Control

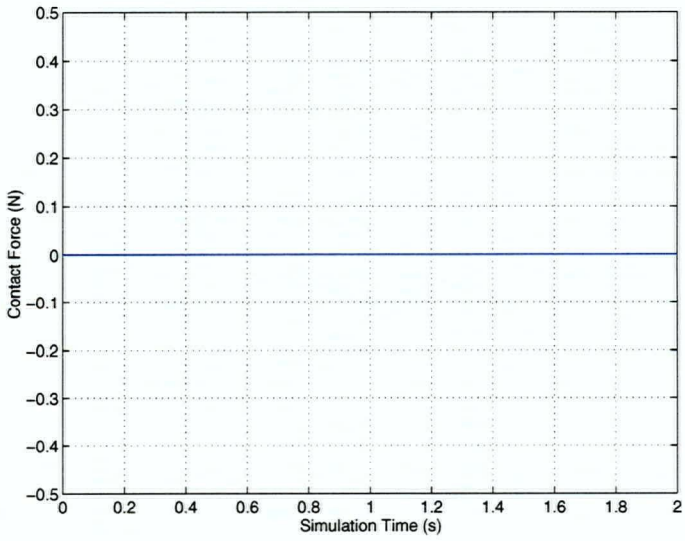

Figure 5.20: Contact Force Simulation Results for the 2DOF Prismatic Robot Constrained to a Concave Circular Constraint Surface with Position Control

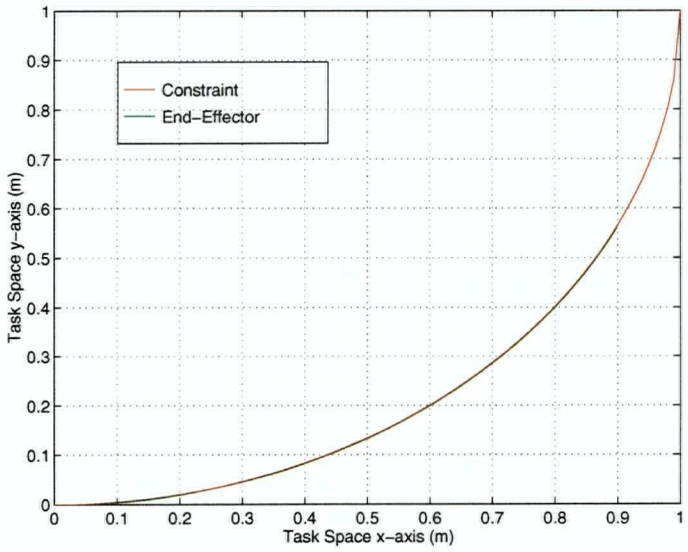

Figure 5.21: Task Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Concave Circular Constraint Surface with Position/Force Control

with previously. Therefore, for linear and curved surfaces, the hybrid constraint space controller is capable of decoupling the position and force problems, as well as providing convergent control for both.

5.3 Convex Circular Constraint Surface

The final simulation environment to be considered is the convex circular constraint surface. Like the previous two constraint trajectories already considered, this trajectory is also a planar line, as shown within Figure 5.24. The physical parameters used for this simulation are summarized within Table 5.3. As before, zero initial conditions are assumed.

The case of pure position control for the convex circular constraint is treated first, the simulation results being shown within Figure 5.25, Figure 5.26 and Figure 5.27. The

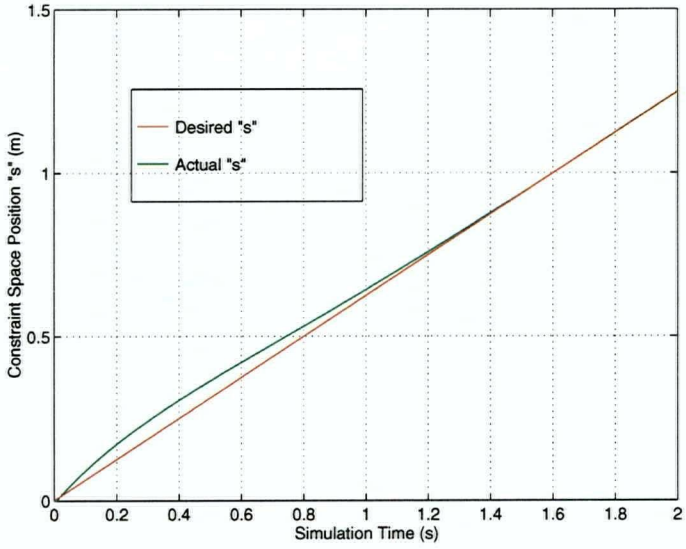

Figure 5.22: Constraint Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Concave Circular Constraint Surface with Position/Force Control

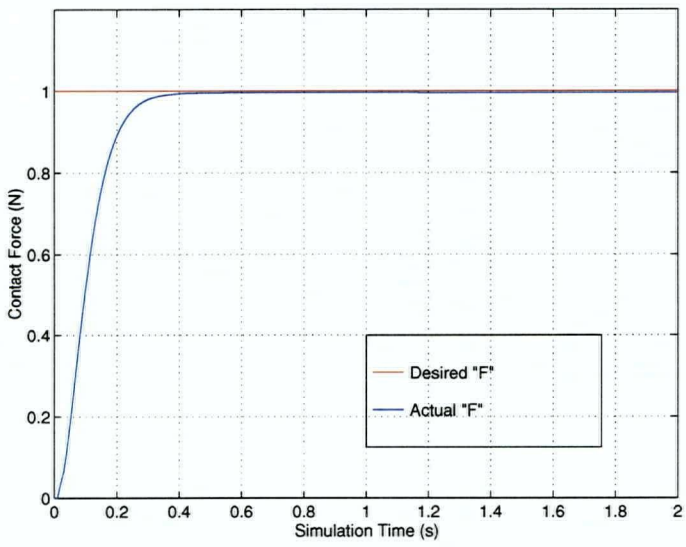

Figure 5.23: Contact Force Simulation Results for the 2DOF Prismatic Robot Constrained to a Concave Circular Constraint Surface with Position/Force Control

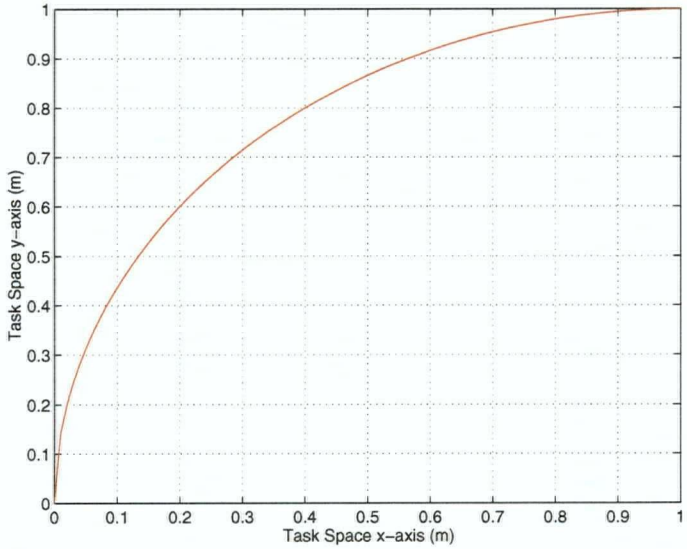

Figure 5.24: Convex Circular Constraint Surface

m_1	m_2	k_e	k_{pv}	k_{pp}	k_{fv}	k_{fp}	$f(x)$
2 kg	1 kg	1000 N/m	5	10	0.075	0.001	$y = \sqrt{1-(x-1)^2}$

Table 5.3: Simulation Parameters for the 2DOF Prismatic Robot with HCS Control Constrained to a Convex Circular Constraint Surface

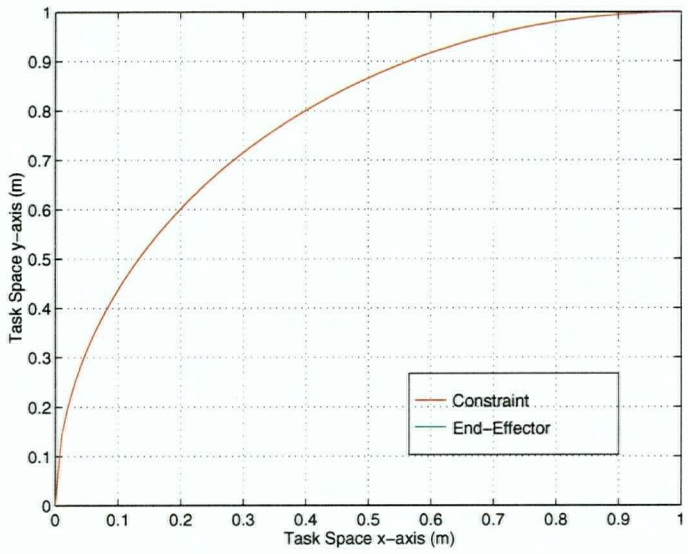

Figure 5.25: Task Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Convex Circular Constraint Surface with Position Control

superposition of the constraint surface with the end-effector path is shown by Figure 5.25, which demonstrates an almost exact correlation between the two. Similar to the previous case, the region of singular slope near the origin is avoided within this simulation. If the entire circular path is required, the singular regions may be dealt with as mentioned previously for the case of the concave circular constraint. Figure 5.26 shows the kinematic response in constraint space coordinates, and again, there is convergence between the desired history and the actual history. Figure 5.27 demonstrates once more, that the hybrid constraint space controller is decoupling, since for this pure position control problem, the contact force history is again zero.

Consider now the final simulation in this chapter, where the prismatic robot is simulated for position/force control while restricted to the convex circular constraint surface. The results of this run are shown within Figure 5.28, Figure 5.29 and Figure 5.30, and like the previous, demonstrate the effectiveness of the hybrid constraint space controller.

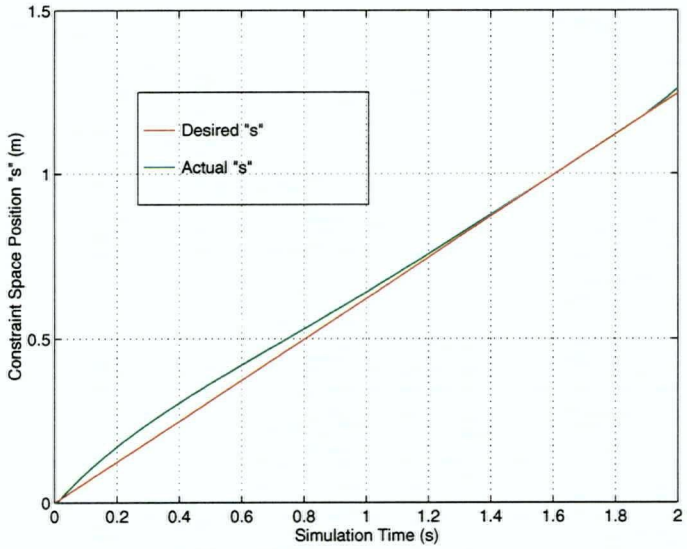

Figure 5.26: Constraint Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Convex Circular Constraint Surface with Position Control

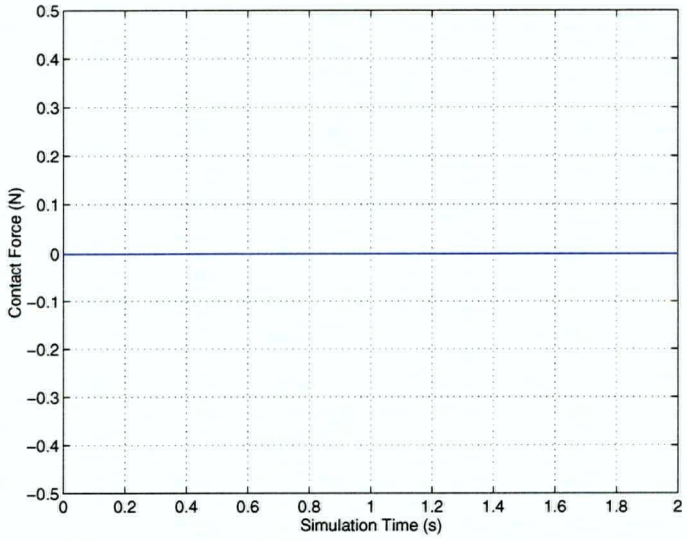

Figure 5.27: Contact Force Simulation Results for the 2DOF Prismatic Robot Constrained to a Convex Circular Constraint Surface with Position Control

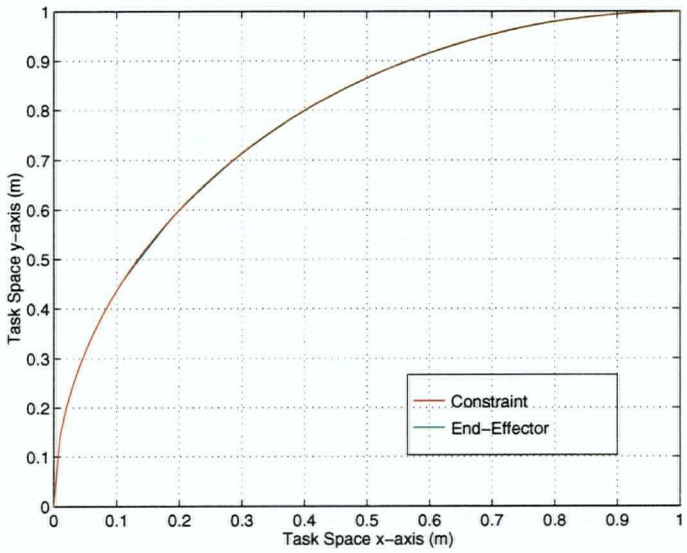

Figure 5.28: Task Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Convex Circular Constraint Surface with Position/Force Control

Figure 5.28 shows that the controlled end-effector perfectly traces the outline of the convex circular constraint surface. Furthermore, Figure 5.29 shows that the end-effector maintains a convergent kinematic response while tracing the surface. Finally, Figure 5.30 shows that the contact force is also convergent to the desired level.

Hence, for the planar, concave circular and convex circular constraint surfaces, the prismatic robot under hybrid constraint space control is completely decoupling with respect to the position and force control problems, as well as capable of providing convergent control to the desired position and force trajectories.

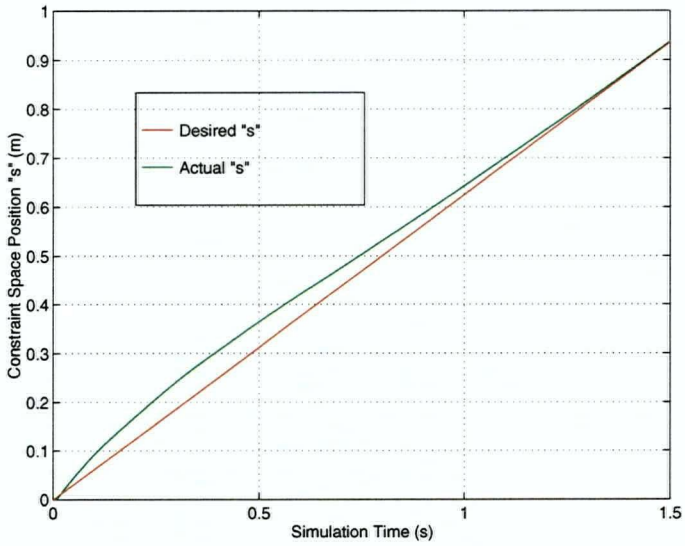

Figure 5.29: Constraint Space Simulation Results for the 2DOF Prismatic Robot Constrained to a Convex Circular Constraint Surface with Position/Force Control

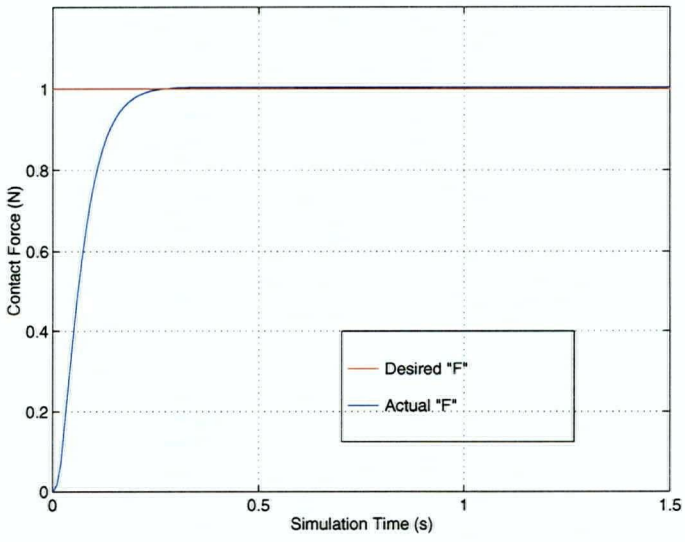

Figure 5.30: Contact Force Simulation Results for the 2DOF Prismatic Robot Constrained to a Convex Circular Constraint Surface with Position/Force Control

5.4 Comparative Discussion

Having performed various simulations for a two degree of freedom prismatic robot under hybrid constraint space control, it is evident that the method provides very good simultaneous position and force regulation in a constrained environment. Now, how does this performance compare to previous methods, for example, the earlier work discussed in Chapter 2? A qualitative assessment may be made as follows.

First, with respect to the stiffness controller, the hybrid constraint space controller is clearly better. The constraint space method gives simultaneous position and force control for arbitrary trajectories, while the stiffness controller is only useful for setpoint force control. On the other hand, the stiffness controller was included in the literature review more as a historical benchmark, than a performance benchmark.

A more fair comparison, would be the relative performance of a hybrid position/force controller to that of the constraint space controller. Like the constraint space method, hybrid position/force control is also capable of enforcing simultaneous position and force trajectories. However, the hybrid position/force controller was developed with respect to stationary task space coordinates. In other words, for a hybrid position/force controller to operate with a curved constraint surface, a time-varying Jacobian function relating the changing task space coordinates with the curvature of the environment would be required. Hybrid constraint space control avoids this problem by employing a constraint space dynamic model. By using differential kinematics, the constraint space dynamic model is capable of tracking along curving constraint trajectories without the need for establishing time-varying Jacobian transformations from task to joint space.

A comparison with hybrid impedance control yields similar results, as the impedance

controller is basically a more realistic version of the hybrid position/force controller which recognizes that position and force are not independent quantities for any specific direction in task space given the compliance of practical constraint environments. Nevertheless, position and force relations in the same task space degree of freedom is not the issue here. Like hybrid position/force control, the impedance controller employs stationary task space coordinates when transforming from task to joint space. As a result, it too requires time-varying Jacobian transformations.

Finally, it is noteworthy to compare the operating principles of the reduced state position/force controller with that of the hybrid constraint space controller. In principle, both methods are very similar, with the intent of formulating constrained system dynamics with respect to a set of optimal coordinates. However, the reduced state position/force controller does not actually detail a specific method for deriving these optimal coordinates. Hybrid constraint space control though, has an established method for deriving an optimal dynamics formulation, which may be used to successfully implement simultaneous position and force control, as shown by the simulation results from this chapter. Consequently, the author feels that hybrid constraint space control may offer a viable alternative to conventional means of position and force control for constrained environment control problems.

Chapter 6

Conclusions & Summary

6.1 Conclusions

The hybrid constraint space controller, as simulated for the two degree of freedom prismatic robot, performed very well. In every instance, the controller was capable of providing decoupled control of both position and force. Furthermore, the actual position and force trajectories consistently converged to the desired position and force trajectories.

6.2 Summary

This thesis has considered in full, the origins, conceptual detail and initial simulation of the hybrid constraint space control technique developed by the author. The literature review has charted the development of the method with respect to previously published work, citing strong conceptual ties with the task space decoupling approach of hybrid constraint space position/force control and the optimal state representation approach of reduced state position/force control. The method for constructing a hybrid constraint space controller with respect to a general manipulator and constraint pair were then discussed, with the emphasis placed on the physical concepts justifying each step in the technique. Subsequently, a two degree of freedom prismatic robot was used as an illustrative example for the construction of a hybrid constraint space controller. Finally,

simulation trials were performed for this specific hybrid constraint space controller under the restriction of three constraint surfaces, a planar, a concave circular and a convex circular environment. The simulation results for these trials were very promising.

6.3 Future Work

It is the author's intent to extend these results for nonlinear manipulator configurations in a future publication. Specific attention will be given to the case of the two degree of freedom rotary manipulator. Furthermore, the prismatic case will be completely generalized to include a three degree of freedom prismatic manipulator constrained to a constraint surface in 3-space.

Bibliography

[1] Arnold, V. I., (1978) *Mathematical Methods of Classical Mechanics.* New York: Springer-Verlag.

[2] Åström, Karl Johan, and Björn Wittenmark, (1989) *Adaptive Control.* Reading, Massachusetts: Addison-Wesley Publishing Company.

[3] Chiaverini, Stefano, and Lorenzo Sciavicco, (1993) "The Parallel Approach to Force/Position Control of Robotic Manipulators." *IEEE Transactions on Robotics and Automation*, Vol.9, No.4.

[4] Fu, Gonzalez, and Lee, (1987) *Robotics: Control, Sensing, Vision, and Intelligence.* New York: McGraw-Hill Book Company.

[5] Goldstein, Herbert, (1980) *Classical Mechanics: 2nd Edition.* Reading, Massachusetts: Addison-Wesley Publishing Company.

[6] Grabbe, Carroll, Dawson, and Qu, (1992) "Robust Control of Robot Manipulators During Constrained and Unconstrained Motion." *Proceedings of the 1992 IEEE International Conference on Robotics and Automation.*

[7] Greenwood, Donald T., (1965) *Principles of Dynamics.* Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

[8] Hogan, N., (1987) "Stable Execution of Contact Tasks using Impedance Control." *Proceedings of the 1987 IEEE International Conference on Robotics and Automation.*

[9] Houshangi, N., (1992) "Position/Force Control of a Robot Manipulator Over an Unknown Surface." *Proceedings of the 1992 IEEE International Conference on Systems, Man and Cybernetics.*

[10] Kankaanranta, R., and H. N. Koivo, (1986) "A Model for Constrained Motion of a Serial Link Manipulator." *Proceedings of the 1986 IEEE International Conference on Robotics and Automation.*

[11] Lewis, Abdallah, and Dawson, (1993) *Control of Robot Manipulators.* New York: Macmillan Publishing Company.

[12] Lipschutz, Seymour, (1964) *Theory & Problems of Set Theory & Related Topics.* New York: McGraw-Hill Book Company.

[13] Liu, Jing-Sin, (1991) "Hybrid Control for a Class of Constrained Mechanical Systems." *Proceedings of the 30th Conference on Decision and Control.*

[14] McClamroch, N. Harris, and Danwei Wang, (1988) "Feedback Stabilization and Tracking of Constrained Robots." *IEEE Transactions on Automatic Control*, Vol.33, No.5.

[15] Raibert, M., and J. Craig, (1981) "Hybrid Position/Force Control of Manipulators." *Journal of Dynamic Systems, Measurement, and Control*, Vol.102.

[16] Salisbury, J., and J. Craig, (1980) "Active Stiffness Control of Manipulator in Cartesian Coordinates." *Proceedings of the 19th IEEE Conference on Decision and Control.*

[17] Su, Chun-Yi, and Yury Stepanenko, (1994) "Robust Motion/Force Control of Mechanical Systems with Classical Nonholonomic Constraints." *IEEE Transactions on Automatic Control*, Vol.39, No.3.

[18] Takahashi, Rabins, and Auslander, (1970) *Control and Dynamic Systems.* Reading, Massachusetts: Addison-Wesley Publishing Company.

[19] Van de Vegte, John, (1990) *Feedback Control Systems: 2nd Edition.* Englewood Cliffs, New Jersey: Prentice Hall.

[20] Wang, Danwei, and N. Harris McClamroch, (1993) "Position and Force Control for Constrained Manipulator Motion: Lyapunov's Direct Method." *IEEE Transactions on Robotics and Automation*, Vol.9, No.3.

[21] Yun, Xiaoping, (1988) "Dynamic State Feedback Control of Constrained Robot Manipulators." *Proceedings of the 27th IEEE Conference on Decision and Control.*

Appendix A

Software Coding for Simulations Performed in Matlab

CONSTRAINT SURFACE #1

```
function[sys,x0]=surf1(t,x,u,flag,l1_dot,l1,l2_dot,l2)

if abs(flag) == 1

m1=2; m2=1; kpv=5; kpp=10; kfv=0.075; kfp=0.001;

sd_dd=u(1); sd_d=u(2); sd=u(3); Fd=u(4);

% Assuming a slope of m=1;
st=sin(atan(1)); ct=cos(atan(1));

% Determine equivalent point on surface.
% x(1) = l1_dot = y_dot
% x(2) = l1     = y
% x(3) = l2_dot = x_dot
% x(4) = l2     = x
xs=(x(4)+x(2))/2; xs_d=(x(3)+x(1))/2;
ys=xs;

% Use equivalent surface point to determine constraint position.
s=xs/ct; s_d=xs_d/ct;

% Determine position controller.
Qs=(m1*st^2+m2)*(sd_dd+kpv*(sd_d-s_d)+kpp*(sd-s));
F2=Qs/(((m1+m2)/m2)*1*st+ct);
F1=F2*((m1+m2)/m2)*1;

% Determine force controller.
ke=1000;
delta=sqrt((xs-x(4))^2+(ys-x(2))^2);
delta_d=-x(1)*ct+x(3)*st;
if ((x(4)>xs)&(x(2)<ys))
        global Fm;
        Fm=ke*delta;
else
        global Fm;
        Fm=0;
end
Qn=Fd+kfp*(Fd-Fm)-kfv*ke*delta_d;

% Superpose the position and force controllers.
F1=F1-Qn*ct;
F2=F2+Qn*st;
```

```
% Compute the manipulator dynamics.
        A=[ 0 0 0 0
            1 0 0 0
            0 0 0 0
            0 0 1 0 ];
        B=[ (F1+Fm*ct)/(m1+m2)
            0
            (F2-Fm*st)/m2
            0 ];
        sys=A*x+B;

elseif abs(flag) == 3

global Fm;
if t==0
        Fm=0;
end

sys=[x(1) x(2) x(3) x(4) Fm];

elseif abs(flag) == 0

sys=[4 0 5 4 0 1];
x0=zeros(4,1);
x0(1)=l1_dot;
x0(2)=l1;
x0(3)=l2_dot;
x0(4)=l2;

else

sys=[];

end
```

SIMULATION #1

```
function [ret,x0,str,ts,xts]=finalsim1(t,x,u,flag);
%FINALSIM1      is the M-file description of the SIMULINK system named FINALSIM1.
%       The block-diagram can be displayed by typing: FINALSIM1.
%
%       SYS=FINALSIM1(T,X,U,FLAG) returns depending on FLAG certain
%       system values given time point, T, current state vector, X,
%       and input vector, U.
%       FLAG is used to indicate the type of output to be returned in SYS.
%
%       Setting FLAG=1 causes FINALSIM1 to return state derivatives, FLAG=2
%       discrete states, FLAG=3 system outputs and FLAG=4 next sample
%       time. For more information and other options see SFUNC.
%
%       Calling FINALSIM1 with a FLAG of zero:
%       [SIZES]=FINALSIM1([],[],[],0),  returns a vector, SIZES, which
%       contains the sizes of the state vector and other parameters.
%               SIZES(1) number of states
%               SIZES(2) number of discrete states
%               SIZES(3) number of outputs
%               SIZES(4) number of inputs
%               SIZES(5) number of roots (currently unsupported)
%               SIZES(6) direct feedthrough flag
%               SIZES(7) number of sample times
%
%       For the definition of other parameters in SIZES, see SFUNC.
%       See also, TRIM, LINMOD, LINSIM, EULER, RK23, RK45, ADAMS, GEAR.

% Note: This M-file is only used for saving graphical information;
%       after the model is loaded into memory an internal model
%       representation is used.

% the system will take on the name of this mfile:
sys = mfilename;
new_system(sys)
simver(1.3)
if (0 == (nargin + nargout))
   set_param(sys,'Location',[217,259,1158,980])
   open_system(sys)
end;
set_param(sys,'algorithm',     'RK-23')
set_param(sys,'Start time',    '0.0')
set_param(sys,'Stop time',     '2.5')
set_param(sys,'Min step size', '0.01')
```

```
set_param(sys,'Max step size', '0.01')
set_param(sys,'Relative error','1e-3')
set_param(sys,'Return vars', '')

add_block('built-in/Mux',[sys,'/','Mux2'])
set_param([sys,'/','Mux2'],...
                'position',[320,321,375,584])

add_block('built-in/S-Function',[sys,'/','Dynamics'])
set_param([sys,'/','Dynamics'],...
                'function name','surf1',...
                'parameters','0,0,0,0',...
                'position',[535,267,695,373])

add_block('built-in/Demux',[sys,'/','Demux'])
set_param([sys,'/','Demux'],...
                'outputs','5',...
                'position',[725,296,775,344])


%    Subsystem 'XY Graph'.

new_system([sys,'/','XY Graph'])
set_param([sys,'/','XY Graph'],'Location',[8,52,282,245])

add_block('built-in/Inport',[sys,'/','XY Graph/y'])
set_param([sys,'/','XY Graph/y'],...
                'Port','2',...
                'position',[10,100,30,120])

add_block('built-in/Inport',[sys,'/','XY Graph/x'])
set_param([sys,'/','XY Graph/x'],...
                'position',[10,30,30,50])

add_block('built-in/Mux',[sys,'/','XY Graph/Mux'])
set_param([sys,'/','XY Graph/Mux'],...
                'inputs','2',...
                'position',[100,61,130,94])

add_block('built-in/S-Function',[sys,'/',['XY        Graph/S-function',13,'M-file        which
plots',13,'lines',13,'']])
set_param([sys,'/',['XY Graph/S-function',13,'M-file which plots',13,'lines',13,'']],...
                'function name','sfunxy',...
                'parameters','ax, st',...
                'position',[185,70,235,90])
```

```
add_line([sys,'/','XY Graph'],[35,110;70,110;70,85;95,85])
add_line([sys,'/','XY Graph'],[35,40;70,40;70,70;95,70])
add_line([sys,'/','XY Graph'],[135,80;180,80])
set_param([sys,'/','XY Graph'],...
                'Mask
Display','plot(0,0,100,100,[12,91,91,12,12],[90,90,45,45,90],[51,57,65,75,80,79,75,67,60
,54,51,48,42,34,28,27,31,42,51],[71,68,66,66,72,79,83,84,81,77,71,60,54,54,58,65,71,74
,71])')
set_param([sys,'/','XY Graph'],...
                'Mask Type','XY scope.',...
                'Mask Dialogue','XY scope using MATLAB graph window.\nFirst input is
used as time base.\nEnter plotting ranges.|x-min:|x-max:|y-min:|y-max:')
set_param([sys,'/','XY Graph'],...
                'Mask Translate','ax = [@1, @2, @3, @4];st=-1;',...
                'Mask Help','This block can be used to explore limit cycles. Look at the m-
file sfunxy.m to see how it works.',...
                'Mask Entries','-1\/3\/-1\/3\/')


%    Finished composite block 'XY Graph'.

set_param([sys,'/','XY Graph'],...
                'position',[660,69,755,156])

add_block('built-in/Derivative',[sys,'/','Acceleration'])
set_param([sys,'/','Acceleration'],...
                'position',[225,334,295,376])

add_block('built-in/Derivative',[sys,'/','Velocity'])
set_param([sys,'/','Velocity'],...
                'position',[130,399,200,441])

add_block('built-in/Sine Wave',[sys,'/','Position'])
set_param([sys,'/','Position'],...
                'amplitude','10',...
                'frequency','1/16',...
                'position',[20,452,105,518])

add_block('built-in/Constant',[sys,'/','Fd'])
set_param([sys,'/','Fd'],...
                'position',[250,532,285,568])

add_block('built-in/Mux',[sys,'/','Mux4'])
set_param([sys,'/','Mux4'],...
                'inputs','2',...
```

```
                'position',[590,479,650,561])

add_block('built-in/Mux',[sys,'/','Mux5'])
set_param([sys,'/','Mux5'],...
                'inputs','2',...
                'position',[855,289,895,356])

add_block('built-in/Fcn',[sys,'/','Fcn'])
set_param([sys,'/','Fcn'],...
                'orientation',2,...
                'Expr','(u[2]+1*u[1])/((1+1^2)*cos(atan(1)))',...
                'position',[630,413,690,457])

add_block('built-in/To Workspace',[sys,'/','To Workspace2'])
set_param([sys,'/','To Workspace2'],...
                'mat-name','link1',...
                'position',[850,142,900,158])

add_block('built-in/To Workspace',[sys,'/','To Workspace1'])
set_param([sys,'/','To Workspace1'],...
                'mat-name','link2',...
                'position',[850,62,900,78])


%    Subsystem 'Position_Compare'.

new_system([sys,'/','Position_Compare'])
set_param([sys,'/','Position_Compare'],'Location',[0,59,274,252])

add_block('built-in/Inport',[sys,'/','Position_Compare/x'])
set_param([sys,'/','Position_Compare/x'],...
                'position',[65,55,85,75])

add_block('built-in/S-Function',[sys,'/',['Position_Compare/S-function',13,'M-file      which
plots',13,'lines',13,'']])
set_param([sys,'/',['Position_Compare/S-function',13,'M-file which plots',13,'lines',13,'']],...
                'function name','sfunyst',...
                'parameters','ax, color, npts, dt',...
                'position',[130,55,180,75])
add_line([sys,'/','Position_Compare'],[90,65;125,65])
set_param([sys,'/','Position_Compare'],...
                'Mask
Display','plot(0,0,100,100,[83,76,63,52,42,38,28,16,11,84,11,11,11,90,90,11],[75,58,47,5
4,72,80,84,74,65,65,65,90,40,40,90,90])',...
                'Mask Type','Storage scope.')
```

```
set_param([sys,'/','Position_Compare'],...
                'Mask Dialogue','Storage scope using MATLAB graph window.\nEnter
plotting ranges and line type.|Initial Time Range:|Initial y-min:|Initial y-max:|Storage
pts.:|Line type (rgbw-.:xo):')
set_param([sys,'/','Position_Compare'],...
                'Mask Translate','npts = @4; color = @5; ax = [0, @1, @2, @3]; dt=-1;')
set_param([sys,'/','Position_Compare'],...
                'Mask Help','This block uses a MATLAB figure window to plot the input
signal. The graph limits are automatically scaled to the min and max values of the signal
stored in the scope"s signal buffer. Line type must be in quotes. See the M-file
sfunyst.m.')
set_param([sys,'/','Position_Compare'],...
                'Mask Entries','5\/-10\/10\/10000\/"g-/r-/c-./w:/m*/ro/b+"\/')


%    Finished composite block 'Position_Compare'.

set_param([sys,'/','Position_Compare'],...
                'position',[720,489,765,551])

add_block('built-in/Mux',[sys,'/','Mux6'])
set_param([sys,'/','Mux6'],...
                'inputs','2',...
                'position',[815,494,855,561])


%    Subsystem 'Force'.

new_system([sys,'/','Force'])
set_param([sys,'/','Force'],'Location',[0,59,274,252])

add_block('built-in/S-Function',[sys,'/',['Force/S-function',13,'M-file                which
plots',13,'lines',13,'']])
set_param([sys,'/',['Force/S-function',13,'M-file which plots',13,'lines',13,'']],...
                'function name','sfunyst',...
                'parameters','ax, color, npts, dt',...
                'position',[130,55,180,75])

add_block('built-in/Inport',[sys,'/','Force/x'])
set_param([sys,'/','Force/x'],...
                'position',[65,55,85,75])
add_line([sys,'/','Force'],[90,65;125,65])
set_param([sys,'/','Force'],...
```

```
            'Mask
Display','plot(0,0,100,100,[83,76,63,52,42,38,28,16,11,84,11,11,11,90,90,11],[75,58,47,5
4,72,80,84,74,65,65,65,90,40,40,90,90])',...
            'Mask Type','Storage scope.')
set_param([sys,'/','Force'],...
            'Mask Dialogue','Storage scope using MATLAB graph window.\nEnter
plotting ranges and line type.|Initial Time Range:|Initial y-min:|Initial y-max:|Storage
pts.:|Line type (rgbw-.:xo):')
set_param([sys,'/','Force'],...
            'Mask Translate','npts = @4; color = @5; ax = [0, @1, @2, @3]; dt=-1;')
set_param([sys,'/','Force'],...
            'Mask Help','This block uses a MATLAB figure window to plot the input
signal. The graph limits are automatically scaled to the min and max values of the signal
stored in the scope''s signal buffer. Line type must be in quotes. See the M-file
sfunyst.m.')
set_param([sys,'/','Force'],...
            'Mask Entries','2\/-1\/1\/10000\/"b-/r-/c-./w:/m*/ro/b+"\/')


%   Finished composite block 'Force'.

set_param([sys,'/','Force'],...
            'position',[885,529,930,591])
add_line(sys,[110,485;315,485])
add_line(sys,[205,420;315,420])
add_line(sys,[205,420;210,420;220,355])
add_line(sys,[300,355;315,355])
add_line(sys,[110,485;115,485;125,420])
add_line(sys,[700,320;720,320])
add_line(sys,[655,520;715,520])
add_line(sys,[780,310;815,310;815,250;635,250;635,135;655,135])
add_line(sys,[290,550;315,550])
add_line(sys,[780,330;805,330;805,35;645,35;655,90])
add_line(sys,[815,310;840,310;850,305])
add_line(sys,[805,330;840,330;850,340])
add_line(sys,[900,325;910,325;910,435;695,435])
add_line(sys,[625,435;565,435;565,500;585,500])
add_line(sys,[805,35;805,70;845,70])
add_line(sys,[765,250;765,150;845,150])
add_line(sys,[220,485;220,605;505,605;505,540;585,540])
add_line(sys,[380,455;450,455;450,320;530,320])
add_line(sys,[780,340;785,340;785,510;810,510])
add_line(sys,[860,530;865,530;865,560;880,560])
add_line(sys,[290,550;300,550;300,595;545,595;545,570;790,570;790,545;810,545])
```

```
drawnow

% Return any arguments.
if (nargin | nargout)
        % Must use feval here to access system in memory
        if (nargin > 3)
                if (flag == 0)
                        eval(['[ret,x0,str,ts,xts]=',sys,'(t,x,u,flag);'])
                else
                        eval(['ret =', sys,'(t,x,u,flag);'])
                end
        else
                [ret,x0,str,ts,xts] = feval(sys);
        end
else
        drawnow % Flash up the model and execute load callback
end
```

CONSTRAINT SURFACE #2

```
function[sys,x0]=surf2(t,x,u,flag,l1_dot,l1,l2_dot,l2)

if abs(flag) == 1

m1=2; m2=1; kpv=5; kpp=10; kfv=0.1; kfp=0.001;

sd_dd=u(1); sd_d=u(2); sd=u(3); Fd=u(4);

% Determine equivalent point on surface.
% x(1) = l1_dot = y_dot
% x(2) = l1     = y
% x(3) = l2_dot = x_dot
% x(4) = l2     = x
m_star=(x(2)-1)/x(4);
xs=1/sqrt(1+m_star^2);
ys=1-sqrt(1-xs^2);
m=xs/sqrt(1-xs^2);
m_d=x(3)/(1-xs^2)^(3/2);
st=sin(atan(m)); ct=cos(atan(m));

% Use equivalent surface point to determine constraint position.
s=asin(xs); s_d=x(3)/sqrt(1-xs^2);

% Determine position controller.
Qs=(m1*st^2+m2)*(sd_dd+kpv*(sd_d-s_d)+kpp*(sd-s));
F2=Qs/(((m1+m2)/m2)*m*st+ct);
F1=F2*((m1+m2)/m2)*m;

F2=F2-m2*st*(m_d/(1+m^2))*s_d;
F1=F1+(m1+m2)*ct*(m_d/(1+m^2))*s_d;

% Determine force controller.
ke=1000;
delta=sqrt((xs-x(4))^2+(ys-x(2))^2);
delta_d=-x(1)*ct+x(3)*st;
if ((x(4)>xs)&(x(2)<ys))
        global Fm;
        Fm=ke*delta;
else
        global Fm;
        Fm=0;
end
Qn=Fd+kfp*(Fd-Fm)-kfv*ke*delta_d;
```

```
% Superpose the position and force controllers.
F1=F1-Qn*ct;
F2=F2+Qn*st;

% Compute the manipulator dynamics.
        A=[ 0 0 0 0
            1 0 0 0
            0 0 0 0
            0 0 1 0 ];
        B=[ (F1+Fm*ct)/(m1+m2)
            0
            (F2-Fm*st)/m2
            0 ];
        sys=A*x+B;

elseif abs(flag) == 3

global Fm;
if t==0
        Fm=0;
end

sys=[x(1) x(2) x(3) x(4) Fm];

elseif abs(flag) == 0

sys=[4 0 5 4 0 1];
x0=zeros(4,1);
x0(1)=l1_dot;
x0(2)=l1;
x0(3)=l2_dot;
x0(4)=l2;

else

sys=[];

end
```

SIMULATION #2

```
function [ret,x0,str,ts,xts]=finalsim2(t,x,u,flag);
%FINALSIM2      is the M-file description of the SIMULINK system named
FINALSIM2.
%       The block-diagram can be displayed by typing: FINALSIM2.
%
%       SYS=FINALSIM2(T,X,U,FLAG) returns depending on FLAG certain
%       system values given time point, T, current state vector, X,
%       and input vector, U.
%       FLAG is used to indicate the type of output to be returned in SYS.
%
%       Setting FLAG=1 causes FINALSIM2 to return state derivatives, FLAG=2
%       discrete states, FLAG=3 system outputs and FLAG=4 next sample
%       time. For more information and other options see SFUNC.
%
%       Calling FINALSIM2 with a FLAG of zero:
%       [SIZES]=FINALSIM2([],[],[],0),  returns a vector, SIZES, which
%       contains the sizes of the state vector and other parameters.
%               SIZES(1) number of states
%               SIZES(2) number of discrete states
%               SIZES(3) number of outputs
%               SIZES(4) number of inputs
%               SIZES(5) number of roots (currently unsupported)
%               SIZES(6) direct feedthrough flag
%               SIZES(7) number of sample times
%
%       For the definition of other parameters in SIZES, see SFUNC.
%       See also, TRIM, LINMOD, LINSIM, EULER, RK23, RK45, ADAMS, GEAR.

% Note: This M-file is only used for saving graphical information;
%       after the model is loaded into memory an internal model
%       representation is used.

% the system will take on the name of this mfile:
sys = mfilename;
new_system(sys)
simver(1.3)
if (0 == (nargin + nargout))
   set_param(sys,'Location',[268,217,1240,938])
   open_system(sys)
end;
set_param(sys,'algorithm',     'RK-23')
set_param(sys,'Start time',    '0.0')
set_param(sys,'Stop time',     '2')
set_param(sys,'Min step size', '0.01')
```

```
set_param(sys,'Max step size', '0.01')
set_param(sys,'Relative error','1e-3')
set_param(sys,'Return vars', '')

add_block('built-in/Mux',[sys,'/','Mux2'])
set_param([sys,'/','Mux2'],...
                'position',[320,321,375,584])

add_block('built-in/S-Function',[sys,'/','Dynamics'])
set_param([sys,'/','Dynamics'],...
                'function name','surf2',...
                'parameters','0,0,0,0',...
                'position',[535,267,695,373])

add_block('built-in/Demux',[sys,'/','Demux'])
set_param([sys,'/','Demux'],...
                'outputs','5',...
                'position',[725,296,775,344])


%    Subsystem  'XY Graph'.

new_system([sys,'/','XY Graph'])
set_param([sys,'/','XY Graph'],'Location',[8,52,282,245])

add_block('built-in/Inport',[sys,'/','XY Graph/y'])
set_param([sys,'/','XY Graph/y'],...
                'Port','2',...
                'position',[10,100,30,120])

add_block('built-in/Inport',[sys,'/','XY Graph/x'])
set_param([sys,'/','XY Graph/x'],...
                'position',[10,30,30,50])

add_block('built-in/Mux',[sys,'/','XY Graph/Mux'])
set_param([sys,'/','XY Graph/Mux'],...
                'inputs','2',...
                'position',[100,61,130,94])

add_block('built-in/S-Function',[sys,'/',['XY        Graph/S-function',13,'M-file        which
plots',13,'lines',13,"]])
set_param([sys,'/',['XY Graph/S-function',13,'M-file which plots',13,'lines',13,"]],...
                'function name','sfunxy',...
                'parameters','ax, st',...
                'position',[185,70,235,90])
```

```
add_line([sys,'/','XY Graph'],[35,110;70,110;70,85;95,85])
add_line([sys,'/','XY Graph'],[35,40;70,40;70,70;95,70])
add_line([sys,'/','XY Graph'],[135,80;180,80])
set_param([sys,'/','XY Graph'],...
                'Mask
Display','plot(0,0,100,100,[12,91,91,12,12],[90,90,45,45,90],[51,57,65,75,80,79,75,67,60
,54,51,48,42,34,28,27,31,42,51],[71,68,66,66,72,79,83,84,81,77,71,60,54,54,58,65,71,74
,71])')
set_param([sys,'/','XY Graph'],...
                'Mask Type','XY scope.',...
                'Mask Dialogue','XY scope using MATLAB graph window.\nFirst input is
used as time base.\nEnter plotting ranges.|x-min:|x-max:|y-min:|y-max:')
set_param([sys,'/','XY Graph'],...
                'Mask Translate','ax = [@1, @2, @3, @4];st=-1;',...
                'Mask Help','This block can be used to explore limit cycles. Look at the m-
file sfunxy.m to see how it works.',...
                'Mask Entries','-1\/3\/-1\/3\/')


%    Finished composite block 'XY Graph'.

set_param([sys,'/','XY Graph'],...
                'position',[660,69,755,156])

add_block('built-in/Derivative',[sys,'/','Acceleration'])
set_param([sys,'/','Acceleration'],...
                'position',[225,334,295,376])

add_block('built-in/Derivative',[sys,'/','Velocity'])
set_param([sys,'/','Velocity'],...
                'position',[130,399,200,441])

add_block('built-in/Sine Wave',[sys,'/','Position'])
set_param([sys,'/','Position'],...
                'amplitude','10',...
                'frequency','1/16',...
                'position',[20,452,105,518])

add_block('built-in/Constant',[sys,'/','Fd'])
set_param([sys,'/','Fd'],...
                'position',[250,532,285,568])

add_block('built-in/Mux',[sys,'/','Mux4'])
set_param([sys,'/','Mux4'],...
                'inputs','2',...
```

```
                'position',[590,479,650,561])

add_block('built-in/Fcn',[sys,'/','Fcn'])
set_param([sys,'/','Fcn'],...
                'orientation',2,...
                'Expr','asin(1/sqrt(1+((u[1]-1)/u[2])^2))',...
                'position',[630,413,690,457])

add_block('built-in/To Workspace',[sys,'/','To Workspace2'])
set_param([sys,'/','To Workspace2'],...
                'mat-name','link1',...
                'position',[850,142,900,158])

add_block('built-in/To Workspace',[sys,'/','To Workspace1'])
set_param([sys,'/','To Workspace1'],...
                'mat-name','link2',...
                'position',[850,62,900,78])


%    Subsystem 'Position_Compare'.

new_system([sys,'/','Position_Compare'])
set_param([sys,'/','Position_Compare'],'Location',[0,59,274,252])

add_block('built-in/Inport',[sys,'/','Position_Compare/x'])
set_param([sys,'/','Position_Compare/x'],...
                'position',[65,55,85,75])

add_block('built-in/S-Function',[sys,'/',['Position_Compare/S-function',13,'M-file    which
plots',13,'lines',13,'']])
set_param([sys,'/',['Position_Compare/S-function',13,'M-file which plots',13,'lines',13,'']],...
                'function name','sfunyst',...
                'parameters','ax, color, npts, dt',...
                'position',[130,55,180,75])
add_line([sys,'/','Position_Compare'],[90,65;125,65])
set_param([sys,'/','Position_Compare'],...
                'Mask
Display','plot(0,0,100,100,[83,76,63,52,42,38,28,16,11,84,11,11,11,90,90,11],[75,58,47,5
4,72,80,84,74,65,65,65,90,40,40,90,90])',...
                'Mask Type','Storage scope.')
set_param([sys,'/','Position_Compare'],...
                'Mask Dialogue','Storage scope using MATLAB graph window.\nEnter
plotting ranges and line type.|Initial Time Range:|Initial y-min:|Initial y-max:|Storage
pts.:|Line type (rgbw-.:xo):')
set_param([sys,'/','Position_Compare'],...
```

```
                'Mask Translate','npts = @4; color = @5; ax = [0, @1, @2, @3]; dt=-1;')
set_param([sys,'/','Position_Compare'],...
                'Mask Help','This block uses a MATLAB figure window to plot the input
signal. The graph limits are automatically scaled to the min and max values of the signal
stored in the scope"s signal buffer. Line type must be in quotes. See the M-file
sfunyst.m.')
set_param([sys,'/','Position_Compare'],...
                'Mask Entries','5\/-10\/10\/10000\/"g-/r-/c-./w:/m*/ro/b+"\/')


%    Finished composite block 'Position_Compare'.

set_param([sys,'/','Position_Compare'],...
                'position',[720,489,765,551])

add_block('built-in/Mux',[sys,'/','Mux5'])
set_param([sys,'/','Mux5'],...
                'inputs','2',...
                'position',[840,289,900,371])

add_block('built-in/Mux',[sys,'/','Mux6'])
set_param([sys,'/','Mux6'],...
                'inputs','2',...
                'position',[805,519,865,601])


%    Subsystem 'Force'.

new_system([sys,'/','Force'])
set_param([sys,'/','Force'],'Location',[0,59,274,252])

add_block('built-in/S-Function',[sys,'/',['Force/S-function',13,'M-file                    which
plots',13,'lines',13,'']])
set_param([sys,'/',['Force/S-function',13,'M-file which plots',13,'lines',13,'']],...
                'function name','sfunyst',...
                'parameters','ax, color, npts, dt',...
                'position',[130,55,180,75])

add_block('built-in/Inport',[sys,'/','Force/x'])
set_param([sys,'/','Force/x'],...
                'position',[65,55,85,75])
add_line([sys,'/','Force'],[90,65;125,65])
set_param([sys,'/','Force'],...
```

```
                'Mask
Display','plot(0,0,100,100,[83,76,63,52,42,38,28,16,11,84,11,11,11,90,90,11],[75,58,47,5
4,72,80,84,74,65,65,65,90,40,40,90,90])',...
                'Mask Type','Storage scope.')
set_param([sys,'/','Force'],...
                'Mask Dialogue','Storage scope using MATLAB graph window.\nEnter
plotting ranges and line type.|Initial Time Range:|Initial y-min:|Initial y-max:|Storage
pts.:|Line type (rgbw-.:xo):')
set_param([sys,'/','Force'],...
                'Mask Translate','npts = @4; color = @5; ax = [0, @1, @2, @3]; dt=-1;')
set_param([sys,'/','Force'],...
                'Mask Help','This block uses a MATLAB figure window to plot the input
signal. The graph limits are automatically scaled to the min and max values of the signal
stored in the scope"s signal buffer. Line type must be in quotes. See the M-file
sfunyst.m.')
set_param([sys,'/','Force'],...
                'Mask Entries','2\/-1\/1\/10000\/"b-/r-/c-./w:/m*/ro/b+"\/')


%    Finished composite block 'Force'.

set_param([sys,'/','Force'],...
                'position',[905,554,950,616])
add_line(sys,[110,485;315,485])
add_line(sys,[205,420;315,420])
add_line(sys,[205,420;210,420;220,355])
add_line(sys,[300,355;315,355])
add_line(sys,[110,485;115,485;125,420])
add_line(sys,[700,320;720,320])
add_line(sys,[655,520;715,520])
add_line(sys,[780,310;815,310;815,250;635,250;635,135;655,135])
add_line(sys,[290,550;315,550])
add_line(sys,[780,330;805,330;805,35;645,35;655,90])
add_line(sys,[625,435;565,435;565,500;585,500])
add_line(sys,[805,35;805,70;845,70])
add_line(sys,[765,250;765,150;845,150])
add_line(sys,[220,485;220,605;505,605;505,540;585,540])
add_line(sys,[380,455;450,455;450,320;530,320])
add_line(sys,[815,310;835,310])
add_line(sys,[805,330;825,330;835,350])
add_line(sys,[905,330;915,330;915,435;695,435])
add_line(sys,[780,340;785,340;785,540;800,540])
add_line(sys,[300,550;300,605;650,605;650,580;800,580])
add_line(sys,[870,560;880,560;880,585;900,585])
```

```
drawnow

% Return any arguments.
if (nargin | nargout)
        % Must use feval here to access system in memory
        if (nargin > 3)
                if (flag == 0)
                        eval(['[ret,x0,str,ts,xts]=',sys,'(t,x,u,flag);'])
                else
                        eval(['ret =', sys,'(t,x,u,flag);'])
                end
        else
                [ret,x0,str,ts,xts] = feval(sys);
        end
else
        drawnow % Flash up the model and execute load callback
end
```

CONSTRAINT SURFACE #3

```
function[sys,x0]=surf3(t,x,u,flag,l1_dot,l1,l2_dot,l2)

if abs(flag) == 1

m1=2; m2=1; kpv=5; kpp=10; kfv=0.075; kfp=0.001;

sd_dd=u(1); sd_d=u(2); sd=u(3); Fd=u(4);

% Determine equivalent point on surface.
% x(1) = l1_dot = y_dot
% x(2) = l1     = y
% x(3) = l2_dot = x_dot
% x(4) = l2     = x
m_star=-x(2)/(1-x(4));
xs=1-(1/sqrt(m_star^2+1));
ys=sqrt(1-(xs-1)^2);

m=(1-xs)/sqrt(1-(xs-1)^2);
m_d=-x(3)/(1-(xs-1)^2)^(3/2);
st=sin(atan(m)); ct=cos(atan(m));

% Use equivalent surface point to determine constraint position.
s=quad('int_convex',0.1,xs); s_d=(1/sqrt(1-(1-xs)^2))*x(3);

% Determine position controller.
Qs=(m1*st^2+m2)*(sd_dd+kpv*(sd_d-s_d)+kpp*(sd-s));
F2=Qs/(((m1+m2)/m2)*m*st+ct);
F1=F2*((m1+m2)/m2)*m;

F2=F2-m2*st*(m_d/(1+m^2))*s_d;
F1=F1+(m1+m2)*ct*(m_d/(1+m^2))*s_d;

% Determine force controller.
ke=1000;
delta=sqrt((xs-x(4))^2+(ys-x(2))^2);
delta_d=-x(1)*ct+x(3)*st;
if ((x(4)>xs)&(x(2)<ys))
        global Fm;
        Fm=ke*delta;
else
        global Fm;
        Fm=0;
end
Qn=Fd+kfp*(Fd-Fm)-kfv*ke*delta_d;
```

```
% Superpose the position and force controllers.
F1=F1-Qn*ct;
F2=F2+Qn*st;

% Compute the manipulator dynamics.
        A=[ 0 0 0 0
            1 0 0 0
            0 0 0 0
            0 0 1 0 ];
        B=[ (F1+Fm*ct)/(m1+m2)
            0
            (F2-Fm*st)/m2
            0 ];
        sys=A*x+B;

elseif abs(flag) == 3

global Fm;
if t==0
        Fm=0;
end

sys=[x(1) x(2) x(3) x(4) Fm];

elseif abs(flag) == 0

sys=[4 0 5 4 0 1];
x0=zeros(4,1);
x0(1)=l1_dot;
x0(2)=l1;
x0(3)=l2_dot;
x0(4)=l2;

else

sys=[];

end
```

SIMULATION #3

```
function [ret,x0,str,ts,xts]=finalsim3(t,x,u,flag);
%FINALSIM3      is the M-file description of the SIMULINK system named
FINALSIM3.
%       The block-diagram can be displayed by typing: FINALSIM3.
%
%       SYS=FINALSIM3(T,X,U,FLAG) returns depending on FLAG certain
%       system values given time point, T, current state vector, X,
%       and input vector, U.
%       FLAG is used to indicate the type of output to be returned in SYS.
%
%       Setting FLAG=1 causes FINALSIM3 to return state derivatives, FLAG=2
%       discrete states, FLAG=3 system outputs and FLAG=4 next sample
%       time. For more information and other options see SFUNC.
%
%       Calling FINALSIM3 with a FLAG of zero:
%       [SIZES]=FINALSIM3([],[],[],0),  returns a vector, SIZES, which
%       contains the sizes of the state vector and other parameters.
%               SIZES(1) number of states
%               SIZES(2) number of discrete states
%               SIZES(3) number of outputs
%               SIZES(4) number of inputs
%               SIZES(5) number of roots (currently unsupported)
%               SIZES(6) direct feedthrough flag
%               SIZES(7) number of sample times
%
%       For the definition of other parameters in SIZES, see SFUNC.
%       See also, TRIM, LINMOD, LINSIM, EULER, RK23, RK45, ADAMS, GEAR.

% Note: This M-file is only used for saving graphical information;
%       after the model is loaded into memory an internal model
%       representation is used.

% the system will take on the name of this mfile:
sys = mfilename;
new_system(sys)
simver(1.3)
if (0 == (nargin + nargout))
   set_param(sys,'Location',[274,209,1215,929])
   open_system(sys)
end;
set_param(sys,'algorithm',     'RK-23')
set_param(sys,'Start time',    '0.0')
set_param(sys,'Stop time',     '2')
set_param(sys,'Min step size', '0.01')
```

```
set_param(sys,'Max step size', '0.01')
set_param(sys,'Relative error','1e-3')
set_param(sys,'Return vars',  '')

add_block('built-in/Mux',[sys,'/','Mux2'])
set_param([sys,'/','Mux2'],...
                'position',[320,321,375,584])

add_block('built-in/S-Function',[sys,'/','Dynamics'])
set_param([sys,'/','Dynamics'],...
                'function name','surf3',...
                'parameters','0,0.4359,0,0.1',...
                'position',[535,267,695,373])

add_block('built-in/Demux',[sys,'/','Demux'])
set_param([sys,'/','Demux'],...
                'outputs','5',...
                'position',[725,296,775,344])


%    Subsystem  'XY Graph'.

new_system([sys,'/','XY Graph'])
set_param([sys,'/','XY Graph'],'Location',[8,52,282,245])

add_block('built-in/Inport',[sys,'/','XY Graph/y'])
set_param([sys,'/','XY Graph/y'],...
                'Port','2',...
                'position',[10,100,30,120])

add_block('built-in/Inport',[sys,'/','XY Graph/x'])
set_param([sys,'/','XY Graph/x'],...
                'position',[10,30,30,50])

add_block('built-in/Mux',[sys,'/','XY Graph/Mux'])
set_param([sys,'/','XY Graph/Mux'],...
                'inputs','2',...
                'position',[100,61,130,94])

add_block('built-in/S-Function',[sys,'/',['XY        Graph/S-function',13,'M-file        which
plots',13,'lines',13,'']])
set_param([sys,'/',['XY Graph/S-function',13,'M-file which plots',13,'lines',13,'']],...
                'function name','sfunxy',...
                'parameters','ax, st',...
                'position',[185,70,235,90])
```

```
add_line([sys,'/','XY Graph'],[35,110;70,110;70,85;95,85])
add_line([sys,'/','XY Graph'],[35,40;70,40;70,70;95,70])
add_line([sys,'/','XY Graph'],[135,80;180,80])
set_param([sys,'/','XY Graph'],...
                'Mask
Display','plot(0,0,100,100,[12,91,91,12,12],[90,90,45,45,90],[51,57,65,75,80,79,75,67,60
,54,51,48,42,34,28,27,31,42,51],[71,68,66,66,72,79,83,84,81,77,71,60,54,54,58,65,71,74
,71])')
set_param([sys,'/','XY Graph'],...
                'Mask Type','XY scope.',...
                'Mask Dialogue','XY scope using MATLAB graph window.\nFirst input is
used as time base.\nEnter plotting ranges.|x-min:|x-max:|y-min:|y-max:')
set_param([sys,'/','XY Graph'],...
                'Mask Translate','ax = [@1, @2, @3, @4];st=-1;',...
                'Mask Help','This block can be used to explore limit cycles. Look at the m-
file sfunxy.m to see how it works.',...
                'Mask Entries','-1\/3\/-1\/3\/')


%    Finished composite block 'XY Graph'.

set_param([sys,'/','XY Graph'],...
                'position',[660,69,755,156])

add_block('built-in/Derivative',[sys,'/','Acceleration'])
set_param([sys,'/','Acceleration'],...
                'position',[225,334,295,376])

add_block('built-in/Derivative',[sys,'/','Velocity'])
set_param([sys,'/','Velocity'],...
                'position',[130,399,200,441])

add_block('built-in/Sine Wave',[sys,'/','Position'])
set_param([sys,'/','Position'],...
                'amplitude','10',...
                'frequency','1/16',...
                'position',[20,452,105,518])

add_block('built-in/Constant',[sys,'/','Fd'])
set_param([sys,'/','Fd'],...
                'position',[250,532,285,568])

add_block('built-in/Mux',[sys,'/','Mux4'])
set_param([sys,'/','Mux4'],...
                'inputs','2',...
```

```
                'position',[590,479,650,561])

add_block('built-in/To Workspace',[sys,'/','To Workspace2'])
set_param([sys,'/','To Workspace2'],...
                'mat-name','link1',...
                'position',[850,142,900,158])

add_block('built-in/To Workspace',[sys,'/','To Workspace1'])
set_param([sys,'/','To Workspace1'],...
                'mat-name','link2',...
                'position',[850,62,900,78])


%    Subsystem 'Position_Compare'.

new_system([sys,'/','Position_Compare'])
set_param([sys,'/','Position_Compare'],'Location',[0,59,274,252])

add_block('built-in/Inport',[sys,'/','Position_Compare/x'])
set_param([sys,'/','Position_Compare/x'],...
                'position',[65,55,85,75])

add_block('built-in/S-Function',[sys,'/',['Position_Compare/S-function',13,'M-file      which
plots',13,'lines',13,'']])
set_param([sys,'/',['Position_Compare/S-function',13,'M-file which plots',13,'lines',13,'']],...
                'function name','sfunyst',...
                'parameters','ax, color, npts, dt',...
                'position',[130,55,180,75])
add_line([sys,'/','Position_Compare'],[90,65;125,65])
set_param([sys,'/','Position_Compare'],...
                'Mask
Display','plot(0,0,100,100,[83,76,63,52,42,38,28,16,11,84,11,11,11,90,90,11],[75,58,47,5
4,72,80,84,74,65,65,65,90,40,40,90,90])',...
                'Mask Type','Storage scope.')
set_param([sys,'/','Position_Compare'],...
                'Mask Dialogue','Storage scope using MATLAB graph window.\nEnter
plotting ranges and line type.|Initial Time Range:|Initial y-min:|Initial y-max:|Storage
pts.:|Line type (rgbw-.:xo):')
set_param([sys,'/','Position_Compare'],...
                'Mask Translate','npts = @4; color = @5; ax = [0, @1, @2, @3]; dt=-1;')
set_param([sys,'/','Position_Compare'],...
                'Mask Help','This block uses a MATLAB figure window to plot the input
signal.  The graph limits are automatically scaled to the min and max values of the signal
stored in the scope"s signal buffer.  Line type must be in quotes.  See the M-file
sfunyst.m.')
```

```
set_param([sys,'/','Position_Compare'],...
                'Mask Entries','5\/-10\/10\/10000\/"g-/r-/c-./w:/m*/ro/b+"\/')


%    Finished composite block 'Position_Compare'.

set_param([sys,'/','Position_Compare'],...
                'position',[720,489,765,551])

add_block('built-in/MATLAB Fcn',[sys,'/','MATLAB Fcn'])
set_param([sys,'/','MATLAB Fcn'],...
                'orientation',2,...
                'MATLAB Fcn','quad("int_convex",0.1,u)',...
                'position',[630,403,705,447])

add_block('built-in/Fcn',[sys,'/','Fcn'])
set_param([sys,'/','Fcn'],...
                'position',[625,600,665,620])

add_block('built-in/Mux',[sys,'/','Mux5'])
set_param([sys,'/','Mux5'],...
                'inputs','2',...
                'position',[800,554,860,636])


%    Subsystem 'Force'.

new_system([sys,'/','Force'])
set_param([sys,'/','Force'],'Location',[0,59,274,252])

add_block('built-in/S-Function',[sys,'/',['Force/S-function',13,'M-file                    which
plots',13,'lines',13,"]])
set_param([sys,'/',['Force/S-function',13,'M-file which plots',13,'lines',13,"]],...
                'function name','sfunyst',...
                'parameters','ax, color, npts, dt',...
                'position',[130,55,180,75])

add_block('built-in/Inport',[sys,'/','Force/x'])
set_param([sys,'/','Force/x'],...
                'position',[65,55,85,75])
add_line([sys,'/','Force'],[90,65;125,65])
set_param([sys,'/','Force'],...
                'Mask
Display','plot(0,0,100,100,[83,76,63,52,42,38,28,16,11,84,11,11,11,90,90,11],[75,58,47,5
4,72,80,84,74,65,65,65,90,40,40,90,90])',...
```

```
                'Mask Type','Storage scope.')
set_param([sys,'/','Force'],...
                'Mask Dialogue','Storage scope using MATLAB graph window.\nEnter plotting ranges and line type.|Initial Time Range:|Initial y-min:|Initial y-max:|Storage pts.:|Line type (rgbw-.:xo):')
set_param([sys,'/','Force'],...
                'Mask Translate','npts = @4; color = @5; ax = [0, @1, @2, @3]; dt=-1;')
set_param([sys,'/','Force'],...
                'Mask Help','This block uses a MATLAB figure window to plot the input signal. The graph limits are automatically scaled to the min and max values of the signal stored in the scope"s signal buffer. Line type must be in quotes. See the M-file sfunyst.m.')
set_param([sys,'/','Force'],...
                'Mask Entries','2\/-1\/1\/10000\/"b-/r-/c-./w:/m*/ro/b+"\/')


%    Finished composite block 'Force'.

set_param([sys,'/','Force'],...
                'position',[890,484,935,546])
add_line(sys,[205,420;315,420])
add_line(sys,[205,420;210,420;220,355])
add_line(sys,[300,355;315,355])
add_line(sys,[110,485;115,485;125,420])
add_line(sys,[700,320;720,320])
add_line(sys,[655,520;715,520])
add_line(sys,[780,310;815,310;815,250;635,250;635,135;655,135])
add_line(sys,[290,550;315,550])
add_line(sys,[780,330;805,330;805,35;645,35;655,90])
add_line(sys,[625,425;565,425;565,500;585,500])
add_line(sys,[805,35;805,70;845,70])
add_line(sys,[765,250;765,150;845,150])
add_line(sys,[380,455;450,455;450,320;530,320])
add_line(sys,[110,485;120,485;120,530;145,530;315,485])
add_line(sys,[805,330;805,425;710,425])
add_line(sys,[145,530;180,530;180,590;380,590;380,540;585,540])
add_line(sys,[780,340;785,340;795,575])
add_line(sys,[300,550;300,640;745,640;745,615;795,615])
add_line(sys,[865,595;870,595;870,515;885,515])

drawnow

% Return any arguments.
if (nargin | nargout)
        % Must use feval here to access system in memory
```

```
        if (nargin > 3)
                if (flag == 0)
                        eval(['[ret,x0,str,ts,xts]=',sys,'(t,x,u,flag);'])
                else
                        eval(['ret =', sys,'(t,x,u,flag);'])
                end
        else
                [ret,x0,str,ts,xts] = feval(sys);
        end
else
        drawnow % Flash up the model and execute load callback
end
```