Robotic Control and Nonholonomic Motion Planning

Richard M. Murray

Memorandum No. UCB/ERL (unknown)

December 19, 1990

Electronics Research Laboratory

College of Engineering
University of California, Berkeley
94720

Abstract

This dissertation addresses the problem of control and kinematic planning for constrained robot systems. An example of a system of this type is a multifingered robot hand grasping an object. The individual fingers act as robot manipulators and are constrained by their contact with the object. If the contacts allow rolling between the object and the fingertips, it is possible for the constraints to be nonholonomic. That is, the constraints may not restrict the reachable configurations of the system, but rather, constrain only the allowable velocities of the system.

Using the multifingered hand as a motivating example, this dissertation presents a detailed analysis of the kinematics, dynamics, and control of robot systems with contact constraints. In particular, it presents a unified derivation of the dynamics of robot manipulators with Pfaffian velocity constraints, including the nonholonomic case. This derivation allows control laws to be specified which are provably stable for an entire class of systems, including unconstrained robots, robot hands, and other systems of multiple robots performing a coordinated task. A method for building complex controllers which respects this class of constraints is also developed using a set of simple primitives which allow hierarchical control structures to be created in an organized fashion.

Finally, the nonholonomic motion planning problem is introduced and discussed in detail. Using tools from differential geometric control theory, it is possible to classify and analyze systems with nonholonomic constraints. A brief review of the necessary tools along with a review of the current literature is presented. A practical method for steering nonholonomic systems using sinusoids is derived and applied to several kinematic systems with contact constraints.

Acknowledgements

First, I would like to thank my advisor, Shankar Sastry, for all of his support and encouragement over the past 5 years. Shankar's enthusiasm for all the things we have worked on has been an inspiration to me and I hope that I can follow his example as I continue my research. His attention to detail and demand for rigor have taught me the importance and joy of doing things right.

Several other professors have also enriched my education here at Berkeley. I would like to thank the members of my qualifying committee, Professors Desoer, Sastry, Canny, and Weinstein, for their time and effort. In addition to serving as my qualifying committee chair, Professor Desoer has also been a source of advice and inspiration during my entire graduate career. I would also like to thank the members of my thesis committee, Professors Sastry, Canny, and Marsden, for many useful comments which improved the quality of this dissertation. In addition, I had the pleasure of working with John Canny, Ron Fearing, and Richard Montgomery on a variety of projects related to my research.

Many members of the robotics lab have contributed to my studies at Berkeley and made the PhD process a very pleasant one. I would particularly like to thank Kris Pister for putting up with me for four and a half years and for sharing in at least as many interesting discussions as we did pots of coffee. I am also indebted to Curt Deno for his insights and for stimulating my interest in biological motor control and non-invasive surgical procedures. I would also like to thank John Hauser, Ping Hsu, Zexiang Li, Greg Heinzinger, Paul Jacobs, Raja Kadayala, Andrew Teel, Matt Berkemeier, Ed Nicolson, Greg Walsh, Karin Hollerbach, and the other members of the U.C. Berkeley robotics and control groups for the endless stream of conversations, questions, answers, encouragement, and amusement which are at the heart of any graduate student's life.

I would also like to acknowledge the financial support that I have received during my stay here. My research has been funded in part by the National Science Foundation, under grants DMC-84-51129, ECS-87-19298, and IRI-90-14490, NASA, under grant NAG-2-243, the Semiconductor Research Corporation, an Earl C. Anthony Scholarship, and an IBM Manufacturing fellowship.

I have also benefited from many people who are outside of the academic environment. My parents have provided emotional, intellectual, and monetary support on countless occasions. I would like to thank my brother, Tom, and

David Hull for sending me e-mail and giving me something to talk about besides my research. And I would like to especially thank Mr. Henderson, my high school math teacher, for introducing me to the joy of mathematics and laying the foundation for my later studies.

Finally, I would like to thank my wife, RuthAnne. She has watched me go through more joy and suffering than any other person; her love and support have enabled me both to complete my PhD and to remain sane at the same time.

Contents

Chapter 1

Introduction

This dissertation addresses the problem of closed loop control and kinematic planning for constrained robot systems. Although the main emphasis is on constrained systems, most of the techniques can also be applied to unconstrained systems. In the unconstrained case, they reduce to methods which are common in robotics. Thus we are interested in studying controllers for a large class of robotic systems which includes single, multiple, constrained and unconstrained manipulators performing a coordinated task.

A single unconstrained robot is an actuated mechanical system which is controlled in some way to perform a useful task. One of the first problems in using a robot to perform a task is free space motion. That is, we would like to command the actuators to move the robot from one position to another, without worry about obstacles (initially). More generally, we may be given a trajectory which the robot should follow.

How we command the actuators to follow a given trajectory depends on the type of actuators used. If a robot has a stepper motor at each joint, we need only command the stepper motors to be at the desired position at the desired time. Assuming the trajectory is sufficiently slow and smooth, this is easily done. In effect, the control problem is solved by using the controllers inherent in a stepper motor. Much greater freedom is possible if the actuators can apply arbitrary torques at each joint. In this case, we are free to choose the torques based on our own measurements of the state of the system.

Once the control problem for robotic manipulators has been solved, the trajectory planning phase begins. We now try to construct a trajectory which accomplishes the desired task. This trajectory is passed to the robot controller for execution. Path planning typically is concerned with methods for avoiding obstacles in the workspace of the robot.

An unconstrained mechanical system is limited to performing simple pick and place tasks. The manipulator moves to a location, picks up an object, transports that object and deposits it. While there are many tasks which require no more sophistication than this, other tasks do not fit this paradigm. One such class of operations is those for which a robot manipulator must satisfy kinematic constraints.

Kinematic constraints for manipulators have many sources. Perhaps the most fundamental is a robot sliding or rolling along a surface. In this case, the robot is free to move tangential to the surface, but not perpendicular to it. Actually, the constraint is unidirectional, since we can move away from a surface but not into it. More complicated kinematic constraints can be found when a system of robots is performing a coordinated task, such as three robots lifting a single object, or a set of fingers grasping an object.

The introduction of constraints introduces several complications. The manipulators are no longer free to move in any direction. Controllers should be aware of this restriction if a task is to be performed correctly. In the grasping example, a controller must maintain its grip on an object without applying excessive force. This is difficult to achieve with simple position controllers which attempt to hold the individual fingers in a fixed location. Small errors in sensing or modeling cause the object to be dropped.

To control such systems we must first model them. For simple contacts (such as a robot gripper rigidly grabbing an object) the kinematic constraint is modeled as an algebraic function relating the position of the object to the configuration of the manipulator. More complicated constraints, such as one object rolling against another, as in a finger rolling against an object, require more complicated models. These constraints cannot always be represented as algebraic constraints between configuration variables.

Once a representation for the constraints have been selected, a dynamic model which observes the constraints must be constructed. Using this dynamic model, a new controller can be generated. Part of the contribution of this dissertation is to provide a technique for constructing the dynamics of a constrained system in such a way that it resembles the dynamics of an unconstrained manipulator. For some types of constraints, this method is equivalent to choosing a different set of generalized coordinates for the system. In more complicated, and realistic, examples, the technique is considerably more general than this.

Finally, we must reconsider the motion planning problem. In the case that the constraint is holonomic, motion of the system is restricted to a manifold in the configuration space. Abstractly, we can reduce the planning problem to planning in local coordinates on this manifold, and hence it is equivalent to the usual path planning problem in $\mathbb{R}^n$. If the constraints are not holonomic, or we cannot explicitly characterize the submanifold, the planning problem becomes much more difficult. We must plan motion which both avoid obstacles *and* satisfy the constraints. This type of planning, called nonholonomic motion planning, has only recently been studied.

Overview of the dissertation

This dissertation contains a systematic description and analysis of control and planning for constrained robot systems. We use as our primary example a multifingered robot hand grasping and manipulating an object. This example contains all of the elements in which we are interested—nonholonomic constraints, multiple robots performing a coordinated task, and kinematic and actuator redundancy.

The first task is to study the form of the kinematic constraints which might be present in a robotic system. These constraints arise from contact between two objects. We review the formulation of the kinematics of contact using a notation suited for our application. These equations of contact are equivalent to other derivations available in the literature.

Using the form of the constraints, we proceed to derive the equations of motion for the system. The primary contribution of the derivation is to show that the equations of motion for a large class of robot systems can be written in a consistent form with consistent properties. The consequence of this derivation is that we can easily extend controllers for simple robots into controllers for robot systems performing a coordinated task.

Having derived the dynamics for constrained systems, we proceed to review some standard controllers applied in the general context. The shortcomings of these controllers suggest the formulation of a new robot control law which allows position and stiffness to be controlled in complementary directions. Experimental results of this control law are presented to verify its correctness in application.

The structure of the dynamics is sufficiently straightforward that it admits an automated formulation. We exploit this by defining a set of control primitives which can be used to construct complicated controllers for robot systems. The control structures generated by these primitives are motivated by biological control structures.

Finally, we study the planning problem for constrained systems. Since this problem has only recently received attention in the robotics and control literature, we begin with a review of the mathematical tools necessary to study such systems. We also review several techniques that have been applied by various researchers.

A new approach to controlling nonholonomic systems is presented in Chapter 6. Using sinusoids at integrally related frequencies, we show how to generate motion for systems with nonholonomic constraints. These methods are first applied to a set of canonical systems and then extended for use in non-canonical systems.

The primary contributions of this dissertation are in the areas of robotic control and nonholonomic motion planning. First we present a unified formulation of the dynamics and control for robot manipulators. This formulation allows us to construct control laws which can be applied to a large class of systems.

This unified approach also provides a basis for the definition of a concise set of primitives for constructing complex robot controllers for complex systems. Secondly, a method is provided for generating feasible trajectories for nonholonomic systems. This method brings out the structure present in many nonholonomic systems in a very interesting manner. The study of nonholonomic motion planning is in its infancy; the research presented here provides first steps toward a better understanding of this problem and computational approaches toward its solution.

Chapter 2

Contact Kinematics and Statics

This chapter is an introduction to the kinematics and statics of contact. The material is presented from the perspective of grasping, although the formulation can be applied to other contact situations. We derive the basic velocity and force transformations for both fixed and rolling contacts. Using these relationships, we derive the general form of the constraints which are present in a contact environment. The results in this chapter are summarized on page 23.

The material in this chapter is largely based on the works of several authors. Early work in formulating the grasping problem can be found in Salisbury [83]. More appropriate to the presentation given here is the thesis of Kerr [47], the work of Li and Sastry [63], and several papers based on these works [48, 21, 60]. An earlier, alternate derivation of the equations of rolling has been given by Montana [68]. Other specific references are contained in the appropriate locations below.

2.1 Rigid body kinematics

A rigid motion of an object is a motion which preserves distance and orientation. Every such rigid motion can be represented by a rotation followed by a translation. Letting $SO(3)$ represent the group of all proper 3×3 rotation matrices and $\mathbb{R}$ denote the real numbers, we can represent a rigid motion by the pair $(R, p) \in SO(3) \times \mathbb{R}^3$. We define $SE(3) = SO(3) \times \mathbb{R}^3$ to be the set of all rigid motions and note that $SE(3)$ is a manifold of dimension 6 as well as a group. It may be verified that $SE(3)$ is a Lie group.

The configuration of a rigid body with respect to some reference configuration is described by an element $g \in SE(3)$. g acting on a point attached to the body defines the new location of the point relative to its reference configuration.

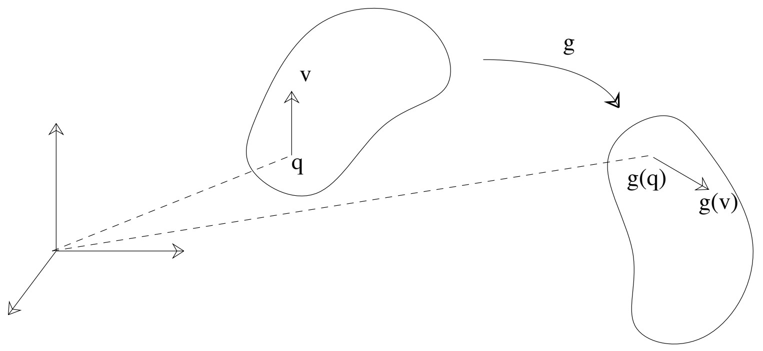


Figure 2.1: Rigid motion

If $q \in \mathbb{R}^3$ is a point on the body relative to some base (world) reference frame, then the location of q with respect to that basis after the body undergoes a rigid motion g is

$$g(q) = Rq + p$$

where R and p are represented in the same basis as q. This action is shown pictorially in Figure 2.1. We refer to the absolute coordinates as the *world* or *base coordinates* and the coordinates of a point on the object relative to the reference configuration as the *body coordinates*.

An object trajectory is described by a time parameterized curve, $g(t) \in SE(3)$. The velocity of an object is a tangent vector at g, so $\dot{g} \in T_g SE(3)$. $\dot{g}$ also acts on points in $\mathbb{R}^3$, giving a velocity vector $\dot{g}(q) \in \mathbb{R}^3$. Since $SE(3)$ is a Lie group, we can associate each element of $T_g SE(3)$ with the Lie algebra $se(3) \approx T_e SE(3)$ where e is the identity element. An element $\xi \in se(3)$ can be represented as a skew symmetric matrix, $\mathcal{S} \in so(3)$ and a vector $v \in \mathbb{R}^3$. Furthermore, any skew symmetric matrix has the form:

$$\mathcal{S} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

and hence we will often write $\mathcal{S}(\omega) \in so(3)$ to be the skew symmetric matrix associated with $\omega \in \mathbb{R}^3$. Note that $\mathcal{S}(\omega)q = \omega \times q$.

There are two ways to map $T_g SE(3)$ to $T_e SE(3)$ — left and right translation. The usual method is to use left translation, $L_{g^{-1}}$, where $L_g h = g \circ h$. The tangent map of $L_{g^{-1}}$ maps $T_g SE(3)$ to $T_e SE(3)$ and when applied to $\dot{g}$, the resulting map, $T_{g^{-1}}(L_{g^{-1}})\dot{g}$, takes a point in body coordinates to the velocity in body coordinates. For our purposes it is more natural to use the velocity of the point in world coordinates. This can be accomplished by using right translation and the resulting map takes a point in world coordinates to a velocity in world coordinates. Formally, we define the generalized velocity, $\xi \in T_eSE(3)$, in terms of $\dot{g} \in T_gSE(3)$ as

$$\xi = \dot{g}g^{-1} \tag{2.1}$$

The generalized velocity ξ is also called a *twist.*

Elements of $SE(3)$ can be represented as 4×4 matrices, referred to as *homogeneous coordinates.* If $g \in SE(3)$ we write

$$g = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix}$$

A point $q \in \mathbb{R}^3$ can be represented as a vector in $\mathbb{R}^4$ by defining $\tilde{q} = (q, 1) \in \mathbb{R}^3 \times \mathbb{R}$. Using this representation, $g(q)$ becomes matrix multiplication

$$g\tilde{q} = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \begin{pmatrix} q \\ 1 \end{pmatrix} = \begin{pmatrix} Rq + p \\ 1 \end{pmatrix}$$

To simplify notation we shall usually refer to $\tilde{q}$ simply as q.

The generalized velocity of a motion, in world coordinates, is

$$\xi = \dot{g}g^{-1} = \begin{bmatrix} \dot{R}R^T & \dot{p} - \dot{R}R^T p \\ 0 & 0 \end{bmatrix}$$

which can be rewritten as

$$\xi = \begin{bmatrix} \mathcal{S}(\omega) & v \\ 0 & 0 \end{bmatrix}$$

where $\omega \in \mathbb{R}^3$, $v \in \mathbb{R}^3$ and $\mathcal{S}(\omega)$ is the skew symmetric matrix generated by ω. The vector

$$\hat{\xi} = \begin{pmatrix} v \\ \omega \end{pmatrix}$$

is referred to as the *twist coordinates* of ξ and represents the rotational and linear velocity of an object as viewed in world coordinates. We will omit the hat when the usage is clear from context.

2.2 Fixed contact kinematics

Traditionally, a *fixed contact* between a finger and an object is described as a mapping between forces exerted by the finger at the point of contact and the resultant forces at some reference point on the object (e.g., the center of mass). We represent the force exerted at the i^{th} contact as $\tilde{F}_{c_i} = (\tilde{f}_{c_i}, \tilde{\tau}_{c_i}) \in \mathbb{R}^6$ where $\tilde{f}_{c_i}$ is the force exerted by contact and $\tilde{\tau}_{c_i}$ is the moment. The relationship between contact force and object force has the form

$$F_o = \begin{pmatrix} f_o \\ \tau_o \end{pmatrix} = \begin{pmatrix} \tilde{f}_{c_i} \\ \tilde{\tau}_{c_i} + r_{c_i} \times \tilde{f}_{c_i} \end{pmatrix} = \begin{bmatrix} I & 0 \\ \mathcal{S}(r_{c_i}) & I \end{bmatrix} \tilde{F}_{c_i}$$

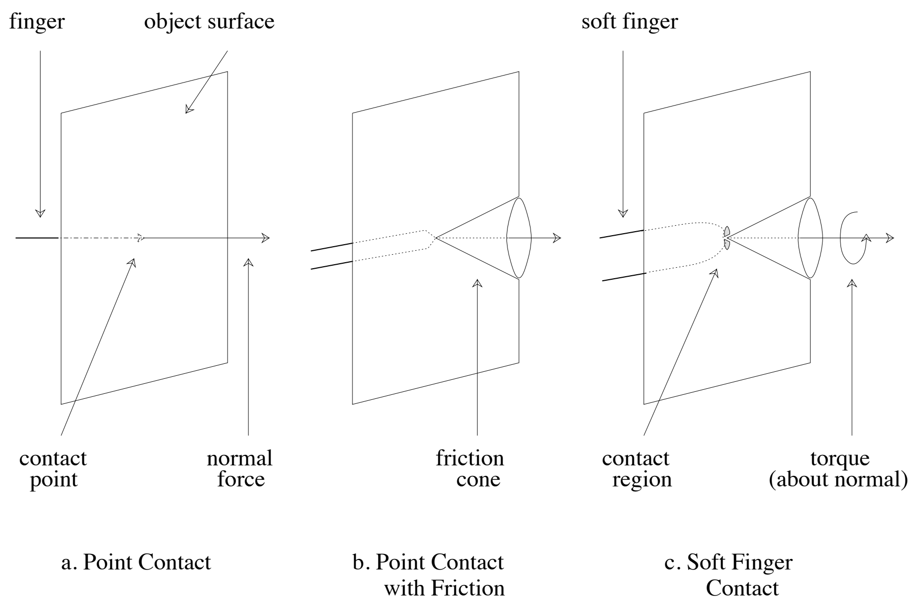


Figure 2.2: Contact types (from [70])

where $r_{c_i} \in \mathbb{R}^3$ is the vector between the object reference point and the contact.

Typically, a finger will not be able to exert forces in every direction; several simple contact models are used to classify common contact configurations. A *point contact* is obtained when there is no friction between the fingertip and the object. In this case, forces can only be applied in the direction normal to the surface of the object and hence we can represent the applied force as

$$\tilde{F}_{c_i} = \begin{bmatrix} n_{c_i} \\ 0 \end{bmatrix} f_{c_i} \tag{2.2}$$

where n_{c_i} is the unit vector normal to the object and $f_{c_i} \in \mathbb{R}$ is the amount of force applied by the finger in that direction.

A *point contact with friction* model is used when friction exists between the fingertip and the object, in which case forces can be exerted in any direction that is within a cone of forces about the direction of the surface normal. This cone, called the *friction cone*, is determined by the coefficient of friction. Figure 2.2b shows a point contact with friction and the resultant friction cone. This model assumes that moments cannot be applied (i.e., there is no torsional friction about the surface normal). As before, we represent the force felt by the object with respect to a basis of directions which are consistent with the friction model:

$$\tilde{F}_{c_i} = \begin{bmatrix} I \\ 0 \end{bmatrix} f_{c_i} \tag{2.3}$$

with $f_{c_i} \in \mathbb{R}^3$.

A more realistic contact model is the *soft finger* contact. Here we allow not only forces to be applied in a cone about the surface normal but also torques about that normal (see Figure 2.2c). These torques are limited by the torsional friction coefficient. Inside the relevant friction cones, this contact can be described as

$$\tilde{F}_{c_i} = \begin{bmatrix} I & 0 \\ 0 & n_{c_i} \end{bmatrix} \begin{pmatrix} f_{c_i} \\ \tau_{c_i} \end{pmatrix} \tag{2.4}$$

where $f_{c_i} \in \mathbb{R}^3$ and $\tau_{c_i} \in \mathbb{R}$.

Matrices mapping finger forces to contact force as in equations (2.2), (2.3) and (2.4) are referred to as *selection matrices* and we denote them by $B_i(x_o) \in \mathbb{R}^{6\times m_i}$, where m_i is the dimension of the range of forces and moments that can be applied for a given contact type. Note their dependence on the (fixed) contact point and the orientation of the object. Each of the contact types thus can be represented as a linear map $G_i(r_{c_i}, x_o)\colon F_{c_i} \in \mathbb{R}^{m_i} \mapsto F_o$

$$G_i(r_{c_i}, x_0) = \begin{bmatrix} I & 0 \\ \mathcal{S}(r_{c_i}) & I \end{bmatrix} B_i(x_0)$$

Since r_{c_i} is a function of the object orientation, we shall usually write $G_i(r_{c_i}, x_o)$ as $G_i(x_o)$.

If we have several fingers contacting an object then the net force on the object is the sum of the forces due to each finger. The map between finger forces and the total object force is called the *grasp map*, $G\colon \mathbb{R}^m \to \mathbb{R}^6$. Since each contact map is linear and forces can be superposed, we can add the individual contact maps to form G:

$$F_o = \begin{bmatrix} G_1 & \cdots & G_k \end{bmatrix} \begin{pmatrix} F_{c_1} \\ \vdots \\ F_{c_k} \end{pmatrix} = G F_c, \qquad \begin{array}{l} F_o \in \mathbb{R}^6 \\ F_c \in \mathbb{R}^{m_1} \times \mathbb{R}^{m_2} \times \ldots \times \mathbb{R}^{m_k} \end{array} \tag{2.5}$$

The null space of the grasp map corresponds to finger forces which cause no net force to be exerted on the object. We call the force on the object resulting from

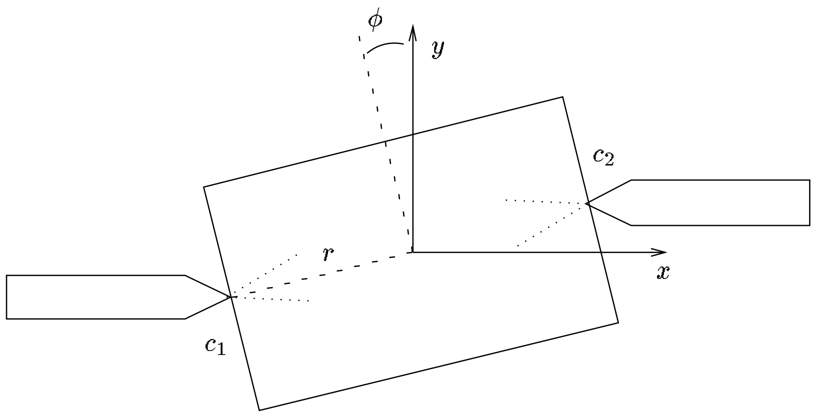


Figure 2.3: Planar two fingered grasp

finger forces which lie in the null space of G, denoted $\mathcal{N}(G)$, *internal* or *null forces*. It is in part these internal forces which allow us to grip or squeeze an object.

Dual to the representation of contacts as applied force and torque, one may also represent a contact as a constraint between the relative velocity of the object and the finger. Letting v_{c_i} and ω_{c_i} represent the linear and angular velocity of the contact point and v_o and ω_o represent the object velocity,

$$\begin{pmatrix} \tilde{v}_{c_i} \\ \tilde{\omega}_{c_i} \end{pmatrix} = \begin{bmatrix} I & \mathcal{S}(r_{c_i}) \\ 0 & I \end{bmatrix} \begin{pmatrix} v_o \\ \omega_o \end{pmatrix}$$

As before, the contact type determines which velocities are actually constrained; unconstrained directions can experience sliding or rolling. If we define v_c to be the velocities conjugate to F_c, the forces exerted by the fingers, it follows that

$$\begin{pmatrix} v_c \\ \omega_c \end{pmatrix} = G^T \begin{pmatrix} v_o \\ \omega_o \end{pmatrix}$$

This relationship between object velocity and finger velocities can also be derived in a more general setting using the principle of virtual work.

Example

Consider a simple two-fingered planar hand as shown in Figure 2.3. Since we are in the plane, the grasp matrix maps finger forces into x and y forces, and a torque perpendicular to the xy plane. If we assume that the contacts are point contacts with friction,

$$G_i(x, y, \phi) = \begin{bmatrix} I & 0 \\ \mathcal{S}\begin{pmatrix} \pm r\cos\phi \\ \pm r\sin\phi \\ 0 \end{pmatrix} & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

and the planar map for Figure 2.3, restricted to the plane, is

$$G(x, y, \phi) = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ r\sin(\phi) & -r\cos(\phi) \end{bmatrix}}_{G_1}\underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -r\sin(\phi) & r\cos(\phi) \end{bmatrix}}_{G_2} \tag{2.6}$$

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

Equation (2.6) shows that x and y forces from the fingers cause the same x and y forces to be exerted on the object as well as a torque that is dependent on the orientation of the object. The null space of this map is spanned by the vector

$$\begin{pmatrix} \cos\phi \\ \sin\phi \\ -\cos\phi \\ -\sin\phi \end{pmatrix}$$

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

2.3 Rolling contact kinematics

Most real world grasping situations involve moving rather than fixed contacts. Human fingers and many robotic fingers are actually surfaces and manipulation of an object by a set of fingers involves rolling of the fingers along the object surface. In this section we derive the kinematic equations for one object rolling against another.

Consider two objects, S_o and S_f in $\mathbb{R}^3$ which are touching at a point. We will restrict ourselves to the case where motion is contained in a single coordinate chart for each object. Let (c_o, U_o) and (c_f, U_f) be charts for the two surfaces and $\alpha_o = (u_o, v_o) \in U_o$ and $\alpha_f = (u_f, v_f) \in U_f$ be local coordinates. We will assume that c_o and c_f are orthogonal representations of the surface.[1] Furthermore, we let ψ represent the relative orientation of the tangent planes at the point of contact (see Figure 2.4). We call $\eta = (\alpha_o, \alpha_f, \psi)$ the *contact coordinates.*

[1]A surface representation $c\colon (u, v) \to R^3$ is orthogonal if $\frac{\partial c}{\partial u}$ and $\frac{\partial c}{\partial v}$ are orthogonal. Such a representation can always be constructed for a regular surface in a given coordinate chart.

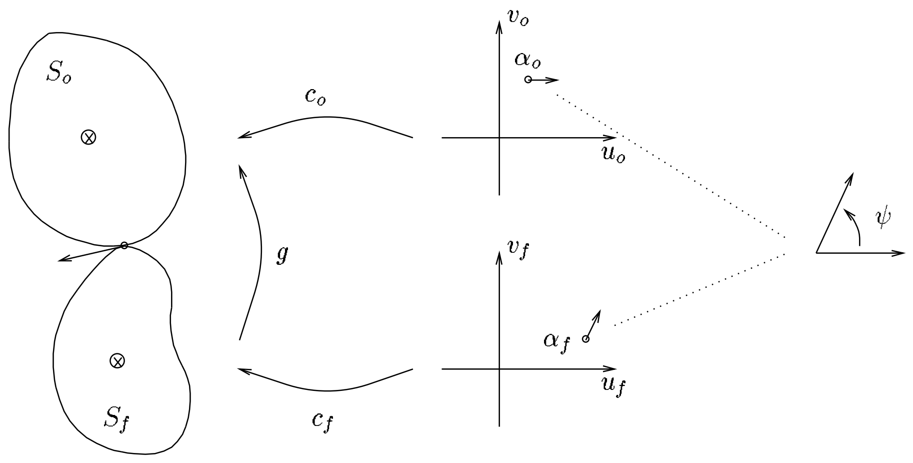


Figure 2.4: Parameterization of rolling contacts

Let $g \in SE(3)$ describe the relative position and orientation of S_f with respect to S_o. We wish to study the relationship between g and the local contact coordinates. To do so we assume that $g \in W \subset SE(3)$ where W is the set of all relative positions for which the two objects remain in contact.

We begin by writing the algebraic equations that η must satisfy. At any point of contact the location of the contact in space must agree for both objects

$$g \circ c_f(\alpha_f) = c_o(\alpha_o) \tag{2.7}$$

Furthermore, the tangent planes must coincide and hence the outward surface normals $n_o : S_o \to S^2 \subset \mathbb{R}^3$ and $n_f : S_f \to S^2 \subset \mathbb{R}^3$ must also agree. Letting $R \in SO(3)$ be the rotational component of g

$$Rn_f(\alpha_f) = -n_o(\alpha_o) \tag{2.8}$$

Finally, we define the angle between the tangent planes as the unique angle $\psi \in [0, 2\pi)$ such that

$$R\frac{\partial c_f}{\partial \alpha_f} M_f^{-1} R_\psi = \frac{\partial c_o}{\partial \alpha_o} M_o^{-1} \tag{2.9}$$

where

$$M_i = \begin{bmatrix} \|\frac{\partial c_i}{\partial u_i}\| & 0 \\ 0 & \|\frac{\partial c_i}{\partial v_i}\| \end{bmatrix}$$

insures that the columns of $\frac{\partial c}{\partial \alpha}$ are unit length and

$$R_\psi = \begin{bmatrix} \cos\psi & -\sin\psi \\ -\sin\psi & -\cos\psi \end{bmatrix}$$

converts α_o coordinates to the equivalent α_f coordinates at the point of contact. Since the normals are in opposite directions, R_ψ acts by negating the y coordinate and rotating by an angle ψ. Note that $R_\psi = R_\psi^T = R_\psi^{-1}$.

Theorem 2.1 *There is a smooth local bijection between η and $g \subset W$ if and only if*

$$\frac{\partial n_o}{\partial \alpha_o} M_o^{-1} + R \frac{\partial n_f}{\partial \alpha_f} M_f^{-1} R_\psi$$

is full rank

Proof. Functionally, equations (2.7) through (2.9) are of the form $h(g, \eta) = 0$. It is therefore sufficient (and necessary) to show that $\frac{\partial h}{\partial \eta}$ spans the allowable velocity space, TW. Since ψ can be defined directly, we omit the ψ coordinate and consider the dependence on $\alpha = (\alpha_o, \alpha_f)$,

$$h(g, \alpha) = \begin{pmatrix} c_o(\alpha_o) - g c_f(\alpha_f) \\ n_o(\alpha_o) + R n_f(\alpha_f) \end{pmatrix} \tag{2.10}$$

$$\frac{\partial h}{\partial \alpha}(g, \alpha) = \begin{pmatrix} \frac{\partial c_o(\alpha_o)}{\partial \alpha_o} & -R\frac{\partial c_f(\alpha_f)}{\partial \alpha_f} \\ \frac{\partial n_o(\alpha_o)}{\partial \alpha_o} & R\frac{\partial n_f(\alpha_f)}{\partial \alpha_f} \end{pmatrix} \tag{2.11}$$

First we show that the span of the rows of $\frac{\partial h}{\partial \alpha}$ does not contain either $(n_o, 0)$ or $(0, n_o)$, corresponding to translation and rotation about n_o. $(0, n_o)$ is spanned directly by $d\psi$ and $(n_o, 0)$ should not belong to the row span of $\frac{\partial h}{\partial \alpha}$ because motion in the n_o direction is not contained in TW. Since the range of $\frac{\partial c_o}{\partial \alpha_o}$ and $\frac{\partial c_f}{\partial \alpha_f}$ define the tangent plane and the n_i's have unit magnitude and using equation (2.8), we find

$$\begin{pmatrix} n_o \\ 0 \end{pmatrix}^T \frac{\partial h}{\partial \alpha} = \begin{bmatrix} n_o^T \frac{\partial c_o}{\partial \alpha_o} & n_f^T R^T R \frac{\partial c_f}{\partial \alpha_f} \end{bmatrix} = 0$$

$$\begin{pmatrix} 0 \\ n_o \end{pmatrix}^T \frac{\partial h}{\partial \alpha} = \begin{bmatrix} n_o^T \frac{\partial n_o}{\partial \alpha_o} & n_f^T R^T R \frac{\partial n_f}{\partial \alpha_f} \end{bmatrix} = 0$$

Next we examine the conditions under which $\frac{\partial h}{\partial \alpha}$ loses rank. Plugging equation (2.9) into equation (2.11), $\frac{\partial h}{\partial \alpha}$ can only lose rank when $\frac{\partial c_f}{\partial \alpha_f} = M_f^{-1} R_\psi M_o \frac{\partial c_o}{\partial \alpha_o}$, so $\frac{\partial h}{\partial \alpha}$ is full rank if and only if

$$\frac{\partial n_o}{\partial \alpha_o} M_o^{-1} + R \frac{\partial n_f}{\partial \alpha_f} M_f^{-1} R_\psi \tag{2.12}$$

is full rank. □

Proceeding along the lines of the proof given above, the differential relationship between η and g can be derived (see appendix at end of this chapter). It is convenient to make use of the *normalized Gauss frame* defined on each surface

$$\begin{bmatrix} x_i & y_i & z_i \end{bmatrix} = \begin{bmatrix} \frac{\partial c_i}{\partial \alpha_i} M_i^{-1} & n_i \end{bmatrix}$$

If we do not allow the fingers to slide on the object (soft finger contacts) then the motion of the contacts, $\dot{\eta}$, as a function of the relative motion, (ω, v), is given by

$$\begin{aligned} \dot{\alpha}_o &= M_o^{-1}(K_o + \tilde{K}_f)^{-1}\omega_t \\ \dot{\alpha}_f &= M_f^{-1}R_\psi(K_o + \tilde{K}_f)^{-1}\omega_t \\ \dot{\psi} &= T_o M_o \dot{\alpha}_o + T_f M_f \dot{\alpha}_f \end{aligned} \tag{2.13}$$

where

$$\begin{aligned} \omega_t &= \begin{bmatrix} x_o^T \\ y_o^T \end{bmatrix} n_o \times \omega \\ K_o &= \begin{bmatrix} x_o^T \\ y_o^T \end{bmatrix} \frac{\partial n_o}{\partial \alpha_o} M_o^{-1} \\ \tilde{K}_f &= R_\psi \begin{bmatrix} x_f^T \\ y_f^T \end{bmatrix} \frac{\partial n_f}{\partial \alpha_f} M_f^{-1} R_\psi \\ T_o &= y_o^T \frac{\partial x_o}{\partial \alpha_o} M_o^{-1} \\ T_f &= y_f^T \frac{\partial x_f}{\partial \alpha_f} M_f^{-1} \end{aligned}$$

$(K_o + \tilde{K}_f)$ is called the *relative curvature* [68]. From equations (2.9) and (2.12) we see that the relative curvature is invertible precisely when $\frac{\partial h}{\partial \eta}$ is onto TW. We shall assume that all manipulation occurs in an open set on which the relative curvature is invertible.

We can now describe the kinematics for rolling contact—the relationship between the object velocities and a set of finger velocities. This situation is identical to that given for fixed contacts except that the vector r_{c_i} between the object reference frame and the i^{th} contact point is now a function of η as well as the object orientation. But η is a continuous function of $g = x_o^{-1}x_{f_i}$ so we have

$$F_o = G(x_o, x_f)F_c$$

where $x_f = (x_{f_f}, \cdots, x_{f_k})$ is the position and orientation of the fingers and $F_c \in \mathbb{R}^{m_1} \times \cdots \times \mathbb{R}^{m_k}$ is the force exerted by the fingers at the contact point.

As before, G is composed of matrices of the form

$$G_i(x_o, x_f) = \begin{bmatrix} I & 0 \\ \mathcal{S}(r_{c_i}) & I \end{bmatrix} B_i(x_o, x_f)$$

The velocity relationship can again be derived from the principle of virtual work or algebraically to determine

$$\begin{pmatrix} v_c \\ \omega_c \end{pmatrix} = G^T(x_o, x_f) \begin{pmatrix} v_o \\ \omega_o \end{pmatrix} \tag{2.14}$$

Examples

To illustrate the form of the contact equations, we consider two examples—a sphere rolling on a plane and a sphere rolling on another sphere [59]. The local coordinates of the plane are chosen to be $c_o(u, v) = (u, v, 0)$. The sphere requires multiple coordinate charts to describe the entire surface, so we shall restrict ourselves to the chart

$$c_f(u, v) = (\rho \cos u \cos v, -\rho \cos u \sin v, \rho \sin u)$$

where ρ is the radius of the sphere and $-\pi/2 < u < \pi/2$, $-\pi < v < \pi$. The curvature, torsion and metric tensors are easily calculated to be

$$\begin{aligned} K_o &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & K_f &= \begin{bmatrix} 1/\rho & 0 \\ 0 & 1/\rho \end{bmatrix} \\ M_o &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & M_f &= \begin{bmatrix} \rho & 0 \\ 0 & \rho \cos u \end{bmatrix} \\ T_o &= \begin{bmatrix} 0 & 0 \end{bmatrix} & T_f &= \begin{bmatrix} 0 & -1/\rho \tan u \end{bmatrix} \end{aligned}$$

Consider first a spherical finger of radius ρ rolling on a plane. The equations governing the evolution of the contact point are

$$\begin{aligned} \dot{u}_f &= \omega_1 \\ \dot{v}_f &= \sec u_f \, \omega_2 \\ \dot{u}_o &= \rho \cos \psi \, \omega_1 - \rho \sin \psi \, \omega_2 \\ \dot{v}_o &= -\rho \sin \psi \, \omega_1 - \rho \cos \psi \, \omega_2 \\ \dot{\psi} &= -\tan u_f \, \omega_2 \end{aligned} \tag{2.15}$$

where $\omega_t = (\omega_1, \omega_2)$. If our object is a sphere of unit radius instead of a plane, the contact equations become

$$\begin{aligned} \dot{u}_f &= \tfrac{1}{1+\rho} \omega_1 \\ \dot{v}_f &= \tfrac{1}{1+\rho} \sec u_f \, \omega_2 \\ \dot{u}_o &= \tfrac{\rho}{1+\rho} \cos \psi \, \omega_1 - \tfrac{\rho}{1+\rho} \sin \psi \, \omega_2 \\ \dot{v}_o &= -\tfrac{\rho}{1+\rho} \sin \psi \sec u_o \, \omega_1 - \tfrac{\rho}{1+\rho} \cos \psi \sec u_o \, \omega_2 \\ \dot{\psi} &= \tfrac{\rho}{1+\rho} \sin \psi \tan u_o \, \omega_1 + \tfrac{\rho}{1+\rho} (\cos \psi \tan u_o - 1/\rho \, \tan u_f) \, \omega_2 \end{aligned} \tag{2.16}$$

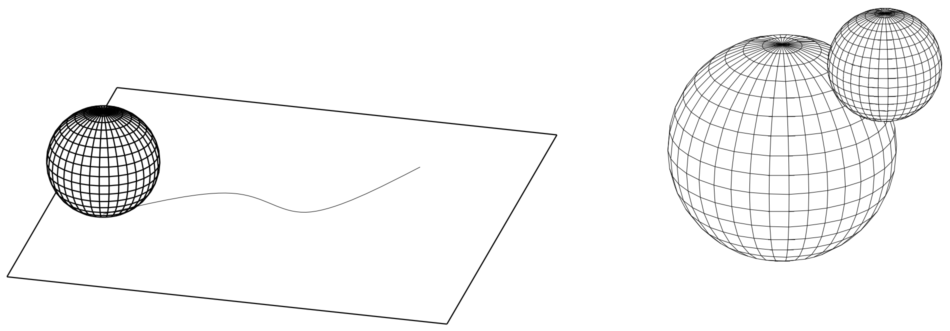


Figure 2.5: Spherical finger rolling on a plane and on another sphere. The finger is only allowed to roll on the object and not slip or twist.

2.4 Finger Kinematics

Up to this point we have assumed that the fingers which contact our object are rigid bodies which can move freely in space. We are more interested in the case when the fingers are kinematic mechanisms. We model each finger as an open kinematic chain with n_i degrees of freedom. Using the product of exponentials formulation of the fingers kinematics, we can then describe the constraints between the object velocity and the joint velocities of the fingers.

The product of exponentials formula was introduced by Brockett [12] as an elegant means of describing the kinematics of open kinematic chains with a single degree of freedom at each joint. Using this formulation allows us to derive the kinematic equations which relate the position of the rigid bodies which compose the robot to the configuration variables which specify the positions of the joints. We briefly review the formalism here; a more complete treatment can be found in Paden's thesis [76].

Consider the motion of a point p which is rotated about a fixed axis in space, as shown in Figure 2.6a. Let $\omega \in \mathbb{R}^3$ be the direction of the axis and $q \in \mathbb{R}^3$ be any point on the axis. If the link connecting the particle to the axis has unit rotational velocity, then the velocity of the particle is given by

$$\dot{p}(t) = \omega \times (p - q) = \begin{bmatrix} \mathcal{S}(\omega) & -\omega \times q \\ 0 & 0 \end{bmatrix} \begin{pmatrix} p \\ 1 \end{pmatrix} =: \xi(p) \tag{2.17}$$

where $\xi \in se(3)$ is a twist which represents the axis of rotation. Equation (2.17) is a differential equation which represents the motion of a particle p which is attached to the link.

Suppose now that we wish to determine where the point p moves if we rotate the link about the axis ξ by an angle θ. One way to accomplish this motion is to move at unit speed and integrate equation (2.17) for time θ. Since (2.17) is

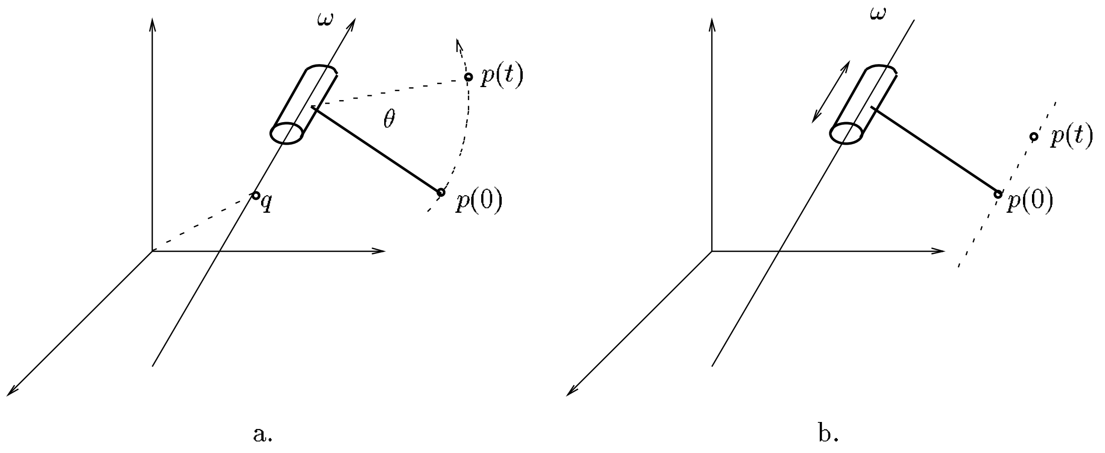


Figure 2.6: Revolute and prismatic joints

a linear equation, the result is

$$p(\theta) = e^{\hat{\xi}\theta} p(0)$$

This formula is the basis for the product of exponentials formula. It gives the location of a point attached to the link after the link has been rotated by θ. A joint which causes a link to rotate about a fixed axis is called a *revolute joint.*

Next we consider a joint which moves a link *along* an axis, called a *prismatic* joint (see Figure 2.6b). This motion can also be modeled as a twist:

$$\dot{p}(t) = \dot{\theta} \begin{bmatrix} 0 & \omega \\ 0 & 0 \end{bmatrix} \begin{pmatrix} p \\ 1 \end{pmatrix}$$

and again $p(\theta) = e^{\hat{\xi}\theta} p(0)$, where θ represents the amount of translation in some convenient set of units. The motion is much simpler in this case, consisting of rectilinear motion in the direction of the axis.

A large number of robot manipulators can be modeled as series chains of revolute and prismatic joints. Such a robot consists of a set of rigid links which are connected to each other by revolute and prismatic joints. Joints may be placed at the same location in space to model a joint with more than one degree of freedom. An example of such a mechanism is a spherical wrist, which can rotate in any direction and is modeled by three revolute joints with mutually perpendicular axes intersecting at a single point.

We are now in a position to calculate the location of a point attached to a robot as the joints move through specified angles. For concreteness, consider a point attached to the end of an n link robot, as shown in Figure 2.7. We begin by choosing a *zero configuration* for the robot and writing down the constant

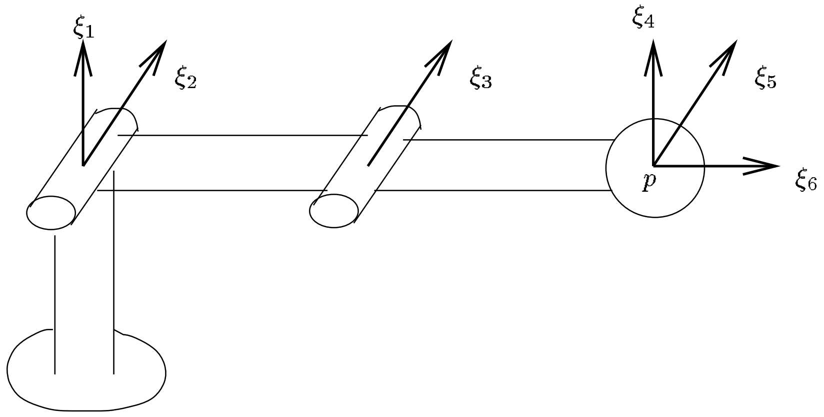


Figure 2.7: Elbow manipulator

screws, ξ_i, corresponding to each joint. Let $\theta \in \mathbb{R}^n$ be the final joint angles of the robot. If we move any single joint of the robot, the particle moves to a new location $e^{\theta_i \xi_i} p$. By moving the links starting from the end of the manipulator and moving towards the base, we can compose the rigid motions and obtain the final location of the particle:

$$p(\theta) = e^{\theta_1 \xi_1} \cdots e^{\theta_{n-1} \xi_{n-1}} e^{\theta_n \xi_n} p$$

This formula for $p(\theta)$ is called the product of exponentials formula.

More generally, we have defined a map $K(\theta) : SE(3) \to SE(3)$ which describes the kinematics of the mechanism:

$$K(\theta) = e^{\theta_1 \xi_1} \cdots e^{\theta_n \xi_n}$$

This map tells us how points (or a frame) attached to the end effector of a robot varies with the joint angles, θ. It is clear from its definition that $K(\theta)$ is a smooth function of θ.

The Jacobian of the forward kinematic map relates joint velocities to the end effector velocities,

$$\begin{pmatrix} \mathcal{S}(\omega) & v \\ 0 & 0 \end{pmatrix} = \dot{K} \circ K^{-1} = DK(\theta)\dot{\theta} \circ K^{-1} \qquad \begin{array}{l} \theta \in \mathbb{R}^n \\ (v, \omega) \in \mathbb{R}^6 \end{array}$$

We can make this formula more explicit using the product of exponentials notation:

$$\dot{K} = \sum_{i=1}^{n} \dot{\theta}_i \left(e^{\theta_1 \xi_1} \cdots e^{\theta_{i-1}\xi_{i-1}}\right) \xi_i \left(e^{\theta_i \xi_i} \cdots e^{\theta_n \xi_n}\right)$$
$$\dot{K}K^{-1} = \sum_{i=1}^{n} \dot{\theta}_i \left(e^{\theta_1 \xi_1} \cdots e^{\theta_{i-1}\xi_{i-1}}\right) \xi_i \left(e^{-\theta_{i-1}\xi_{i-1}} \cdots e^{-\theta_1 \xi_1}\right)$$

This map is linear in $\dot{\theta}$. If we replace the conjugated ξ_i's by $\tilde{\xi}_i$ and use twist coordinates (suppressing the hat), we have

$$\begin{pmatrix} v_{f_i} \\ \omega_{f_i} \end{pmatrix} = [\ \xi_1 \quad \tilde{\xi}_2 \quad \cdots \quad \tilde{\xi}_n\ ]\dot{\theta}_{f_i} =: J_{f_i}(\theta_{f_i})\dot{\theta}_{f_i}$$

$J_{f_i}(\theta_{f_i}) \in \mathbb{R}^{6\times n}$ is the *finger Jacobian* for the i^{th} finger.

Combining this with the velocity transformation between the finger location and the contact location (a function of $x_o^{-1}x_f$) we write the *contact Jacobian* as

$$J_{c_i}(x_o, \theta_{f_i}) = \begin{pmatrix} I & \mathcal{S}(r_{c_i}) \\ 0 & I \end{pmatrix} J_{f_i} \qquad J_{c_i} : \dot{\theta}_{f_i} \mapsto \begin{pmatrix} v_{c_i} \\ \omega_{c_i} \end{pmatrix} \tag{2.18}$$

As with the fixed contacts, fingers are only allowed to exert forces in certain directions depending on the contact type. This is equivalent to saying that finger motions are only constrained in certain directions; these directions are given by the column span of $B_i^T(x_o, \theta_{f_i})$: $\mathbb{R}^{m_i} \to \mathbb{R}^6$ (where B_i is the selection matrix defined in section 2.2). Combining this with the grasp map for the i^{th} finger, we obtain the velocity constraint due to the i^{th} contact,

$$G_i^T(x_o, \theta_{f_i}) \begin{pmatrix} v_o \\ \omega_o \end{pmatrix} = B_i^T(x_o, \theta_{f_i}) J_{c_i}(x_o, \theta_{f_i}) \dot{\theta}_{f_i}$$

We now stack these matrices and write the grasp constraint for the hand as

$$\begin{bmatrix} G_1^T \\ \vdots \\ G_k^T \end{bmatrix} \begin{pmatrix} v_o \\ \omega_o \end{pmatrix} = \begin{bmatrix} B_1^T J_{c_1} & & 0 \\ & \ddots & \\ 0 & & B_k^T J_{c_k} \end{bmatrix} \dot{\theta}$$

$$G^T(x_o, \theta) \begin{pmatrix} v_o \\ \omega_o \end{pmatrix} = J(x_o, \theta)\dot{\theta} \tag{2.19}$$

Equations (2.19) describes the constraints between the object and finger velocities. Roughly speaking, it equates the contact velocities of the fingers and the object. As in previous cases, this velocity constraint induces a relationship between joint forces and object forces. A diagram which summarizes the transformations between frames is shown in Figure 2.8.

$$\begin{tikzcd}
 & \texttt{robot} & \texttt{contact} & \texttt{object} \\
\texttt{velocity} & \dot{\theta} \arrow[r, "J"] \arrow[d] & (v_c, \omega_c) & (v_o, \omega_o) \arrow[l, "G^T"'] \arrow[ll, bend right, "\mathcal{A}"] \arrow[d] \\
\texttt{position} & \theta \arrow[rr, dashed, "h"] & & x_o \\
\texttt{force} & \tau \arrow[u] \arrow[rr, bend right, "\mathcal{A}^T"] & F_c \arrow[l, "J^T"'] \arrow[r, "G"] & F_o \arrow[u]
\end{tikzcd}$$

Figure 2.8: Diagram relating different robot coordinate frames. In this figure, θ represents the configuration of the robot, x_o the configuration of the object. The velocity constraints imposed by the grasp are specified in contact coordinates. The dashed line connecting θ and x_o represents the fact that x_o is determined by θ only if the contact constraint is holonomic.

2.5 Grasp stability and manipulability

In the preceding analysis, we have modeled a contact constraint as a *bidirectional* constraint. Using such a constraint, a finger can both push and pull on an object. In reality, a finger is only allowed to apply unidirectional forces. A further complication is sliding—if the ratio of the tangent forces to the normal force is too great, a finger may slide along an object, violating the constraint. By choosing a grasp carefully, we can often circumvent these restrictions.

For contact models involving friction, we must insure that all contact forces lie within the friction cone determined by the coefficient of friction. The set of all forces lying in or on the friction cone is

$$FC = \left\{ f_c \in \mathbb{R}^n : ||f^t_{c_{ij}}|| \leq \mu_{ij} ||f^n_{c_i}||, \quad i = 1, \ldots, k, \quad j = 1, \ldots, m_i \right\} \tag{2.20}$$

where $f^t_{c_{ij}}$ is the tangent component of the j^{th} element of f_{c_i}, $f^n_{c_i}$ is the normal force for the i^{th} contact, and μ_{ij} is the coefficient of friction corresponding to $f_{c_{ij}}$. For soft finger contacts, the torques exerted by the fingers also satisfy equation (2.20) with $f^t_{c_{ij}}$ replaced by the torque (i.e., we do not want to apply a torque which is greater than the torsional friction coefficient multiplied by the magnitude of the normal force).

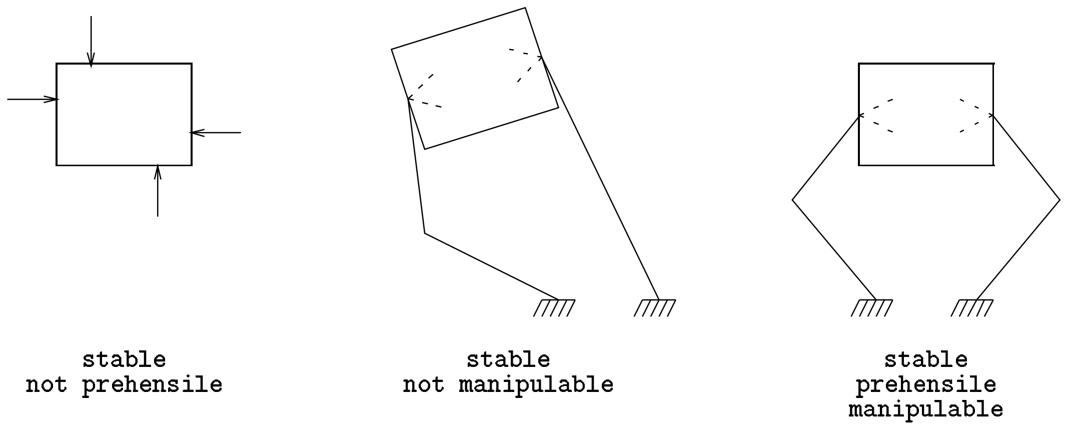


Figure 2.9: Examples of stable, prehensile, and manipulable grasps

We say a grasp on an object is *stable* if we can resist, through a set of contacts, arbitrary forces and torques on the object. This requires that the image of the grasp map over the set of forces in the friction cone span the space of forces and torques on the object, that is $G_{x,\theta}(FC) = \mathbb{R}^6$ for all x, θ. Note that this is a condition only on the contact kinematics and not the finger kinematics. A stable grasp is also called a *force closure* grasp [73].

A grasp is *prehensile* [21] if there exists a force contained in the null space of the grasp map which also lies in the *interior* of the friction cone. More formally, $\mathcal{N}(G) \cap \overset{\circ}{FC} \neq \{\}$ where $\overset{\circ}{FC}$ is the set of forces lying completely within the friction cone (i.e., $\|f^t_{c_{ij}}\| < \mu_{ij}\|f^n_{c_i}\|$). We shall require this property in order to insure that our controllers can maintain a grip on an object while manipulating it. A prehensile grasp guarantees that we can exert any force on the object while remaining *inside* the friction cone.

If a grasp contains no frictionless contacts and G is full rank, then stable and prehensile grasps are equivalent. A proof of this fact can be found in the thesis of Cole [20] using the mathematical formulation of Mishra, Schwartz, and Sharir [67]. If a grasp contains frictionless contacts then $\overset{\circ}{FC}$ is necessarily empty since the condition $\|f^t_{c_{ij}}\| < 0$ can never be satisfied. A example of a grasp which is stable but not prehensile is shown in Figure 2.9.

A grasp is said to be *manipulable* if arbitrary motions of the object can be accommodated by the fingers. Unlike stability, manipulability is a property of both the contact and finger kinematics. Since the range of motion of the contacts is given locally by the range of the hand Jacobian, the condition can be written as $\mathcal{R}(G^T) \subseteq \mathcal{R}(J)$, for all x, θ. It is clearly necessary that a grasp be manipulable if we are interested in unconstrained object motion. An example of a point at which a grasp loses manipulability is shown in Figure 2.9. Motion which results in the right-hand contact moving in the singular direction of the right finger is not possible, hence we loose manipulability at that point. Unlike the previous properties, manipulability does not depend on the friction cone.

We shall generally assume that a grasp has been chosen which is stable, manipulable and prehensile in some operating region. Methods for find such grasps have been explored by Nguyen [74], Li and Sastry [64], and Mishra, Schwartz, and Sharir [67] and the references therein.

2.6 Summary

In this chapter we have derived the kinematic equations for fixed and rolling contacts. The fundamental kinematic relationships are the grasp kinematics

$$G^T(x_o,\theta)\dot{x}_o = J(x_o,\theta)\dot{\theta} \tag{2.21}$$

and, for rolling contacts, the contact kinematics

$$\begin{aligned}\dot{\alpha}_o &= (K_o+\tilde{K}_f)^{-1}\omega_t \\ \dot{\alpha}_f &= R_\psi(K_o+\tilde{K}_f)^{-1}\omega_t \\ \dot{\psi} &= T_o\dot{\alpha}_o + T_f\dot{\alpha}_f\end{aligned} \tag{2.22}$$

The rolling kinematics are useful because it is easier to calculate G and J using $\eta = (\alpha_o, \alpha_f, \psi)$ rather than θ and x_o alone.

A grasp is said to be stable when finger forces lying in the friction cone span the space of object forces

$$\mathbb{R}^6 = G(FC) \tag{2.23}$$

manipulable when arbitrary motions can be generated by the fingers

$$\mathcal{R}(G^T) \subseteq \mathcal{R}(J) \tag{2.24}$$

and prehensile when

$$\mathcal{N}(G) \cap \overset{\circ}{FC} \neq \{\} \tag{2.25}$$

In the case that G is full rank and $FC \neq \{\}$ (i.e., no frictionless contacts), stability and prehensility are equivalent.

Appendix I – Contact kinematics derivation

In this appendix we derive the kinematics of contact for two objects touching each other at a point. The notation is described more fully in Section 2.3. An alternate derivation can be found in a recent paper by Montana [68].

To derive the kinematics, we begin with constraint equations given by equating the points of contact, normals of contact and tangent planes at the contact points:

$$\begin{aligned} Rc_f(\alpha_f) + p &= c_o(\alpha_o) && (2.26)\\ Rn_f(\alpha_f) &= -n_o(\alpha_o) && (2.27)\\ R\frac{\partial c_f}{\partial \alpha_f} M_f^{-1} R_\psi &= \frac{\partial c_o}{\partial \alpha_o} M_o^{-1} && (2.28)\end{aligned}$$

Differentiate (2.26) and (2.27)

$$\begin{aligned} \dot{R}c_f + R\frac{\partial c_f}{\partial \alpha_f}\dot{\alpha}_f + \dot{p} &= \frac{\partial c_o}{\partial \alpha_o}\dot{\alpha}_o && (2.29)\\ \dot{R}n_f + R\frac{\partial n_f}{\partial \alpha_f}\dot{\alpha}_f &= -\frac{\partial n_o}{\partial \alpha_o}\dot{\alpha}_o && (2.30)\end{aligned}$$

Multiply (2.29) by $\frac{\partial c_o}{\partial \alpha_o}^T$ and substitute $\dot{\alpha}_o$ into (2.30)

$$\dot{R}n_f + R\frac{\partial n_f}{\partial \alpha_f}\dot{\alpha}_f = -\frac{\partial n_o}{\partial \alpha_o} M_o^{-2} \frac{\partial c_o}{\partial \alpha_o}^T \left(\dot{R}c_f + R\frac{\partial c_f}{\partial \alpha_f}\dot{\alpha}_f + \dot{p}\right) \qquad (2.31)$$

Using (2.28) in the last term of (2.31) and rearranging

$$\begin{aligned} &\left(R\frac{\partial n_f}{\partial \alpha_f} + \frac{\partial n_o}{\partial \alpha_o} M_o^{-2} (\frac{\partial c_o}{\partial \alpha_o}^T \frac{\partial c_o}{\partial \alpha_o}) M_o^{-1} R_\psi M_f\right)\dot{\alpha}_f \\ &= -\dot{R}n_f - \frac{\partial n_o}{\partial \alpha_o} M_o^{-2} \frac{\partial c_o}{\partial \alpha_o}^T (\dot{R}c_f + \dot{p}) && (2.32)\end{aligned}$$

Simplify the first term and multiply by $M_o^{-T}\frac{\partial c_o}{\partial \alpha_o}^T$ on the left

$$\begin{aligned} &M_o^{-T}\frac{\partial c_o}{\partial \alpha_o}^T \left(R\frac{\partial n_f}{\partial \alpha_f} + \frac{\partial n_o}{\partial \alpha_o} M_o^{-1} R_\psi M_f\right)\\ &= M_o^{-T}\frac{\partial c_o}{\partial \alpha_o}^T \left(R\frac{\partial n_f}{\partial \alpha_f} M_f^{-1} R_\psi + \frac{\partial n_o}{\partial \alpha_o} M_o^{-1}\right) R_\psi M_f\\ &= \left(\underbrace{R_\psi M_f^{-T}\frac{\partial c_f}{\partial \alpha_f}^T \frac{\partial n_f}{\partial \alpha_f} M_f^{-1} R_\psi}_{\tilde{K}_f} + \underbrace{M_o^{-T}\frac{\partial c_o}{\partial \alpha_o}^T \frac{\partial n_o}{\partial \alpha_o} M_o^{-1}}_{K_o}\right) R_\psi M_f \end{aligned}$$

Multiply both sides of (2.32) by $M_o^{-T}\frac{\partial c_o}{\partial \alpha_o}^T$ and use the previous calculation

$$\begin{aligned}
\dot{\alpha}_f &= M_f^{-1}R_\psi\left(\tilde{K}_f + K_o\right)^{-1}\cdot \\
&\qquad M_o^{-T}\frac{\partial c_o}{\partial \alpha_o}^T\left(-\dot{R}n_f - \frac{\partial n_o}{\partial \alpha_o}M_o^{-2}\frac{\partial c_o}{\partial \alpha_o}^T(\dot{R}c_f + \dot{p})\right) \\
&= M_f^{-1}R_\psi\left(\tilde{K}_f + K_o\right)^{-1}\cdot \\
&\qquad \left(-M_o^{-T}\frac{\partial c_o}{\partial \alpha_o}^T\dot{R}n_f - K_oM_o^{-T}\frac{\partial c_o}{\partial \alpha_o}^T(\dot{R}c_f + \dot{p})\right)
\end{aligned}$$

Let ω_t stand for the $\dot{R}n_f$ term and v_t represent the $\dot{R}c_f + \dot{p}$ term:

$$\dot{\alpha}_f = M_f^{-1}R_\psi\left(\tilde{K}_f + K_o\right)^{-1}(w_t - K_ov_t) \qquad (2.33)$$

Now w_t and v_t can now be calculated in terms of the relative velocity given by $(\mathcal{S}(\omega), v) = \dot{g}g^{-1}$. We use the fact that $\mathcal{S}(\omega)a = \omega \times a$ and $\omega \times a = -a \times \omega$ to obtain

$$\begin{aligned}
w_t &= -M_o^{-T}\frac{\partial c_o}{\partial \alpha_o}^T(\omega \times (Rn_f)) = -M_o^{-T}\frac{\partial c_o}{\partial \alpha_o}^T(n_o \times \omega) \\
v_t &= M_o^{-T}\frac{\partial c_o}{\partial \alpha_o}^T(\omega \times (Rc_f) + \omega \times p + v) \\
&= M_o^{-T}\frac{\partial c_o}{\partial \alpha_o}^T(\omega \times (c_o - p) + \omega \times p + v) \\
&= M_o^{-T}\frac{\partial c_o}{\partial \alpha_o}^T(-c_o \times \omega + v)
\end{aligned}$$

We see that ω_t is the relative rotational velocity projected onto the tangent plane at the contact. It includes only terms due to *rolling* since rotation normal to the surface is annihilated by taking the cross product with n_o. Likewise, v_t is the relative linear velocity between the contacts, projected onto the tangent plane, i.e., the *sliding* velocity.

A similar calculation yields

$$\dot{\alpha}_o = M_o^{-1}\left(\tilde{K}_f + K_o\right)^{-1}\left(w_t - \tilde{K}_fv_t\right) \qquad (2.34)$$

which gives the kinematics for the object contact point in local coordinates.

Next we solve for ψ, the angle between the tangent planes of the finger and object. Combining (2.27) and (2.28) we can write

$$R\begin{bmatrix} \frac{\partial c_f}{\partial \alpha_f}M_f^{-1} & n_f \end{bmatrix}\begin{bmatrix} R_\psi & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} \frac{\partial c_o}{\partial \alpha_o}M_o^{-1} & n_o \end{bmatrix}$$

and using the normalized Gaussian coordinates this can be rewritten

$$R[x_f\, y_f\, z_f]\bar{R}_\psi = [x_o\, y_o\, z_o] \tag{2.35}$$

Take the derivative of (2.35)

$$\dot{R}[x_f\, y_f\, z_f]\bar{R}_\psi + R[\dot{x}_f\, \dot{y}_f\, \dot{z}_f]\bar{R}_\psi + R[x_f\, y_f\, z_f]\begin{bmatrix} \dot{\bar{R}}_\psi & 0 \\ 0 & 0 \end{bmatrix} = [\dot{x}_o\, \dot{y}_o\, \dot{z}_o]$$

Premultiply by $y_f^T R^T$

$$y_f^T R^T \dot{R}[x_f\, y_f\, z_f]\bar{R}_\psi + y_f^T[\dot{x}_f\, \dot{y}_f\, \dot{z}_f]\bar{R}_\psi + (0\,1\,0)\begin{bmatrix} \dot{\bar{R}}_\psi 0 \\ 0\; 0 \end{bmatrix} = y_f^T R^T[\dot{x}_o\, \dot{y}_o\, \dot{z}_o]$$

Postmultiply by $\bar{R}_\psi\begin{pmatrix}1\\0\\0\end{pmatrix}$ and note $\bar{R}_\psi \bar{R}_\psi = I$

$$y_f^T R^T \dot{R} x_f + y_f^T \dot{x}_f + (0\,1\,0)\begin{bmatrix} \dot{\bar{R}}_\psi \bar{R}_\psi & 0 \\ 0 & 0 \end{bmatrix}\begin{pmatrix}1\\0\\0\end{pmatrix} = y_f^T R^T[\dot{x}_o\, \dot{y}_o\, \dot{z}_o]\bar{R}_\psi\begin{pmatrix}1\\0\\0\end{pmatrix}$$

$$y_f^T R^T \dot{R} x_f + y_f^T \dot{x}_f + (0\,1)\begin{bmatrix} 0 & \dot{\psi} \\ -\dot{\psi} & 0 \end{bmatrix}\begin{pmatrix}1\\0\end{pmatrix} = y_f^T R^T[\dot{x}_o\, \dot{y}_o\, \dot{z}_o]\bar{R}_\psi\begin{pmatrix}1\\0\\0\end{pmatrix}$$

$$y_f^T R^T \dot{R} x_f + y_f^T \dot{x}_f - \dot{\psi} = y_f^T R^T[\dot{x}_o\, \dot{y}_o\, \dot{z}_o]\bar{R}_\psi\begin{pmatrix}1\\0\\0\end{pmatrix}$$

From (2.35) we see that $y_f^T R^T = (0\,1\,0)\bar{R}_\psi\begin{bmatrix} x_o^T \\ y_o^T \\ z_o^T \end{bmatrix} = (0\,1)R_\psi\begin{bmatrix} x_o^T \\ y_o^T \end{bmatrix}$ and so

$$\dot{\psi} = y_f^T R^T \dot{R} x_f + y_f^T \frac{\partial x_f}{\partial \alpha_f}\dot{\alpha}_f - (0\,1)R_\psi\begin{bmatrix} x_o^T \dot{x}_o & x_o^T \dot{y}_o \\ y_o^T \dot{x}_o & y_o^T \dot{y}_o \end{bmatrix}R_\psi\begin{pmatrix}1\\0\end{pmatrix} \tag{2.36}$$

Using the following identities

$$x_i^T y_i = 0 \quad\Rightarrow\quad \dot{x}_i^T y_i = -x_i^T \dot{y}_i = y_i^T \dot{x}_i$$

$$x_i^T x_i = 1 \quad\Rightarrow\quad \dot{x}_i^T x_i = 0$$

(2.36) can be written as

$$\begin{aligned} \dot{\psi} &= y_f^T R^T \dot{R} x_f + y_o^T \frac{\partial x_o}{\partial \alpha_o}\dot{\alpha}_o + y_f^T \frac{\partial x_f}{\partial \alpha_f}\dot{\alpha}_f \\ &= \omega_n + T_o M_o \dot{\alpha}_o + T_f M_f \dot{\alpha}_f \end{aligned} \tag{2.37}$$

where

$$\begin{aligned} \omega_n &= y_f^T R^T \dot{R} x_f = (R y_f)^T \omega \times (R x_f) \\ &= (R z_f)^T \omega = z_o^T \omega \end{aligned}$$

and the last equality follows from the vector formula $a \cdot b \times c = b \cdot c \times a$. This last equation shows that ω_n is just the relative rotational velocity projected onto the surface normal.

Collecting equations (2.33), (2.34) and (2.37) we have

$$\begin{aligned}
\dot{\alpha}_o &= M_o^{-1}(K_o + \tilde{K}_f)^{-1}\left(\omega_t - \tilde{K}_f v_t\right) \\
\dot{\alpha}_f &= M_f^{-1} R_\psi (K_o + \tilde{K}_f)^{-1}(\omega_t - K_o v_t) \\
\dot{\psi} &= \omega_n + T_o M_o \dot{\alpha}_o + T_f M_f \dot{\alpha}_f
\end{aligned}$$

The matrix $K_o + \tilde{K}_f$ is called the *relative curvature* by Montana [68].

Chapter 3

Dynamics and Control of Constrained Manipulator Systems

In this chapter we review the formulation of robot dynamics and extend those results to include robot systems with contact constraints. The primary result is that even for relatively complicated robot systems, the equations of motion can be written in standard form. This point of view has been used by Khatib in his operational space formulation [49]; recent extensions [50] can be used to cover special, but important, cases of the results presented here.

3.1 Introduction

There are several methods for generating the dynamic equations of an open kinematic chain. All of these methods generate the same set of equations, but the form of the equations may be better suited for computation or analysis depending on the formulation. We will rely on Lagrangian analysis for our derivation.

Lagrange's equations work well for for constrained systems since they make explicit use of allowable displacements, usually called *virtual displacements*. Define the Lagrangian as

$$L(q, \dot{q}) = T(q, \dot{q}) - V(q)$$

where q is a set of coordinates for the mechanism, T is the kinetic energy, and V the potential energy. Let δq represent the infinitesimal displacements which satisfy the constraints on the system. Then the motion of the system satisfies

$$\left(\frac{d}{dt} \frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} - Q \right) \delta q = 0$$

where L is the Lagrangian and Q is a vector of external forces conjugate to q. If q is a set of generalized coordinates (i.e., all configurations of the system are parameterized by q and q is unconstrained) then δq is free and Lagrange's equations become

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = Q$$

This is the usual form of Lagrange's equations when applied to robots and $q = \theta$. We derive a more explicit formulation using this method in later sections.

History of robot control

A control law for a robot manipulator is a rule which assigns joint torques at a given time such that a given task is accomplished. The most common control task is trajectory tracking—given a desired joint trajectory $\theta_d(\cdot)$, design a control law such that the actual robot position tracks the desired position, $\lim_{t\to\infty} \theta_d(t) - \theta(t) = 0$. Such a task is an integral part of any higher level robot task such as assembly, painting, or welding. The task itself is usually described in terms of a desired trajectory of the end-effector rather than the joint angles of the robot.

One of the simplest approaches to controlling the end-effector location is using joint interpolation. By using the inverse kinematics of the manipulator, an end-effector trajectory can be converted to a joint trajectory. Due to the computational complexity of inverse kinematics algorithms, this calculation is typically performed at a only a few points. The commanded trajectory consists of straight line motions (in joint space) between interpolation points. The resulting end-effector motion consists of curvilinear segments which match the desired end-effector trajectory at the interpolation points. The main feature of controllers using this approach is that the control law calculation occurs in the joint space of the manipulator. Examples of joint space controllers can be found in standard texts on robot control [29, 89].

The next generation of controllers improved performance by measuring errors in end-effector coordinates rather than joint coordinates. There are several advantages to such an approach. The inverse kinematics no longer have to be calculated since the control algorithm instead uses the computationally simpler forward kinematics to map the current position into end-effector coordinates. Furthermore, motion between points consists of straight line segments in end-effector space (with properly defined controllers). For many tasks this is highly desirable. Controllers which were originally proposed in joint space were extended to operate in end-effector coordinates [66].

In the late 1980's, researchers began experimenting with coordinated robot systems, such as a multi-fingered hand. These systems presented new challenges since the motion of individual robots was constrained. This led to yet another round of control laws, many bearing strong resemblances to joint and end-effector based control laws. A variety of techniques were developed, including master/slave algorithms [91, 95], Cartesian (object) space algorithms [21, 60] and many others (see for example [37, 71]).

Overview of results

The goal of this chapter is to develop a systematic method for writing control laws for systems of robots. To accomplish this, we begin with a systematic development of the dynamics of such systems. In particular, we are interested in writing the dynamic equations of a set of robot manipulators interacting with each other via a set of contact constraints. This class of systems includes classical robot manipulators which are not in contact with anything, robots which have limited degrees of freedom due to contact constraints with a fixed environment, and robots interacting with each other through a grasped object, such as the case with multi-fingered robot hands.

By writing the equations of motion in a consistent form, we are able to derive control laws which work for the entire class of robot systems. This allows us to simultaneously prove convergence properties for control laws which operate in joint, end-effector (Cartesian), or object coordinates. In proving stability, we use both the form and structure of the dynamic equations.

On the surface, the results presented seem to be straightforward consequence of Lagrangian nature of robot dynamics. The different spaces in which the control law calculation is carried out naively correspond to different choices of generalized coordinates for the configuration space of the manipulator. However, in may important situations, the coordinates in which the control law operates is *not* a valid set of generalized coordinates. In particular, for the case of rolling contact between a finger and an object, it is possible that there are no minimal set of generalized coordinates for the system (due to the nonholonomic nature of the system).

The first half of this chapter presents a complete derivation of the dynamics of an open chain robot manipulator. We then extend that derivation to cover more general robot systems with contact constraints such as the rolling contact constraints derived in the previous chapter. Although the notation will reflect our prejudice towards control of multifingered hands, the results are more general, holding for any robot system with constraints which can be written in a certain way. In particular, we show how these constraints can be used to model changes of coordinates and contact with the environment.

The second half of the chapter uses the dynamics formulation to derive some common control laws for robot manipulators. In particular, we derive and analyze a control law which generalizes some of the more commonly used control laws. As previously noted, the control law can be applied to a wide variety of robot systems using the same proof of stability. Experimental results are presented to show the performance of the control law in a contact situation.

3.2 Simple robot dynamics

The Lagrangian for an open chain robot

To apply Lagrange's equations, we must calculate the kinetic and potential energy of the robot links as a function of the joint angles and velocities. This in turn requires that we have a model for the mass distribution of the links. Since each link is a rigid body, its kinetic and potential energy can be be defined in terms of its total mass and its inertia about the center of mass.

Let $V \subset \mathbb{R}^3$ be the volume occupied by a rigid body and $\rho(r)$, $r = (x, y, z) \in \mathbb{R}^3$, be the mass distribution of the body. If the object is made from a homogeneous material then $\rho(r) = \rho$, a constant. The mass of the body is the volume integral of the mass density:

$$m = \int_V \rho(r) dV$$

The center of mass is the weighted average of the density

$$\bar{r} = \frac{\int_V \rho(r) r dV}{m}$$

By choosing coordinates $r' = r - \bar{r}$ we can place the center of mass at the origin of the new coordinate system. This coordinate system is referred to as *center of mass coordinates.*

The inertia of a rigid body is more complicated. It models the kinetic energy due to rotation and is represented by a matrix $\mathcal{I} \in \mathbb{R}^{3\times 3}$. The kinetic energy for a body rotating about its center of mass with body angular velocity ω_b is $\omega_b^T \mathcal{I} \omega_b$. The components of $\mathcal{I}$ in center of mass coordinates (denoted $r = (x_1, x_2, x_3)$ for simplicity) are given by

$$\begin{aligned} \mathcal{I}_{ii} &= \int_V \rho(r)(r^2 - x_i^2) dV \qquad & i = 1, 2, 3 \\ \mathcal{I}_{ij} &= \int_V \rho(r) x_i x_j dV & i \neq j \end{aligned}$$

$\mathcal{I}$ is symmetric and positive definite.

We now wish to write the total kinetic energy of a moving rigid body. Let $g = (R, p) \in SE(3)$ be the configuration of the center of mass of the body and $\lambda = (v, \omega)$ its velocity in *world* coordinates. The kinetic energy is the sum of the translational and rotational components

$$K(g, \lambda) = \frac{1}{2} m v^T v + \frac{1}{2} \omega^T R^T \mathcal{I} R \omega$$

We will write

$$K(g,\lambda) := \frac{1}{2}\lambda^T \mathcal{M}(g)\lambda := \frac{1}{2}\begin{pmatrix} v \\ \omega \end{pmatrix}^T \begin{bmatrix} mI & 0 \\ 0 & R^T \mathcal{I} R \end{bmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix}$$

The matrix $\mathcal{M}(g)$ is the *inertia matrix* for a rigid body. $\mathcal{M}(g)$ is symmetric and positive definite for all $g \in SE(3)$.

To calculate the kinetic energy of the entire robot manipulator, we sum the kinetic energy of each link. Using the forward kinematics, we can write an expression for K_i, the kinetic energy of the i^{th} link, in terms of the robot configuration and joint velocities. Let $g_i(\theta)$ be the configuration of the i^{th} link and $\lambda_i(\theta, \dot{\theta}) = (v_i, \omega_i) = J_i(\theta)\dot{\theta}$ be the corresponding velocity. If $\bar{r}_i$ is the location of the center of mass of the i^{th} link relative to the origin of that link, then the velocity of the center of mass is given by

$$\bar{\lambda}_i = \begin{pmatrix} v_i + \omega_i \times \bar{r}_i \\ \omega_i \end{pmatrix} =: \begin{pmatrix} \bar{v}_i \\ \omega_i \end{pmatrix}$$

Since v_i and ω_i are linear in $\dot{\theta}$, $\bar{\lambda}_i$ is also linear in $\dot{\theta}$ and hence we can write the kinetic energy as

$$K_i(\theta, \dot{\theta}) = \frac{1}{2}\bar{\lambda}_i^T \mathcal{M}_i(\theta)\bar{\lambda}_i = \frac{1}{2}\dot{\theta}^T \bar{J}_i^T(\theta)\mathcal{M}_i(\theta)\bar{J}_i(\theta)\dot{\theta}$$

where $\bar{\lambda}_i =: \bar{J}_i(\theta)\dot{\theta}$. Now the total kinetic energy can be written as

$$K(\theta, \dot{\theta}) = \sum_{i=1}^{n} K_i(\theta, \dot{\theta}) =: \frac{1}{2}\dot{\theta}^T M(\theta)\dot{\theta} \tag{3.1}$$

To complete our derivation of the Lagrangian, we must calculate the potential energy of the manipulator. Let $\bar{h}_i(\theta)$ be the height of the center of mass of the i^{th} link (height is the component of the position of the center of mass in the direction of gravity). The potential energy for the i^{th} link is

$$V_i(\theta) = m_i g \bar{h}(\theta)$$

where m_i is the mass of the i^{th} link and g is the gravitational constant. Combining this with the kinetic energy, we have

$$\begin{aligned} L(\theta, \dot{\theta}) &= \sum_{i=1}^{n} \Big(K_i(\theta, \dot{\theta}) - V_i(\theta) \Big) \\ &= \tfrac{1}{2}\dot{\theta}^T M(\theta)\dot{\theta} - V(\theta) \end{aligned}$$

Equations of motion for an open chain manipulator

Let $\theta \in \mathbb{R}^n$ be the joint angles for the manipulator and $\tau \in \mathbb{R}^n$ be the corresponding joint torques. The Lagrangian is of the form

$$L(\theta, \dot{\theta}) = \frac{1}{2}\dot{\theta}^T M(\theta)\dot{\theta} - V(\theta)$$

where $M(\theta)$ is the inertia matrix for the manipulator and $V(\theta)$ is the potential energy due to gravity. Substituting into Lagrange's equations

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} - \tau = 0$$

and letting τ represent the actuator torques (and other non-conservative forces), we obtain

$$M_{ij}(\theta)\ddot{\theta}_j + \frac{\partial M_{ij}(\theta)}{\partial \theta_k}\dot{\theta}_j\dot{\theta}_k - \frac{1}{2}\frac{\partial M_{jk}(\theta)}{\partial \theta_i}\dot{\theta}_j\dot{\theta}_k + \frac{\partial V}{\partial \theta_i}(\theta) = \tau_i$$

where summation over repeated indices is assumed. To put this in a more conventional form we define the matrix $C(\theta, \dot{\theta})$ as

$$C_{ij}(\theta, \dot{\theta}) = \frac{1}{2}\frac{\partial M_{ij}(\theta)}{\partial \theta_k}\dot{\theta}_k + \frac{1}{2}\frac{\partial M_{ik}(\theta)}{\partial \theta_j}\dot{\theta}_k - \frac{1}{2}\frac{\partial M_{jk}(\theta)}{\partial \theta_i}\dot{\theta}_k \tag{3.2}$$

and write

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + N(\theta, \dot{\theta}) = \tau \tag{3.3}$$

where $N(\theta, \dot{\theta})$ includes gravity terms and other forces (such as friction) which act at the joints. This equation has several structural properties:

Theorem 3.1 *Equation (3.3) has the following properties*

1. *$M(\theta)$ is symmetric and positive definite*
2. *$\dot{M} - 2C \in \mathbb{R}^{n \times n}$ is a skew symmetric matrix.*

Proof. (1) From equation (3.1) and the definition of K_i,

$$M(\theta) = \sum_{i=1}^{n} \bar{J}_i^T \mathcal{M}_i(\theta)\bar{J}_i$$

where $\mathcal{M}_i(\theta)$ is the inertia matrix of the i^{th} link and is therefore positive definite. Expanding the kinetic energy

$$\begin{aligned}
\dot{\theta}^T M(\theta)\dot{\theta} &= \sum_{i=1}^{n} \dot{\theta}^T \bar{J}_i^T \mathcal{M}_i \bar{J}_i \dot{\theta} \\
&= \sum_{i=1}^{n} \begin{pmatrix} \bar{v}_i \\ \omega_i \end{pmatrix}^T \mathcal{M}_i \begin{pmatrix} \bar{v}_i \\ \omega_i \end{pmatrix}
\end{aligned}$$

Each term in the summation is ≥ 0 and hence $M(\theta) \geq 0$. Equality is only achieved when $(v_i, \omega_i) = 0$ for all i and this is possible only if $\dot{\theta} = 0$. Thus $M(\theta) > 0$.

For (2) we calculate the matrix $\dot{M} - 2C$:

$$\begin{aligned}
(\dot{M} - 2C)_{ij} &= \dot{M}_{ij}(\theta) - 2C_{ij}(\theta) \\
&= \tfrac{\partial M_{ij}}{\partial \theta_k}\dot{\theta}_k - \tfrac{\partial M_{ij}}{\partial \theta_k}\dot{\theta}_k - \tfrac{\partial M_{ik}}{\partial \theta_j}\dot{\theta}_k + \tfrac{\partial M_{jk}}{\partial \theta_i}\dot{\theta}_k \\
&= \tfrac{\partial M_{jk}}{\partial \theta_i}\dot{\theta}_k - \tfrac{\partial M_{ik}}{\partial \theta_j}\dot{\theta}_k
\end{aligned}$$

Switching i and j shows $(\dot{M} - 2C)^T = -(\dot{M} - 2C)$. □

3.3 Robot hand dynamics

We now examine the dynamics of a set of fingers actuated at each joint connected through a set of contacts to a rigid body. The finger dynamics can be written as

$$M_f(\theta)\ddot{\theta} + C_f(\theta, \dot{\theta})\dot{\theta} + N_f(\theta, \dot{\theta}) = \tau - J^T f_c \tag{3.4}$$

where $\theta \in \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_k}$ is now the set of joint angles for *all* of the robots and τ is the corresponding set of torques. f_c is the vector of contact forces. The object dynamics are given by the Newton-Euler equations

$$\begin{bmatrix} m_o I & 0 \\ 0 & \mathcal{I}_o \end{bmatrix} \begin{pmatrix} \dot{v}_o \\ \dot{\omega}_o \end{pmatrix} + \begin{pmatrix} 0 \\ \omega_o \times \mathcal{I}_o \omega_o \end{pmatrix} + \begin{pmatrix} DV_o(x_o) \\ 0 \end{pmatrix} = G f_c$$

where $\mathcal{I}_o = R \mathcal{I} R^T$ is the object inertia in world coordinates and V_o is the potential energy. In local coordinates this has the same basic form as the robot dynamics, lacking only the actuator torques:

$$M_o(x)\ddot{x} + C_o(x, \dot{x})\dot{x} + N_o(x, \dot{x}) = G f_c \tag{3.5}$$

where x is a local parameterization of $x_o \in SE(3)$. To find the dynamics we can solve for f_c in equation (3.4) (assuming we stay away from finger singularities) and substitute the resulting expression into (3.5). f_c represents the forces which maintain the grasping constraint.

A slightly more elegant analysis can be obtained using the Lagrangian approach directly. The advantage of using this method is that it holds for arbitrary mechanical systems with equality constraints which are linear in the velocities. Consider two mechanical systems with decoupled dynamics of the form

$$\begin{aligned}
M_f(\theta)\ddot{\theta} + C_f(\theta, \dot{\theta})\dot{\theta} + N_f(\theta, \dot{\theta}) = \tau \\
M_o(x)\ddot{x} + C_o(x, \dot{x})\dot{x} + N_o(x, \dot{x}) = 0
\end{aligned}$$

We attach these two systems with a set of constraints

$$G^T(x, \theta)\dot{x} = J(x, \theta)\dot{\theta} \tag{3.6}$$

which represents the grasp. We will assume that the grasp is both stable and manipulable. For the moment we will also require J to be injective.

This velocity constraint generates a constraint on the virtual displacements $\delta\theta$ and δx, namely $\delta\theta = J^{-1}(q)G^T(q)\delta x$ with $q = (x, \theta)$. Using this relationship, we can write Lagrange's equations as

$$\begin{aligned}
\left(\tfrac{d}{dt}\tfrac{\partial L}{\partial \dot q} - \tfrac{\partial L}{\partial q} - (\tau, 0)\right)\delta q &= 0 \\
\begin{pmatrix} \tfrac{d}{dt}\tfrac{\partial L}{\partial \dot\theta} - \tfrac{\partial L}{\partial \theta} - \tau \\ \tfrac{d}{dt}\tfrac{\partial L}{\partial \dot x} - \tfrac{\partial L}{\partial x} \end{pmatrix} \cdot \begin{pmatrix} \delta\theta \\ \delta x \end{pmatrix} &= 0 \\
\left(\tfrac{d}{dt}\tfrac{\partial L}{\partial \dot\theta} - \tfrac{\partial L}{\partial \theta} - \tau\right)\delta\theta + \left(\tfrac{d}{dt}\tfrac{\partial L}{\partial \dot x} - \tfrac{\partial L}{\partial x}\right)\delta x &= 0 \\
GJ^{-T}\left(\tfrac{d}{dt}\tfrac{\partial L}{\partial \dot\theta} - \tfrac{\partial L}{\partial \theta} - \tau\right)\delta x + \left(\tfrac{d}{dt}\tfrac{\partial L}{\partial \dot x} - \tfrac{\partial L}{\partial x}\right)\delta x &= 0
\end{aligned}$$

and since δx is arbitrary

$$\frac{d}{dt}\frac{\partial L}{\partial \dot x} - \frac{\partial L}{\partial x} + GJ^{-T}\left(\frac{d}{dt}\frac{\partial L}{\partial \dot\theta} - \frac{\partial L}{\partial \theta}\right) = GJ^{-T}\tau \tag{3.7}$$

This equation together with the velocity constraint given in equation (3.6) describes the system completely. Note that equation (3.7) is a vector differential equation with $n - m$ rows and equation (3.6) is a vector equation with m rows.

It is tempting to derive equation (3.7) by using the velocity constraint directly in the kinetic energy equation (which is a function of $\dot\theta$ and $\dot x$) and then substituting this into Lagrange's equations. As noted in Rosenberg [81], section 14.2, this can only be done if the constraint is holonomic, i.e., θ can be written as a function of x.

Next we separate the kinetic energy into an object portion and a robot portion

$$T = \frac{1}{2}\dot\theta^T M_f(\theta)\dot\theta + \frac{1}{2}\dot x^T M_o(x)\dot x$$

Using equation (3.7) we find

$$\tilde M(q)\ddot x + \tilde C(q, \dot q)\dot x \;=\; GJ^{-T}\tau = F \tag{3.8}$$

where

$$\begin{aligned}
\tilde M &= M_o + GJ^{-T}M_f J^{-1}G^T \\
\tilde C &= C_o + GJ^{-T}\left(C_f J^{-1}G^T + M_f \tfrac{d}{dt}\left(J^{-1}G^T\right)\right)
\end{aligned}$$

and C_f and C_o are obtained from equation (3.2) by replacing M with M_f and M_o respectively. F is the applied force represented in the object's frame of reference.

Theorem 3.2 *Equation (3.8) has the following properties*

1. *$\tilde M$ is symmetric and positive definite for all q.*

2. $\tilde{M} - 2\tilde{C}$ is skew-symmetric.

Proof. Since the grasp is assumed to be stable and manipulable and J is assumed injective, (1) follows from its definition. To show (2),

$$\begin{aligned}\dot{\tilde{M}} - 2\tilde{C} &= (\dot{M}_o - 2C_o) + GJ^{-T}(\dot{M}_f - 2C_f)J^{-1}G^T \\ &\quad + \tfrac{d}{dt}\left(GJ^{-T}\right)M_f J^{-1}G^T - GJ^{-T}M_f \tfrac{d}{dt}\left(J^{-1}G^T\right)\end{aligned}$$

The first line is the sum of skew-symmetric pieces. Taking transposes and using symmetry of M_f inverts the sign of the last line and hence it too is skew-symmetric. □

Thus we have an equation with form and structure similar to the open chain robot. In the object frame of reference, $\tilde{M}$ is the effective mass of the object, and $\tilde{C}$ is the effective Coriolis and centrifugal matrix. These matrices include the dynamics of the fingers, which are being used to actually control the motion of the object. However the details of the finger kinematics and dynamics are effectively hidden in the definition of $\tilde{M}$ and $\tilde{C}$.

This simple result has important consequences in control. Typically robot controllers are designed by placing a feedback loop around the joint positions (and velocities) of the robot. The controller generates torques which attempt to make the robot follow a prescribed joint trajectory. This can lead to difficulty in grasping situations since the joint level controllers are often not aware of the constraints and therefore may violate them. However, since the grasping dynamics are of the same form as the dynamics of a single manipulator, we can just as easily write the control algorithm in object coordinates. An additional advantage of this approach is that controller objectives are often specified in terms of the object motion and hence it is easier to perform the controller design and analysis in that space.

Internal forces

Even though we will write our controllers in terms of F, it is actually the joint torques which we are able to specify. Given the desired force in constrained coordinates, we can apply that force using an actuator force of $J^T G^+ F$, where G^+ is a pseudo-inverse for G. In general G is not square and by examining the right side of the equations of motion (3.8) we note that if $J^{-T}\tau \in \mathcal{N}(G)$ then the net force in the object frame of reference is zero and hence forces of this form cause no net motion on the object. These forces are in fact the forces which act against the constraint and are generally termed *internal* or *constraint* forces. We can use these internal forces to satisfy other conditions, such as keeping the contact forces inside the friction cone (to avoid slipping) or varying the load distribution of a set of manipulators rigidly grasping an object.

To describe the internal forces more explicitly, we can extend the grasp map by defining an orthonormal matrix $H(\theta)$ whose rows form a basis for the null space of $G(\theta)$. As before we assume that $G(\theta)$ has constant rank and we break all forces up into an external and an internal piece, F_e and F_i. Given these desired forces, the torques that should be applied by the actuators is

$$\tau = J^T \begin{pmatrix} G \\ H \end{pmatrix}^{-1} \begin{pmatrix} F_e \\ F_i \end{pmatrix} = J^T \begin{pmatrix} G^+ & H^T \end{pmatrix} \begin{pmatrix} F_e \\ F_i \end{pmatrix} \tag{3.9}$$

Other robot systems

Although we have derived the dynamics in the context of grasping, the same derivation holds for a number of other interesting robot systems. If we let $\mathcal{F}: \mathbb{R}^n \to \mathbb{R}^n$ represent the forward kinematics of a manipulator (in local coordinates), then differentiating $x = \mathcal{F}(\theta)$ gives

$$D\mathcal{F}(\theta)\dot{\theta} = \dot{x}$$

This has the same form as our basic constraint with $G = I$. Defining $M_o = 0$ gives the proper dynamic equations for a robot in Cartesian coordinates:

$$\tilde{M}(q)\ddot{x} + \tilde{C}(q,\dot{q})\dot{x} = F \tag{3.10}$$

This formulation holds only away from singularities of the matrix $D\mathcal{F}(\theta)$ since the effective inertia matrix, $\tilde{M}$, is unbounded when $D\mathcal{F}$ is singular. This reflects the physical condition that if there is a kinematic singularity then the effective inertia in the direction of the singularity appears to be infinite (since we cannot move in that direction).

From an analytical framework, we can simplify (3.10) further. Since (3.10) only holds away from singularities of $\mathcal{F}$, we can assume that $\mathcal{F}$ is a diffeomorphism and write $x = \mathcal{F}^{-1}(\theta)$. This allows us to write $\tilde{M}(q)$ as $\tilde{M}(x)$ and hence (3.10) becomes a function only of x. This corresponds exactly to choosing x as a generalized set of coordinates for the system. Computationally, it is much more desirable to leave (3.10) in the given form. Since we typically measure the joint angles and use them to determine x, a great computation savings is achieved by evaluating $M(\theta)$ rather than $M(x) = M(\mathcal{F}^{-1}(\mathcal{F}(\theta)))$.

Another system which can be modeled using this framework is a robot following a surface. If we let α denote the local coordinates of a point on the surface, then our constraint is $\mathcal{F}(\theta) = \mathcal{G}(\alpha)$, where $\mathcal{G}(\alpha)$ is the world coordinates of the point α. Differentiating this system gives the desired velocity constraint. As in the previous example, setting $M_o = 0$ gives the equations of motion in terms of the surface position α. In the case of a rolling contact between the robot and the surface, it may be necessary to use the more general tools developed in Chapter 2.

3.4 Control

For a constrained robotic system, we can break the control problem into two parts: tracking the desired trajectory and maintaining a desired internal force.

In the context of grasping, this is a restatement of our desire to both maneuver the grasped object and maintain the finger forces inside of their respective friction cones. This latter goal is particularly important in the context of grasping since we assumed in our derivation of the grasp dynamics that contact was not broken.

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

To illustrate the control of robot systems, we look at two controllers which have appeared in the robotics literature and apply them to robotic grasping. We consider only grasps which are stable, manipulable and prehensile. We start by considering systems of the form

$$M(q)\ddot{x} + C(q,\dot{q})\dot{x} + N(q,\dot{q}) = F \tag{3.11}$$

where $M(q) \in \mathbb{R}^{n\times n}$ is a positive definite inertia matrix and $C(q,\dot{q})\dot{x}$ is the Coriolis and centrifugal force vector. The vector $N(q,\dot{q}) \in \mathbb{R}^n$ contains all friction and gravity terms and the vector $F \in \mathbb{R}^n$ represents generalized forces in the object coordinate frame. Given a desired object force F, we apply that force by commanding a set of joint torques

$$\tau = J^T G^+ F + J^T F_N \tag{3.12}$$

where J and G define the grasping constraint and $F_N \in \mathcal{N}(G)$.

As before, we note that although our analysis is being done in the context of grasping, the control laws derived here can be applied to a number of robotic systems. The proofs of stability rely only on the positive definiteness of $M(q)$ and the property that $\dot{M} - 2C$ is skew-symmetric. In particular, we can apply these control laws to robots using joint coordinates or end-effector coordinates (away from kinematic singularities). As we shall see in a subsequent section, the controllers can also be extended to robot systems with kinematic redundancies.

Computed torque

Computed torque is an exactly linearizing control law (i.e., the dynamics are rendered linear by state feedback) that has been used extensively in robotics research. It has been used for joint level control [5], Cartesian control [66], and most recently, control of multi-fingered hands [60, 21, 69].

Given a desired trajectory x_d we use the control

$$F = M(q)\,(\ddot{x}_d + K_v\dot{e} + K_p e) + C(q,\dot{q})\dot{x} + N(q,\dot{q}) \tag{3.13}$$

where $e = x_d - x$ and K_v and K_p are constant gain matrices. When substituted into equation (3.11), this gives the resulting error equation

$$M(q)\,(\ddot{e} + K_v\dot{e} + K_p e) = 0 \tag{3.14}$$

and since $M(q)$ is always positive definite we have

$$\ddot{e} + K_v\dot{e} + K_p e = 0 \tag{3.15}$$

This differential equation is linear and thus it is easy to choose K_v and K_p so that the overall system is stable and $e \rightarrow 0$ exponentially as $t \rightarrow \infty$. Moreover, since equation (3.15) is linear we can choose K_v and K_p such that we get independent exponentially stable systems (by choosing K_p and K_v diagonal).

The disadvantage of this control law is that it is not easy to specify the interaction with the environment. From the form of the error equation we might think that we could use K_p to model the stiffness of the system and exert forces by commanding trajectories which result in fixed errors. Unfortunately this does not work as can be seen by examining the force due to a quasi-static displacement Δq

$$\Delta F = M(q)K_p\Delta q \tag{3.16}$$

Since K_p must be constant, the resultant stiffness will vary with configuration. Additionally, given a desired stiffness matrix it may not be possible to find a positive definite K_p that achieves that stiffness (the product of two symmetric positive definite matrices is not necessarily positive definite nor symmetric).

PD controllers

A much simpler control law can be achieved by using linear feedback to attempt trajectory tracking. A control law of the form

$$F = K_v\dot{e} + K_p e$$

is called a PD controller since it contains proportional plus derivative feedback of the error. The convergence properties of this control law are weak: the most that has been shown is stabilization to a stationary point. In many settings, we can achieve trajectory tracking if the rate of convergence of the controller is sufficiently fast relative to the rate of change of the trajectory.

There have been many control laws which extend the basic PD control law to give provably asymptotic trajectory tracking for arbitrary trajectories. One of the earliest (and most elegant) examples of one of these control methods is the natural control law proposed by Koditschek [52]:

$$F = M(q)\ddot{x}_d + C(q,\dot{q})\dot{x}_d + N(q,\dot{q}) + K_v\dot{e} + K_p e \tag{3.17}$$

The first three terms in the control law provide a feedforward-like torque to move the manipulator along the desired trajectory.[1] The feedback terms, $K_v\dot{e} + K_p e$, are used to move the manipulator back onto the desired trajectory in the case of initial condition or sensing errors.

The proof of stability for the natural control law relies on the skew-symmetry of $\dot{M} - 2C$. This is the reason that the Coriolis matrix is multiplied by $\dot{x}_d$ instead of $\dot{x}$, as in the computed torque control law. We omit the proof of stability for this control law since it is substantially similar to the proof of a slightly more general case.

Consider the control law

$$F = M(q)\,(\ddot{x}_d + \lambda\dot{e}) + C(q,\dot{q})\,(\dot{x}_d + \lambda e) + N(q,\dot{q}) + K_p e + K_v \dot{e} \qquad (3.18)$$

where $\lambda > 0$ and K_p, K_v are symmetric positive definite. This control law is very similar to controllers which have appeared in the literature for joint-based control of robots [86, 82]. The additional damping term, not present in the natural control law, allows us to prove *exponential* trajectory tracking.

Theorem 3.3 *The control law (3.18) applied to the system (3.11) results in exponential trajectory tracking if* $\lambda, K_v, K_p > 0$.

Proof. We use a Lyapunov-based argument. The closed loop dynamics are

$$M(q)(\ddot{e} + \lambda\dot{e}) + C(q,\dot{q})(\dot{e} + \lambda e) + K_v\dot{e} + K_p e = 0$$

Choose the Lyapunov candidate

$$V = \frac{1}{2}(\dot{e} + \lambda e)^T M(q)(\dot{e} + \lambda e) + \frac{1}{2}e^T K_p e + \frac{1}{2}\lambda e^T K_v e$$

Since $M(q) > 0$, it follows that V is quadratic in e and $\dot{e}$:

$$\sigma_1 ||(e,\dot{e})||^2 \leq ||V(e,\dot{e})|| \leq \sigma_2 ||(e,\dot{e})||^2$$

Differentiating V along trajectories of the system yields

$$\begin{aligned} \dot{V} &= (\dot{e} + \lambda e)^T M(q)(\ddot{e} + \lambda\dot{e}) \\ &\quad + \tfrac{1}{2}(\dot{e} + \lambda e)^T \dot{M}(q)(\dot{e} + \lambda e) + \dot{e}^T K_p e + \lambda e^T K_v \dot{e} \\ &= (\dot{e} + \lambda e)^T(-C(q,\dot{q})(\dot{e} + \lambda e) - K_v\dot{e} - K_p e) \\ &\quad + \tfrac{1}{2}(\dot{e} + \lambda e)^T \dot{M}(q)(\dot{e} + \lambda e) + \dot{e}^T K_p e + \lambda e^T K_v \dot{e} \end{aligned}$$

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

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

Since $\dot{V}$ is quadratic in e and $\dot{e}$, $e \to 0$ exponentially as $t \to \infty$. □

[1]strictly speaking a feedfoward torque would only depend on the desired input

This PD-like control law has the advantage that for a change in position Δq the resulting force is

$$\Delta F = K_p \Delta q \tag{3.19}$$

and thus we can achieve an arbitrary symmetric stiffness. This is convenient for tasks in which we wish to specify how the robot interacts with the environment. Unfortunately, experimental results indicate that the trajectory tracking performance of this control law does not compare favorably with the computed torque control law [69]. Additionally there is no simple design criteria for choosing λ, K_v and K_p to achieve good tracking performance. While the stability results give necessary conditions for stability they do not provide a method for choosing the gains.

Internal forces

For constrained manipulator systems satisfying $J\dot{\theta} = G^T\dot{x}$, applied forces which lie in the null space of G generate no net motion of the object. In a real control application, we choose internal force to insure that all contact forces remain strictly inside the relevant friction cones. Using the notation of Section 2.5, if the condition $\mathcal{N}(G) \cap \overset{\circ}{FC} \neq \{\}$ is satisfied then we can apply an arbitrary external force (in the range of G) and still remain inside the friction cone by applying sufficient internal forces.

It is not clear how such an internal force should be chosen. In practice, it is often easiest to use a single constant internal force which is chosen based on the task to be performed [69]. Since there are no dynamics associated with applying a force, no control law is needed (in principle) to stabilize the applied internal force. To overcome measurement noise and modeling errors, some researchers use integral feedback to adjust the internal force [60].

3.5 Redundant manipulators

In order to perform a given task, a robot must have enough degrees of freedom to accomplish that task. In the analysis presented so far, we have assumed that our robots had exactly the number of degrees of freedom required to complete the task. This assumption manifested itself in our requirement that J be a square matrix in the dynamics derivation. We now relax that requirement and discuss its consequences.

Unlike conventional robot manipulators, constrained robot manipulators do not need to have more than 6 degrees of freedom in order to be redundant. The constraints themselves can introduce kinematic redundancy into a system. For example, if we attach a 6 degree of freedom finger to an object using a rolling contact, we have introduced two redundant degrees of freedom: the finger is free to roll in either of two directions without affecting the position of the object.

Thus it is absolutely essential that we include redundant mechanisms in our formulation.

It is interesting to note that there are two sources of redundancy introduced by our constraints: kinematic redundancy and actuator redundancy. *Kinematic redundancy* refers to *motions* of the fingers which do not affect the position of the object. *Actuator redundancy* refers to *forces* applied by the fingers which do not affect the object location, i.e., internal forces. The grasping constraint

$$J(q)\dot{\theta} = G^T(q)\dot{x}$$

contains all of the information necessary to determine these redundancies. Namely, the null space of J is the set of internal motions which do not affect the location of the object and the null space of G is precisely the space of internal forces. Since we have already discussed internal forces, we restrict our discussion to kinematic redundancy.

Dynamics of redundant systems

Consider first the kinematics of a single redundant manipulator, with no constraints. If we are willing to control the manipulator in joint space, the dynamics and control formulation presented above holds without modification. Suppose instead that we wish to write our controllers in end-effector coordinates. Mathematically, we represent the kinematics as a function $\mathcal{F} : \mathbb{R}^m \to \mathbb{R}^n$, where $m > n$. In this case $J := \frac{\partial \mathcal{F}}{\partial \theta}$ is not square and hence J^{-1} is not well defined so our derivation of equation (3.8) does not hold.

It is still possible to write the dynamics of redundant manipulators in a form consistent with equation (3.8). To do so, we define a matrix $K^T(\theta)$ whose columns span the null space of $J(\theta)$. As before, we assume that $J(\theta)$ is full row rank and hence $K(\theta)$ has constant rank $m - n$. The rows of $K^T(\theta)$ are basis elements for the space of velocities which cause no motion of the end-effector; we can thus define an *internal motion*, $\dot{y} \in \mathbb{R}^{m-n}$, using the equation

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{bmatrix} J \\ K \end{bmatrix} \dot{\theta} =: \bar{J}\dot{\theta}$$

By definition $\bar{J}$ is invertible and it follows from our previous derivation that

$$\tilde{M}(q)\begin{pmatrix} \ddot{x} \\ \ddot{y} \end{pmatrix} + \tilde{C}(q,\dot{q})\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} + \tilde{N}(q,\dot{q}) = \bar{J}^{-T}\tau$$

where $\tilde{M}$ and $\tilde{C}$ are obtained as in the nonredundant case but replacing J with $\bar{J}$ and G with I. If we choose K such that its rows are orthonormal, then $\bar{J}^{-1} = (J^+ \, K^T)$ where $J^+ = J^T(JJ^T)^{-1}$ is the least-squares right (pseudo) inverse of J. This approach is related to the *extended Jacobian* technique for resolving kinematic redundancy [2].

It would appear from the notation we have chosen that we have parameterized the internal motion of the system by a variable y. This is not necessarily the case. Since we chose K only to span the null space of J, there may not exist a function g such that $y = g(\theta)$ and $Dg = K$. A necessary and sufficient condition for such a g to exist is that each row of K satisfy $\frac{\partial K_{ij}}{\partial \theta_k} = \frac{\partial K_{ik}}{\partial \theta_j}$. This is merely the statement that mixed partials of g must commute. A more thorough treatment is given in Appendix I, at the end of the chapter.

It may not always be easy to choose $K(\theta)$ such that it is the differential of some function. For this reason, we shall not generally assume that an explicitly coordinatization of the internal motion manifold is available.

Example

Consider a three link planar manipulator with unit length links. If we let (x, y) be the location of the end-effector then

$$\begin{aligned} x &= \cos\theta_1 + \cos\theta_2 + \cos\theta_3 \\ y &= \sin\theta_1 + \sin\theta_2 + \sin\theta_3 \end{aligned}$$

where the link angles are all with respect to a fixed (inertial) axis. The Jacobian is

$$J(\theta) = \begin{bmatrix} -\sin\theta_1 & -\sin\theta_2 & -\sin\theta_3 \\ \cos\theta_1 & \cos\theta_2 & \cos\theta_3 \end{bmatrix}$$

There are many choices of $K(\theta)$ to complete $J(\theta)$. If we choose

$$K(\theta) = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \qquad \Rightarrow \qquad g(\theta) = \theta_3$$

this corresponds to choosing the angle of the end effector to parametrize internal motions. This choice of $K(\theta)$ is valid as long as $\theta_1 \neq \theta_2$ (i.e., when the first two links are not aligned).

If on the other hand we allow Mathematica to solve for $K(\theta)$, we are led to

$$K(\theta) = \begin{bmatrix} \sin(\theta_2 - \theta_3) & -\sin(\theta_1 - \theta_3) & \sin(\theta_1 - \theta_2) \end{bmatrix}$$

However

$$\frac{\partial K_1(\theta)}{\partial \theta_2} = \cos(\theta_2 - \theta_3) \neq -\cos(\theta_1 - \theta_3) = \frac{\partial K_2(\theta)}{\partial \theta_1}$$

Hence there is no g such that $Dg = K$ and the "variable" y has no physical meaning.

Control of redundant manipulators

Since the dynamics for a redundant manipulator have the same form as our canonical robot system, it is easy to extend the previous control laws to handle this case. If a coordinatization of the internal motion manifold is available, the control laws are identical with the addition of the redundant states. If we do not have a set of coordinates for the internal motion, but rather, only the velocities, then we must be slightly more careful. For example, the computed torque control law becomes

$$F = M(q)\begin{pmatrix} \ddot{x}_d + K_v \dot{e}_x + K_p e_x \\ \ddot{y}_d + K_v \dot{e}_y \end{pmatrix} + C(q,\dot{q})\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} + N(q,\dot{q})$$

Motion specification for such a control law would be in terms of a position trajectory $x_d(\cdot)$ and a velocity trajectory $\dot{y}_d(\cdot)$.

This control law will guarantee tracking of the given velocity. One method of calculating this velocity is to attempt to use the redundant degrees of freedom to minimize a cost function. Given the gradient of the cost function, we can project this vector (in joint space) onto the range of $K(\theta)$ which spans the internal motion directions. This determines $\dot{y}_d$, which can be passed to the controller. This type of cost criterion has been used with a computed torque-like controller by Hsu, et al. [44].

3.6 Hybrid stiffness controller

We now present a controller which partially generalizes the computed torque and PD-like controllers of Section 3.4. The motivation in analyzing this controller is twofold. First, it allows us to use the form and structure of the equations of motion and show how they are required in analyzing a robot control algorithm. Furthermore, the control includes as special cases the computed torque and PD controllers and hence we can prove stability for a class of controllers in one stroke.

While the analysis so far has only considered position control of manipulators, it is clear that we would like to allow for the possibility of contact with the environment. Such contact is crucial in many operations such as small parts assembly. The PD controller is useful for such situations since we can directly specify the quasi-static impedance of the manipulator. Unfortunately, the trajectory tracking capabilities of this controller are not ideal. To overcome this difficulty, we combine the computed torque and PD control laws. The intuition is that we can separate the controller and the task into two sets of directions—position directions and force directions. In the position directions we use a control law similar to computed torque, on the basis that we know how to select computed torque gains since the error equation is linear. In the force directions we use a PD controller with the same basic form as equation (3.18).

This type of hybrid approach has a rich history in the robotics literature. One of the first such hybrid controllers was proposed by Raibert and Craig [79]. They decomposed the task as above and applied closed loop position control in one set of directions and closed loop *force* control in a complementary set of directions. The disadvantage of their approach is that the system does not behave well when contact is lost. Indeed, if the robot loses contact with the environment, the force controlled variables undergo constant acceleration. By using *stiffness* control rather than force control in constrained directions, we seek to avoid this problem.

Combining a variant of computed torque control law and the natural control law we propose the position feedback law

$$F = M(q)\,(\ddot{x}_d + (\lambda_1+\lambda_2)\dot{e} + \lambda_1\lambda_2 e) + N(q,\dot{q}) + C(q,\dot{q})\,(\dot{x}_d + \lambda_1 e) + K_v\dot{e} + K_p e$$

with $\lambda_1, \lambda_2 \in \mathbb{R}$ and $K_v, K_p \in \mathbb{R}^{n\times n}$. This gives the error equation

$$M(q)\,(\ddot{e} + (\lambda_1+\lambda_2)\dot{e} + \lambda_1\lambda_2 e) + C(q,\dot{q})\,(\dot{e} + \lambda_1 e) + K_v\dot{e} + K_p e = 0 \qquad (3.20)$$

Before separating the space into position and force directions we examine the stability of the general equation. Notice that with $\lambda_2 = K_v = 0$ this control law is the same as the modified natural control law in equation (3.18). If $K_p = K_v = 0$ then the control law is similar to computed torque. The primary difference is that the control gains have a special form and consequently we can only place the poles of the linearized system at real values in the complex plane. Also, for simplicity, we have taken λ_1, λ_2 to be scalars, implying that the position directions have diagonal gain matrices.

Theorem 3.4 *There exists λ_1, λ_2 and K_v, K_p such that the control law (3.6) asymptotically tracks any $x_d(\cdot)$.*

Proof. To prove stability we use the Lyapunov candidate

$$\begin{aligned} V &= \tfrac{1}{2}(\dot{e}+\lambda_1 e)^T M(q)(\dot{e}+\lambda_1 e) + \tfrac{1}{2}e^T K_p e + \tfrac{1}{2}e^T\lambda_1 K_v e \\ &= \tfrac{1}{2}\begin{pmatrix} e \\ \dot{e}\end{pmatrix}^T \begin{bmatrix} \lambda_1^2 M(q) + K_p + \lambda_1 K_v & \lambda_1 M(q) \\ \lambda_1 M(q) & M(q)\end{bmatrix}\begin{pmatrix} e \\ \dot{e}\end{pmatrix}\end{aligned}$$

which can be bounded above and below by positive scalar multiples of the magnitude of $(e\,\dot{e})$. The derivative of V along a trajectory of (3.20) is

$$\begin{aligned} \dot{V} &= -(\dot{e}+\lambda_1 e)^T\lambda_2 M(q)(\dot{e}+\lambda_1 e) - \dot{e}^T K_v\dot{e} - e\lambda_1 K_p e \\ &= -\begin{pmatrix} e \\ \dot{e}\end{pmatrix}^T \begin{bmatrix} \lambda_1^2\lambda_2 M(q) + \lambda_1 K_p & \lambda_1\lambda_2 M(q) \\ \lambda_1\lambda_2 M(q) & \lambda_2 M(q) + K_v\end{bmatrix}\begin{pmatrix} e \\ \dot{e}\end{pmatrix}\end{aligned}$$

The calculation is similar to that given in the proof of Theorem 3.3 and requires that $\dot{M} - 2C$ is skew-symmetric. A more explicit calculation is given in Appendix II, at the end of this chapter.

A sufficient set of conditions for an exponentially stable error system is

$$\begin{gathered} M(q) > 0 \\ \lambda_1 > 0 \quad K_p > 0 \\ \lambda_2 \geq 0 \quad K_v \geq 0 \end{gathered}$$

Controller	λ_1	λ_2	K_p	K_v	Comments
Computed torque	>0	>0	$=0$	$=0$	exponential trajectory tracking
PD	$=0$	$=0$	>0	>0	asymptotic setpoint control
Natural	>0	$=0$	$=0$	>0	asymptotic trajectory tracking
Modified natural	>0	$=0$	>0	>0	exponential trajectory tracking

Table 3.1: Control laws which are special cases of the hybrid stiffness control law

and we see that the stability of the modified natural control law (3.18) can be derived from this by setting $\lambda_2 = 0$. Also, the simpler PD control law

$$F = K_p e + K_v \dot{e} + N(q, \dot{q})$$

can be shown stable when $\lambda_1 = \lambda_2 = 0$ and x_d is a constant ($\dot{x}_d, \ddot{x}_d = 0$). A table of control laws which can derived from this one is given in Table 3.1. □

Next we would like to analyze the stability of the control law when we only allow λ_2 to be nonzero in position directions and K_p, K_v to be nonzero in force directions. Intuitively this would give us a computed torque like scheme in position directions and a stiffness controller in the force directions. If we break x into x_1 (position) and x_2 (force) directions and apply the above noted restrictions the control law becomes:

$$F = M(q)\begin{pmatrix} \ddot{x}_{1d} + \lambda_1 \dot{e}_1 \\ \ddot{x}_{2d} \end{pmatrix} + C(q,\dot{q})\begin{pmatrix} \dot{x}_{1d} + \lambda_1 \dot{e}_1 \\ \dot{x}_{2d} \end{pmatrix} + \begin{pmatrix} M_{11}\lambda_2(\dot{e}_1 + \lambda_1 e_1) \\ K_v \dot{e}_2 + K_p e_2 \end{pmatrix} + N(q,\dot{q}) \tag{3.21}$$

Note that we have redefined K_v and K_p in this equation to act only in the x_2 direction. That is, $K_p, K_v \in \mathbb{R}^{n_2 \times n_2}$ where n_2 is the number of force directions. As before we choose λ_1, λ_2 as scalars for notational simplicity.

Theorem 3.5 *The closed loop system obtained using the control law (3.21) is asymptotically stable to zero if $\lambda_1, \lambda_2 > 0$ and $K_v, K_p > 0$.*

Proof. The error dynamics become

$$M(q)\begin{pmatrix} \ddot{e}_1 + \lambda_1 \dot{e}_1 \\ \ddot{e}_2 \end{pmatrix} + C(q,\dot{q})\begin{pmatrix} \dot{e}_1 + \lambda_1 e_1 \\ \dot{e}_2 \end{pmatrix} + \begin{pmatrix} M_{11}\lambda_2(\dot{e}_1 + \lambda_1 e_1) \\ K_v \dot{e}_2 + K_p e_2 \end{pmatrix} = 0 \tag{3.22}$$

We can analyze the stability of the system with a similar Lyapunov function,

$$V = \frac{1}{2}\begin{pmatrix} \dot{e}_1 + \lambda_1 e_1 \\ \dot{e}_2 \end{pmatrix}^T M(q) \begin{pmatrix} \dot{e}_1 + \lambda_1 e_1 \\ \dot{e}_2 \end{pmatrix} + \frac{1}{2} e_2^T K_p e_2$$

which gives (using the calculation in the appendix)

$$\dot{V} = -(\dot{e}_1 + \lambda_1 e_1)^T \lambda_2 M_{11} (\dot{e}_1 + \lambda_1 e_1) - \dot{e}_2^T K_v \dot{e}_2 \tag{3.23}$$

We see that $\dot{V}$ is only positive semidefinite and thus we cannot show exponential stability. We can however show asymptotic stability using Matrosov's theorem as presented by Paden and Panja [77].[2] In order to use the version of Matrosov's theorem presented there we must verify that $V^{(3)}$ is negative definite when restricted to the set of states $\dot{V} = 0$ (i.e. $\dot{e}_1 + \lambda_1 e_1 = 0$ and $\dot{e}_2 = 0$). Differentiating equation (3.23) twice and using the system dynamics in equation (3.22) we find

$$V^{(3)} = \begin{pmatrix} 0 \\ K_p e_2 \end{pmatrix}^T M^{-T} \begin{pmatrix} \lambda_2 M_{11} & 0 \\ 0 & K_v \end{pmatrix} M^{-1}(q) \begin{pmatrix} 0 \\ K_p e_2 \end{pmatrix}$$

which is negative definite as a function of e_2. Thus $(\dot{e}_1 + \lambda_1 e, e_2, \dot{e}_2) \to 0$. It can be shown that e_1 and $\dot{e}_1$ also approach zero (consider the linear system $\dot{e} = -\lambda e + u$; if u is zero then the state must converge to zero). □

The end result is that we get asymptotic stability in the case where no external forces are applied (i.e. we are not actually touching the environment). As a consequence, if we have a task in which we need to apply forces to an object and we lose contact completely we converge to the center of stiffness and the desired trajectory. Furthermore, if our task is such that motion in force directions is not allowed (i.e. pressing against a rigid surface) then we have exponential stability in the position directions since we can eliminate the force directions from the dynamics and the resulting stiffness control law is exponentially stable.

Experimental results

The hybrid stiffness control law presented in equation (3.21) has been implemented on a GE-A4 robot at U.C. Berkeley. The robot was commanded to apply a constant force along a surface aligned with the y axis while controlling the position along the x axis. To demonstrate the stability of the control law in both free and constrained environments, the surface was misaligned so that contact would be broken (see Figure 3.1). To apply a force normal to the surface the robot was commanded to move along a trajectory a fixed distance past the expected surface and K_p was chosen to achieve the desired force. A sinusoidally varying position was commanded in the x direction.

The results of the experiment are shown in Figure (3.2). Note that when contact is broken the end-effector converges to the commanded path. When contact is regained the force applied to the surface increases as the distance to the commanded path increases. The forces applied in the y direction do not affect the positioning along the x axis. Due to the gear backlash and coulomb friction on the GE arm the performance when the manipulator joints change direction is degraded (this occurs at $y = \pm 15$cm).

[2]LaSalle's invariance principle cannot be used in this context since (3.22) is a nonautonomous system which respect to e.

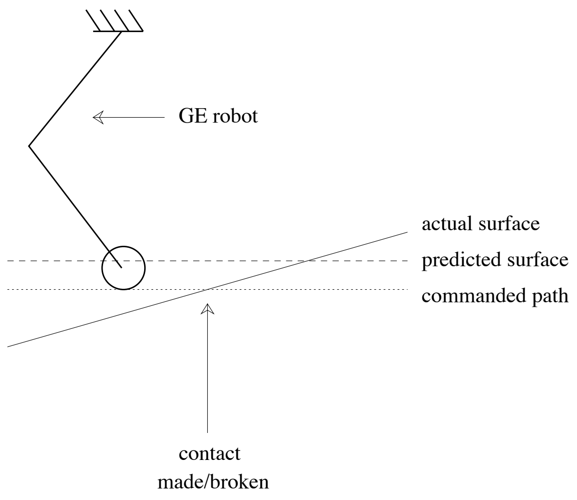


Figure 3.1: Simple grinding task with misaligned surface

There are several important points to make about the control law presented here. Unlike the computed torque control law, the closed loop dynamics of the system using a hybrid stiffness controller are *not* decoupled in the position and forces directions. This is evident from the form of equation (3.22) (M is not diagonal). This is the price that must be paid for being able to specify the quasi-static stiffness. If the position error is small then the position terms will not cause large forces in the stiffness directions so in practice this coupling is often negligible.

Furthermore, the hybrid stiffness control law should not be considered to be a replacement for the more conventional hybrid force/position control law presented by Raibert and Craig [79]. If the task is such that a desired force is required, then the hybrid force formulation can often provide better performance since the force is controlled directly. For some problems, like peg-in-hole insertion, the system must be controlled in such a way that a desired stiffness is achieved in a certain set of directions. It is this class of problems for which the hybrid control algorithm is applicable. The hybrid force control algorithm can be derived using the formulation presented here but it cannot be shown to be stable when the force directions are not constrained (the manipulator accelerates at a constant rate). For this reason there may be force tasks for which the hybrid stiffness/position algorithm has a better overall performance than the hybrid force/position formulation.

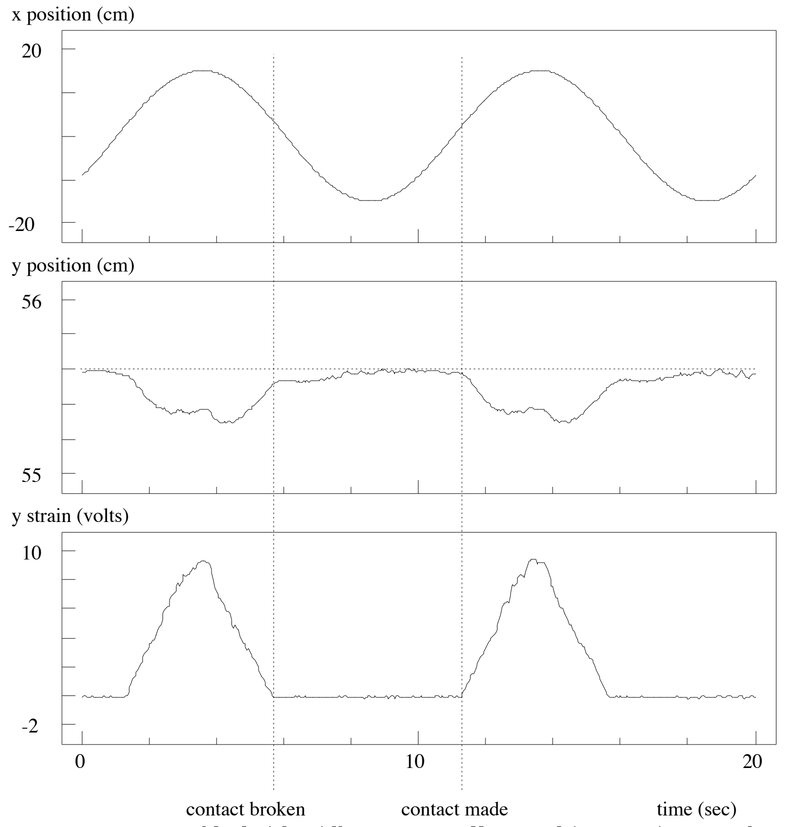


Figure 3.2: Response of hybrid stiffness controller pushing against surface that is misaligned

3.7 Summary

Given a set of robots with joint configuration θ and an object with configuration x, it is possible to derive the dynamics of the constrained system. We consider constraints of the form

$$J(\theta, x)\dot{\theta} = G^T(\theta, x)\dot{x} \tag{3.24}$$

This constraint includes grasping contacts (fixed or rolling), changes of coordinates, and contact with the environment. The resulting dynamics can be written as

$$M(q)\ddot{x} + C(q, \dot{q})\dot{x} + N(q, \dot{q}) = F = GJ^{-T}\tau \tag{3.25}$$

where $q = (\theta, x)$. The following properties are satisfied for this system: $M(q) > 0$ for all q and $\dot{M} - 2C$ is skew symmetric for all q.

The constraints (3.24) contain the information on the structure of the redundancies of the system:

$$\begin{aligned} \mathcal{N}(J) &\longleftrightarrow \text{internal motions} \\ \mathcal{N}(G) &\longleftrightarrow \text{internal forces} \end{aligned}$$

In the case of internal motions, it is possible to extend the derivation of the dynamics by extending the Jacobian. The form of the equations of motion are preserved by this extension.

Using the form and structure of the robot dynamics, several control laws can be shown to track arbitrary trajectories with *exponential rate of convergence*:

$$\begin{aligned} \text{Computed torque:} \quad F &= M(q)(\ddot{x}_d + K_v\dot{e} + K_p e) + C(q, \dot{q})\dot{x} + N(q, \dot{q}) \\ \text{Modified natural:} \quad F &= M(q)(\ddot{x}_d + \lambda\dot{e}) + C(q, \dot{q})(\dot{x}_d + \lambda e) + N(q, \dot{q}) + K_v\dot{e} + K_p e \end{aligned}$$

It is possible to combine these control laws by identifying position and force control directions and applying the computed torque and natural controllers in respective directions. Experimental results indicate that this is can be an effective approach for systems requiring simultaneous position and stiffness control.

Appendix I – parameterization of internal motions

In this appendix we discuss the parameterization of the internal motion of a redundant manipulator. This presentation requires the use of exterior differential forms on $\mathbb{R}^n$. The necessary details can be found in Spivak [88] or another book on differential geometry. In particular, we rely on the fact that on $\mathbb{R}^n$ all closed one forms are exact. That is, given a one form ω on $\mathbb{R}^n$, if the exterior derivative $d\omega$ is identically zero then ω is exact; there exists a real-valued function θ on $\mathbb{R}^n$ such that $\omega = d\theta$.

Let $\mathcal{F} : \mathbb{R}^m \to \mathbb{R}^n$ be a function representing the kinematics of a redundant manipulator ($m > n$). Define $J(\theta) := \frac{\partial \mathcal{F}}{\partial \theta}(\theta) \in \mathbb{R}^{m \times n}$ and let $K(\theta) \in \mathbb{R}^{(m-n)\times m}$ be a matrix such that the columns of K^T span the null space of $J(\theta)$. We consider only those θ for which J is full row rank.

Proposition 3.6 *For all θ for which $J(\theta)$ is full row rank, the matrix*

$$\bar{J}(\theta) = \begin{bmatrix} J(\theta) \\ K(\theta) \end{bmatrix}$$

is invertible.

Proof. For any matrix $A \in \mathbb{R}^{m\times n}$, $\mathbb{R}^n = \mathcal{R}(A^T) \oplus \mathcal{N}(A)$. Letting $A = J$, $\mathcal{R}(A^T)$ is precisely the row span of J and the null space of A is spanned by the rows of K. Hence the rows of J and K are linearly independent. □

Proposition 3.7 *There exists $g : \mathbb{R}^m \to \mathbb{R}^{n-m}$ such that $K = \frac{\partial g}{\partial \theta}$ if and only if*

$$\frac{\partial K_{ij}}{\partial \theta_k} = \frac{\partial K_{ik}}{\partial \theta_j} \quad i = 1, \cdots m$$

Proof. Let $\omega = K_i$, the i^{th} row of K. ω is a one-form on $\mathbb{R}^n$

$$\omega = \sum \alpha_i d\theta_i$$

Since all closed one forms on $\mathbb{R}^n$ are exact, a necessary and sufficient condition for K_i to be exact is that it be closed

$$0 = d\omega = \sum_{i,j} \frac{\partial \alpha_i}{\partial \theta_j} d\theta_j \wedge d\theta_i$$

Equating terms and using skew-symmetry of the wedge product yields $\frac{\partial \alpha_i}{\partial \theta_j} - \frac{\partial \alpha_j}{\partial \theta_i}$ which is equivalent to the stated condition. □

Appendix II – calculations for hybrid stiffness proofs

This appendix provides detailed calculations for the proofs of stability (tracking) for the hybrid control laws presented in Section 3.6. The notation used here is described in more detail in that section.

Calculations for Theorem 3.4

We use the Lyapunov candidate

$$V = \frac{1}{2}(\dot{e} + \lambda_1 e)^T M(\dot{e} + \lambda_1 e) + \frac{1}{2}e^T K_p e + \frac{1}{2}\lambda_1 e^T K_v e$$

The error equation for the system, obtained by substituting the control law (3.6) into the equations of motion, is

$$M(\ddot{e} + \lambda_1 \dot{e}) = -M(\lambda_2 \dot{e} + \lambda_2 \lambda_1 e) - C(\dot{e} + \lambda_1 e) - K_v \dot{e} - K_p e$$

Taking the derivative of V and evaluating along trajectories of the error system:

$$\begin{array}{rcl}
\dot{V} & = & (\dot{e} + \lambda_1 e)^T M(\ddot{e} + \lambda_1 \dot{e}) + \frac{1}{2}(\dot{e} + \lambda_1 e)^T \dot{M}(\dot{e} + \lambda_1 e) + \dot{e}^T K_p e + \lambda_1 \dot{e}^T K_v e \\
& = & (\dot{e} + \lambda_1 e)^T [-M(\lambda_2 \dot{e} + \lambda_2 \lambda_1 e) - C(\dot{e} + \lambda_1 e) - K_v \dot{e} - K_p e] + \\
& & \frac{1}{2}(\dot{e} + \lambda_1 e)^T \dot{M}(\dot{e} + \lambda_1 e) + \dot{e}^T K_p e + \lambda_1 \dot{e}^T K_v e \\
& = & -(\dot{e} + \lambda_1 e)^T M \lambda_2 (\dot{e} + \lambda_1 e) + \frac{1}{2}(\dot{e} + \lambda_1 e)^T [\dot{M} - 2C](\dot{e} + \lambda_1 e) \\
& & -(\dot{e} + \lambda_1 e)^T K_v \dot{e} - (\dot{e} + \lambda_1 e)^T K_p e + \dot{e}^T K_p e + \dot{e}^T K_v \lambda_1 e \\
& = & -(\dot{e} + \lambda_1 e)^T M \lambda_2 (\dot{e} + \lambda_1 e) - \dot{e}^T K_v \dot{e} - e \lambda_1 K_p e \\
& = & -\begin{pmatrix} e \\ \dot{e} \end{pmatrix}^T \begin{bmatrix} \lambda_1^2 \lambda_2 M + \lambda_1 K_p & \lambda_1 \lambda_2 M \\ \lambda_1 \lambda_2 M & \lambda_2 M + K_v \end{bmatrix} \begin{pmatrix} e \\ \dot{e} \end{pmatrix}
\end{array}$$

This proof can be extended to the case where λ_1 and λ_2 are matrices if λ_1 and K_v commute. Also $M\lambda_2$ becomes $\lambda_2^{T/2} M \lambda_2^{1/2}$ for $\lambda_2 \in \mathbb{R}^{n \times n}$ symmetric.

Calculations for Theorem 3.5

The control law is

$$F = M\begin{pmatrix} \ddot{x}_{1d} + \lambda_1 \dot{e}_1 \\ \ddot{x}_{2d} \end{pmatrix} + C\begin{pmatrix} \dot{x}_{1d} + \lambda_1 \dot{e}_1 \\ \dot{x}_{2d} \end{pmatrix} + \begin{pmatrix} M_{11}\lambda_2(\dot{e}_1 + \lambda_1 e_1) \\ K_v \dot{e}_2 + K_p e_2 \end{pmatrix} + N \tag{3.26}$$

This gives closed loop dynamics

$$M\begin{pmatrix} \ddot{e}_1 + \lambda_1 \dot{e}_1 \\ \ddot{e}_2 \end{pmatrix} + C\begin{pmatrix} \dot{e}_1 + \lambda_1 e_1 \\ \dot{e}_2 \end{pmatrix} + \begin{pmatrix} M_{11}\lambda_2(\dot{e}_1 + \lambda_1 e_1) \\ K_v \dot{e}_2 + K_p e_2 \end{pmatrix} = 0 \tag{3.27}$$

Use the Lyapunov candidate

$$V = \tfrac{1}{2}\begin{pmatrix} \dot e_1 + \lambda_1 e_1 \\ \dot e_2 \end{pmatrix}^T M \begin{pmatrix} \dot e_1 + \lambda_1 e_1 \\ \dot e_2 \end{pmatrix} + \tfrac{1}{2} e_2^T K_p e_2$$

This is basically the restriction of the previous Lyapunov function to the respective force and position coordinates (i.e., λ's acting in position directions and K's acting in force directions). The derivative of V evaluated along the trajectory is calculated as in the previous case, leaving:

$$\dot V = -\begin{pmatrix} e_1 \\ e_2 \\ \dot e_1 \\ \dot e_2 \end{pmatrix}^T \begin{pmatrix} \lambda_1^2\lambda_2 M_{11} & 0 & \lambda_1\lambda_2 M_{11} & 0 \\ 0 & 0 & 0 & 0 \\ \lambda_1\lambda_2 M_{11} & 0 & \lambda_2 M_{11} & 0 \\ 0 & 0 & 0 & K_v \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ \dot e_1 \\ \dot e_2 \end{pmatrix}$$

This matrix is clearly singular, so we calculate the derivatives in order to apply Matrosov's theorem. Define $S := \dot e_1 + \lambda_1 e_1$

$$\begin{aligned} \dot V &= -S^T \lambda_2 M_{11} S + \dot e_2^T K_v \dot e_2 \\ V^{(2)} &= -2\dot S^T \lambda_2 M_{11} S - S^T \lambda_2 \dot M_{11} S - 2\dot e_2 K_v \ddot e_2 \\ V^{(3)} &= -2\ddot S^T \lambda_2 M_{11} S - 2\dot S^T \lambda_2 M_{11} \dot S - 2\dot S^T \lambda_2 \dot M_{11} S \\ &\quad -2\dot S^T \lambda_2 \dot M_{11} S - S^T \lambda_2 \ddot M_{11} S - 2\ddot e_2 K_v \ddot e_2 - 2\dot e_2 K_v e_2^{(3)} \end{aligned}$$

Restricting $V^{(3)}$ to the surface $\dot V = 0$ ($S = 0$, $\dot e_2 = 0$)

$$\begin{aligned} V^{(3)} &= -2\dot S^T \lambda_2 M_{11} \dot S - 2\ddot e_2 K_v \ddot e_2 \\ &= -2(\ddot e_1 + \lambda_1 \dot e_1)^T \lambda_2 M_{11} (\ddot e_1 + \lambda_1 \dot e_1) - 2\ddot e_2^T K_v \ddot e_2 \\ &= -2\begin{pmatrix} \ddot e_1 + \lambda_1 \dot e_1 \\ \ddot e_2 \end{pmatrix}^T \begin{bmatrix} \lambda_2 M_{11} & 0 \\ 0 & K_v \end{bmatrix} \begin{pmatrix} \ddot e_1 + \lambda_1 \dot e_1 \\ \ddot e_2 \end{pmatrix} \end{aligned}$$

On the manifold $\dot V = 0$ the closed loop dynamics become

$$\begin{pmatrix} \ddot e_1 + \lambda_1 \dot e_1 \\ \ddot e_2 \end{pmatrix} = M^{-1} \begin{pmatrix} 0 \\ K_p e_2 \end{pmatrix}$$

and hence

$$V^{(3)} = -2\begin{pmatrix} 0 \\ K_p e_2 \end{pmatrix}^T M^{-T} \begin{pmatrix} \lambda_2 M_{11} & 0 \\ 0 & K_v \end{pmatrix} M^{-1} \begin{pmatrix} 0 \\ K_p e_2 \end{pmatrix}$$

which is negative definite as a function of e_2. Thus the conditions for Matrosov's theorem are satisfied and $(S, e_2, \dot e_2) \to 0$.

To show that $e_1, \dot e_1 \to 0$, we use a simple linear systems argument. Consider the linear system

$$\dot e_1 + \lambda_1 e_1 = u \quad \Rightarrow \quad \dot e_1 = -\lambda_1 e_1 + u$$

This is a stable linear system ($\lambda_1 > 0$) and hence $u \to 0$ implies $e_1 \to 0$ as $t \to \infty$. Therefore $S \to 0$ implies that e_1 (and hence $\dot e_1$) converge to zero.

Chapter 4

Primitives for robot control

This chapter presents a constructive method for describing hierarchical control systems for robots with contact constraints. Our goal is to define a set of language primitives which can be used as building blocks to construct complex controllers for complex systems. The actions of the individual primitives are derived from the mathematical structure of the equations of motion derived in the previous chapter.

4.1 Introduction

Motivation

A multi-fingered robot hand can be modeled as a set of robots which are connected to an object by a set of constraints. The analysis presented in Chapter 3 allows us to model this interconnection and create a new dynamic system which encodes the constraints. In fact, this procedure is sufficiently straightforward that it may be automated: by specifying the contact constraint between the robots and the object, the new equations of motion for the composite robot can be derived using a symbolic manipulation program. Computer programs have been used to automatically generate robot equations of motion for some time (see [18] and the references therein). Part of the goal of this chapter is to describe the manner in which this can be done for robots obeying contact constraints.

More importantly, in the context of robotics we are interested in constructing controllers for robot systems. Section 3.4 showed control laws which could be applied to a very general class of robots. In an actual robot control system, it may not be desirable to perform the feedback compensation only to the composite robot. The computations that must be carried out at this level are complex and may take too much time to compute for a satisfactory control implementation [69, 70]. To avoid these difficulties, we would like the ability to distribute our control at different levels and among different processing units.

Controlling systems with many degrees of freedom performing a coordinated task is of interest not only to robotics researchers but also to the biomechanics community. The study of human motor control mechanisms led the Russian psychologist Bernstein to question how the brain could control a system with so many different degrees of freedom interacting in a complex fashion [41]. Many of the properties which make mammalian motor control systems complicated are also present in robotic systems. By studying some of the control structures present in biological systems, we can gain insight into the design of robot control systems.

To some degree of accuracy, a human finger can be modeled as a set of rigid bodies (bones) which are actuated by muscles. Control of these muscles is distributed throughout the central nervous system [32]. The first level of motor control occurs at the spinal cord, which is responsible for reflex responses, such as the knee jerk in humans. Higher levels of control occur in the brain stem and motor cortex, with both hierarchical and parallel pathways to the muscle present. These higher levels of motor control integrate motor commands and allow more sophisticated processing of stimula to affect muscle behavior. The difference levels of motor control are characterized by differences in response times, typically 30-50 msec for the spinal loop and 150-250 msec for higher level control loops [33]. These differences in time scale are also seen in robotic systems, where fusion of complex sensor information requires increased computational complexity.

Another important feature of biological systems is redundancy. From the mechanics point of view, the redundancy has two chief forms.

Kinematic redundancy refers to an excess in the number of degrees of freedom present over that needed to perform a generic task. Thus a human arm has (roughly) seven degrees of freedom even though only six are needed to locally position the hand in space. These excess degrees of freedom serve many purposes. Since the range of motion of the individual joints is limited, kinematic redundancy can be used to increase the volume of the reachable configurations. In a more local context, redundancy can be used to optimized the arm configuration based on the task to be performed. Thus we might use different arm configurations when writing on a chalk board versus lifted a heavy object. The first task requires dexterity while the second requires strength and stability. Different arm configurations can be used to maximize these properties [43].

Actuator redundancy is indicated by the presence of internal forces which cause no net motion on the underlying system. In lifting a box we may use large or small grasping forces without affecting the location of the box. In biological systems actuator redundancy is often inherent in the muscle actuation system. Internal forces in muscles can be used to alter the characteristics of the overall system [43]. By pretensing muscles in the arm, the impulse response of the arm is considerably changed.

In designing controllers for robot systems, it is useful to consider systems with some of the same properties as biological robots. In doing so, we gain an understanding not only of robot control, but also insight into the biological control process. In particular, we will be interested in allowing specification of hierarchical control systems for robots with kinematic and actuator redundancy.

Background

The robotics and control literature contains a number of topics which are applicable to the approach presented here. We limit ourselves to a brief review of robot programming languages and hierarchical control mechanisms. Other relevant references will be presented in the body of the chapter.

Robot programming languages

Traditional robot programming languages, such as AML, Robot-BASIC, and VAL II, are general purpose computer languages with special features added to allow specification of robot motion [22, 29]. By incorporating the ability to perform coordinate transformations, communicate with sensing and control hardware, and alter program flow based on sensor measurements, robot programming languages allow complicated tasks to be implemented. The control algorithms underlying robot programming languages usually employ a two stage hierarchy. Using kinematics calculations and joint interpolation, a task is converted to a set of joint angle trajectories. These trajectories are passed to a low level controller which executes them using feedback to insure performance and accuracy.

A more general description of motion is possible using Brockett's Motion Description Language [14, 28]. One of the key ideas is the use of sequences of triples (u, K, T) which convey trajectory information, feedback control and termination conditions to a PostScript-like interpreter. The controller construction described in this chapter was motivated by descriptions of MDL. Our work utilizes the explicit form of robot dynamics and contact constraints to describe the organization and control of complex robots; Brockett's work is less explicit but more complete in the description of sequences of motion.

An object oriented approach similar to that presented here has been described by Cutkosky, Howe and Witkin [23]. They present a method for describing the dynamics of a robot hand grasping an object. The advantage of the method they propose is that it is very general and does not rely on rigid contact models, allowing compliant fingertips to be considered. Their method is closely related to graph theoretic methods in mechanics which keep track of generalized forces and displacements along branches of a graph representing the system interconnection. The main emphasis of their approach is on system description rather than controller design. It is precisely because we are interested in designing controllers that we have initially limited the class of interconnections we are allowed.

Hierarchical control

Time delays inherent in biological motor systems indicate that control is decentralized, occurring at many different levels of the central nervous system. For the same reasons that biological controllers are decentralized, it is desirable to construct hierarchical controllers in a computer controlled robot system. Communication and computation delays make nested levels of control and attractive method for providing high system bandwidth while coordinating many degrees of freedom. These motivations are not limited to robotics: there is a large literature concerned with the use of decentralized control for general dynamic systems.

Centralized control has been defined as a case in which every sensor's output influences every actuator [84]. The study of large scale systems led to a number of results concerning weakly coupled systems and multi-rate controllers. Graph decomposition techniques permitted the isolation of sets of states, inputs, and outputs which were weakly coupled. This decomposition simplified stability analyses and controller design. In multi-processor control systems, decentralized control is often mandated by restrictions on the communication rates between processors. Hierarchical controllers limit communication to adjacent levels of the the hierarchy.

Robotic applications of hierarchical control are exemplified by Clark's HIC [19], an operating system intended to manage servo loops found in robot controllers. Clark emphasises distributed processing and interprocessor communication in his description of HIC. He also notes that in executing a task it is necessary to switch between different planning procedures at appropriate points in a task. Thus, writing may be decomposed into grasping a pencil, transporting it, and making marks on a sheet of paper. Each of these subtasks requires a different controller to properly reflect the constraints on the system. Many of the same considerations are present in our work. However, the approach here is considerably more explicit in the areas of controller construction and analysis.

Overview

The fundamental objects in our robot specification environment are objects called robots. In a graph theoretic formalism they are nodes of a tree structure. At the lowest level of the tree are leaves which are instantiated by the `define` primitive. Robots are dynamical systems which are recursively defined in terms of the properties of their daughter robot nodes. Inputs to robots consist of desired positions and conjugate forces. The outputs of a robot consist of actual positions and forces. Robots also possess attributes such as inertial parameters and kinematics.

There are two other primitives which act on sets of robots to yield new robots, so that the set of robots is closed under these operations. These primitives (`attach` and `control`) may be considered as links between nodes and result in composite robot objects. Nodes closer to the root may possess fewer degrees of freedom, indicating a compression of information upon ascending the tree.

The `attach` primitive reflects geometrical constraints among variables and in the process of yielding another robot object, accomplishes coordinate transformations. `Attach` is also responsible for a bidirectional flow of information: expanding desired positions and forces to the robots below, and combining actual position and force information into an appropriate set for the higher level robot. In this sense the state of the root robot object is recursively defined in terms of the states of the daughter robots.

The `control` primitive seeks to direct a robot object to follow a specified "desired" position/force trajectory according to some control algorithm. The controller applies its control law (several different means of control are available such as PD and computed torque) to the desired and actual states to compute expected states for the daughter robot to follow. In turn, the daughter robot passes its actual states through the controller to robot objects further up the tree.

The block diagram portion of Figure 4.1 may be seen to be an example of a robot system comprised of these primitives. Starting from the bottom: two fingers are defined; each finger is controlled by muscle tension/stiffness and spinal reflexes; the fingers are attached to form a composite hand; the brainstem and cerebellum help control and coordinate motor commands and sensory information; and finally at the level of the cortex, the fingers are thought of as a pincer which engages in high level tasks such as picking.

The material in this chapter arose from joint work with Curt Deno, Kris Pister and Shankar Sastry [26, 24]. A complete description, including an implementation of the basic primitives using Mathematica is described in a technical memo [25].

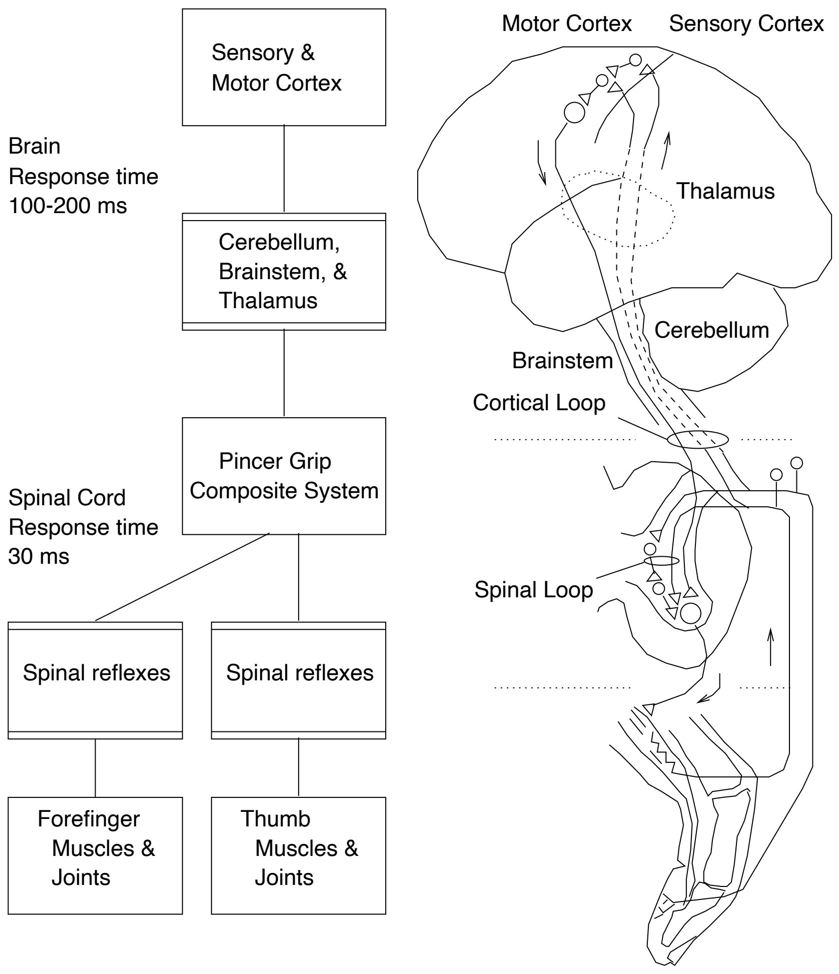


Figure 4.1: Hierarchical control scheme of a human finger [26]. At the highest level, the brain is represented as sensory and motor cortex (where sensory information is perceived and conscious motor commands originate) and brainstem and cerebellar structures (where motor commands are coordinated and sent down the spinal cord). A pair of fingers forms a composite system for grasping which is shown integrated at the level of the spinal cord. The muscles and sensory organs of each finger form low level spinal reflex loops. These low level loops respond more quickly to disturbances than sensory motor pathways which travel to the brain and back. Brain and spinal feedback controllers are represented by double lined boxes. (Figure courtesy of D. Curtis Deno)

4.2 Primitive definitions

In this section we describe a set of primitives that gives us the mathematical structure necessary to build a system and control specification for dynamical robot systems. We do not require any particular programming environment or language, borrowing instead freely from languages such as C, Lisp and C++. As much as possible, we have tried to define the primitives so that they can be implemented in any of these languages.

As our basic data structure, we will assume the existence of an object with an associated list of attributes. These attributes can be thought of as a list of name-value pairs which can be assigned and retrieved by name. A typical attribute which we will use is the inertia of a robot. The existence of such an attribute implies the existence of a function which is able to evaluate and return the inertia matrix of a robot given its configuration.

Attributes will be assigned values using the notation *attribute* := *value*. Thus we might define our inertia attribute as

$$M(\theta) := \begin{bmatrix} m_1 l_1^2 + m_2 l_2^2 & m_2 l_1 l_2 \cos(\theta_1 - \theta_2) \\ m_2 l_1 l_2 \cos(\theta_1 - \theta_2) & m_2 l_2^2 \end{bmatrix} \tag{4.1}$$

In order to evaluate the inertia attribute, we would call M with a vector $\theta \in \mathbb{R}^2$. This returns a 2×2 matrix as defined above. The Coriolis/centrifugal attribute, C, and friction/gravity/nonlinear attribute, N, are defined similarly.

To encourage intuition, we will first describe the actions of the primitives for the case of non-redundant robots. Additionally, we ignore the internal forces that are present in constrained systems. Extensions to these cases are presented in Section 4.4.

The robot object

The fundamental object used by all primitives is a *robot*. Associated with a robot are a set of attributes which are used to define its behavior:

M	inertia of the robot
C	Coriolis/centrifugal vector
N	friction and gravity vector
rd	return position and force information about the robot
wr	send position and information to the robot

The rd function returns the current position, velocity, and acceleration of the robot, and the forces measured by the robot. Each of these will be a vector quantity of dimension equal to the number of degrees of freedom of the robot. Typically a robot may only have access to its joint positions and velocities, in which case $\ddot{x}$ and F will be *nil*.

The wr function is used to specify an expected position and force trajectory that the robot is to follow. In the simplest case, a robot would ignore everything

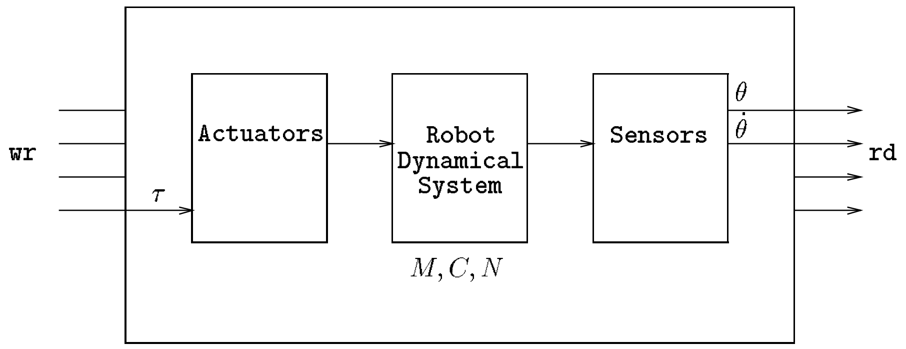


Figure 4.2: Example of the `define` primitive. The robot shown here corresponds to a robot with torque driven motors and only position and velocity sensing.

but F and try to apply this force/torque at its actuators. As we shall see later, other robots may use this information in a more intelligent fashion. We will often refer to the arguments passed to write by using the subscript e. Thus x_e is the expected or desired position passed to the wr function.

The task of describing a primitive is essentially the same as describing how it generates the attributes of the new robot. The following sections describe how each of the primitives generates these attributes. The new attributes created by a primitive are distinguished by a tilde over the name of the attribute.

DEFINE primitive

Synopsis:

DEFINE(M, C, N, rd, wr)

The `define` primitive is used to create a simple robot object. It defines the minimal set of attributes necessary for a robot. These attributes are passed as arguments to the `define` primitive and a new robot object possessing those attributes is created:

$$\begin{array}{rcl}
\tilde{M}(\theta) & := & M(\theta) \\
\tilde{C}(\theta,\dot\theta) & := & C(\theta,\dot\theta) \\
\tilde{N}(\theta,\dot\theta) & := & N(\theta,\dot\theta) \\
\tilde{rd}() & := & rd() \\
\tilde{wr}(\theta_e,\dot\theta_e,\ddot\theta_e,\tau_e) & := & wr(\theta_e,\dot\theta_e,\ddot\theta_e,\tau_e)
\end{array}$$

Several different types of robots can be defined using this basic primitive. For example, a DC motor actuated robot would be implemented with a wr function which converts the desired torque to a motor current and generates this current by communicating with some piece of hardware (such as a D/A converter). This type of robot system is shown in Figure 4.2. On the other hand, a stepper motor actuated robot might use a *wr* function which ignores the torque argument and uses the position argument to move the actuator. Both robots would use a *rd* function which returns the current position, velocity, acceleration and actuator torque. If any of these pieces of information is missing, it is up to the user to insure that they are not needed at a higher level. We may also define a *payload* as a degenerate robot by setting the *wr* argument to the *nil* function. Thus commanding a motion and/or force on a payload produces no effect.

ATTACH primitive

```
Synopsis:
    ATTACH(J, G, h, payload, robot-list)
```

The `attach` primitive is used to describe constrained motion involving a payload and one or more robots. `Attach` must create a new robot object from the attributes of the payload and of the robots being attached to it. The specification of the new robot requires a velocity relationship between coordinate systems ($J\dot{\theta} = G^T\dot{x}$), an invertible kinematic function relating robot positions to payload position ($x = h(\theta)$), a payload object, and a list of robot objects involved in the contact.

The only difference between the operation of the `attach` primitive and the equations derived for constrained motion of a robot manipulator is that we now have a *list* of robots each of which is constrained to contact a payload. However, if we define θ_R to be the combined joint angles of the robots in `robot-list` and similarly define M_R and C_R as block diagonal matrices composed of the individual inertia and Coriolis matrices of the robots, we have a system which is identical to that presented previously. Namely, we have a "robot" with joint angles θ_R and inertia matrix M_R connected to an object by a constraint of the form

$$J\dot{\theta}_R = G^T\dot{x} \tag{4.2}$$

where once again J is a block diagonal matrix composed of the Jacobians of the individual robots. To simplify notation, we define $\mathcal{A} := J^{-1}G^T$ so that

$$\dot{\theta}_R = \mathcal{A}\dot{x} \tag{4.3}$$

The attributes of the new robot can thus be defined as:

$$\begin{aligned}
\tilde{M} &:= M_p + \mathcal{A}^T M_R \mathcal{A} && (4.4)\\
\tilde{C} &:= C_p + \mathcal{A}^T C_R \mathcal{A} + \mathcal{A}^T M_R \dot{\mathcal{A}} && (4.5)\\
\tilde{N} &:= N_p + \mathcal{A}^T N_R && (4.6)\\
\tilde{rd}() &:= (h(\theta_R),\ \mathcal{A}^+\dot{\theta}_R,\ \mathcal{A}^+\ddot{\theta}_R + \dot{\mathcal{A}}^+\dot{\theta}_R,\ \mathcal{A}^T\tau_R) && (4.7)\\
\tilde{wr}(x_e, \dot{x}_e, \ddot{x}_e, F_e) &:= wr_R(h^{-1}(x_e),\ \mathcal{A}\dot{x}_e,\ \mathcal{A}\ddot{x}_e + \dot{\mathcal{A}}\dot{x}_e,\ \mathcal{A}^{+T}F_e) && (4.8)
\end{aligned}$$

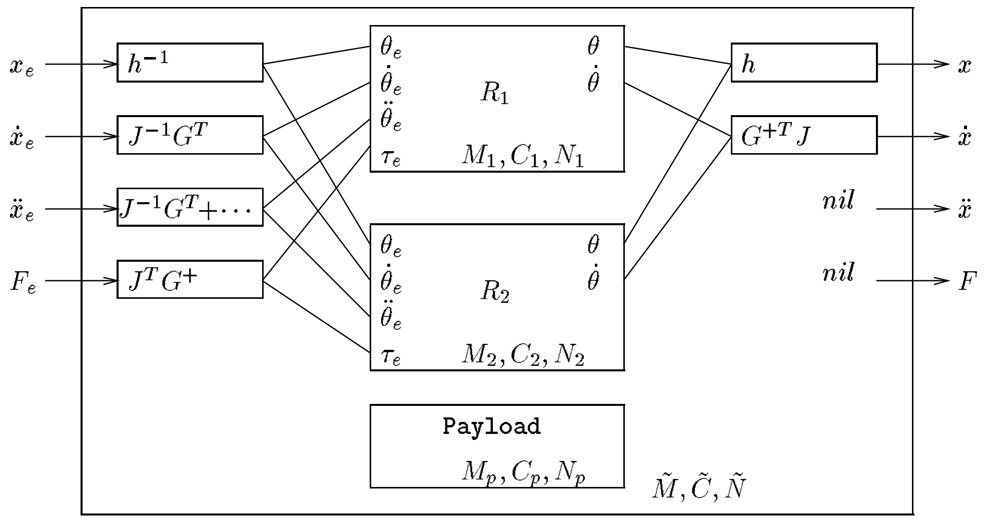


Figure 4.3: Data flow in a two robot **attach**. In this example we illustrate the structure generated by a call to **attach** with 2 robots and a payload (e.g. a system like Figure 4.5). The two large interior boxes represent the two robots, with their input and output functions and their inertia properties. The outer box (which has the same structure as the inner boxes) represents the new robot generated by the call to **attach**. In this example the robots do not have acceleration or force sensors, so these outputs are set to *nil*.

where M_p, C_p, N_p are attributes of the payload, M_R and C_R are as described above, and N_R is a stacked vector of friction and gravity forces. This construction is illustrated in Figure 4.3.

The $\tilde{rd}$ attribute for an attached robot is a function which queries the state of all the robots in `robot-list`. Thus θ_R in equation (4.7) is constructed by calling the individual rd functions for all of the robots in the list. The θ values for each of these robots are then concatenated to form θ_R and this is passed to the forward kinematic function. A similar computation occurs for $\dot{\theta}_R$, $\ddot{\theta}_R$ and τ_R. Together, these four pieces of data form the return value for the $\tilde{rd}$ attribute.

In a dual manner, the $\tilde{wr}$ attribute is defined as a function which takes a desired trajectory (position and force), converts it to the proper coordinate frame and sends each robot the correct portion of the resultant trajectory. A special case of the **attach** primitive is its use with a *nil* payload object and $G = I$. In this case, M_p, C_p, and N_p are all zero and the equations above reduce to a simple change of coordinates.

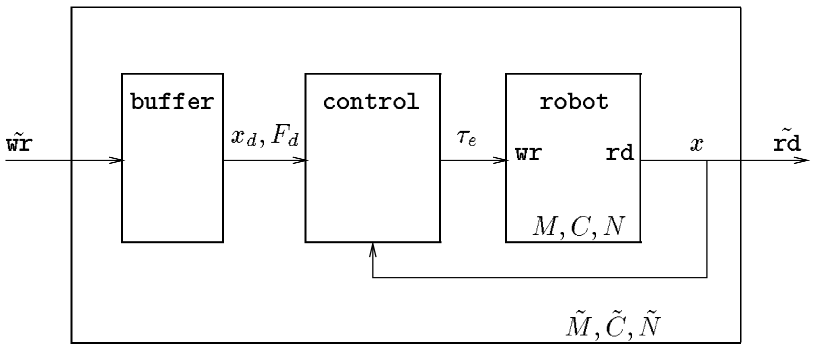


Figure 4.4: Data flow in a typical controlled robot. Information written to the robot is stored in an internal buffer where it can be accessed by the controller. The controller uses this information and the current state of the robot to generate forces which cause it to follow the desired trajectory.

CONTROL primitive

```
Synopsis:
      CONTROL(robot, controller)
```

The `control` primitive is responsible for assigning a controller to a robot. It is also responsible for creating a new robot with attributes that properly represent the controlled robot. The attributes of the created robot are completely determined by the individual controller. However, the *rd* and *wr* attributes will often be the same for different controllers. Typically the *rd* attribute for a controlled robot will be the same as the *rd* attribute for the underlying robot. That is, the current state of the controlled robot is equivalent to the current state of the uncontrolled robot. A common *wr* attribute for a controlled robot would be a function which saved the desired position, velocity, acceleration and force in a local buffer accessible to our controller. This configuration is shown in Figure 4.4.

The dynamic attributes $\tilde{M}$, $\tilde{C}$ and $\tilde{N}$ are determined by the controller. At one extreme, a controller which compensates for the inertia of the robot would set the dynamic attributes of the controlled robot to zero. This does not imply that the robot is no longer a dynamic object, but rather that controllers at higher levels can ignore the dynamic properties of the robot, since they are being compensated for at a lower level. At the other end of the spectrum, a controller may make no attempt to compensate for the inertia of a robot, in which case it should pass the dynamic attributes on to the next higher level. Controllers which lie in the middle of this range may partially decouple the dynamics of the manipulator without actually completely compensating for them. To illustrate these concepts we next consider one possible controller class, computed torque.

However, many control laws originally formulated in joint space may also be employed since the structure of equation (3.8) has been preserved.

Computed torque controller

As we mentioned in Chapter 3.4, the computed torque controller is an exactly linearizing controller which inverts the nonlinearities of a robot to construct a linear system. The dynamics of the system are compensated for by the use of feedforward torques and premultiplication of the feedback terms by the inertia matrix of the system. As a consequence, the resulting system has no uncompensated dynamics. The proper representation for such a system sets the dynamical attributes $\tilde{M}$, $\tilde{C}$, and $\tilde{N}$ to zero and uses the *rd* and *wr* attributes as described above. We introduce x_d to refer to the buffered desired trajectory.

The control process portion of the controller is responsible for generating input robot forces which cause the robot to follow the desired trajectory (available in x_d). Additionally, the controller must determine the "expected" trajectory to be sent to lower level robots. For the computed torque controller we use the resolved acceleration [66] to generate this path. This allows computed torque controllers running at lower levels to properly compensate for nonlinearities and results in a linear error response. The methodology is similar to that used in determining that the dynamic attributes of the output robot should be zero. The control algorithm is implemented by the following equations:

$$
\begin{array}{lcl}
(x, \dot{x}, \cdot, \cdot) & = & rd() \\
\ddot{x}_e & = & \ddot{x}_d + K_v(\dot{x}_d - \dot{x}) + K_p(x_d - x) \\
\dot{x}_e & = & \int_0^t \ddot{x}_e \\
x_e & = & \int_0^t \dot{x}_e \\
F_e & = & M(q)\ddot{x}_e + C(q, \dot{q})\dot{x} + N(q, \dot{q}) + F_d \\
wr(x_e, \dot{x}_e, \ddot{x}_e, F_e) & &
\end{array}
$$

where *rd* and *wr* are attributes of the robot which is being controlled.

Note the existence of the F_d term in the calculation of F_e. This is placed here to allow higher level controllers to specify not only a trajectory but also a force term to compensate for higher level payloads. In essence, a robot which is being controlled in this manner can be viewed as an ideal force generator which is capable of following an arbitrary path.

The computed torque controller defines two new attributes, K_p and K_v, which determine the gains (and hence the convergence properties) of the controller. A variation of the computed torque controller is the feedforward controller, which is obtained by setting $K_p = K_v = 0$. This controller can be used to distribute nonlinear calculations in a hierarchical controller, as we shall see in Section 4.3.

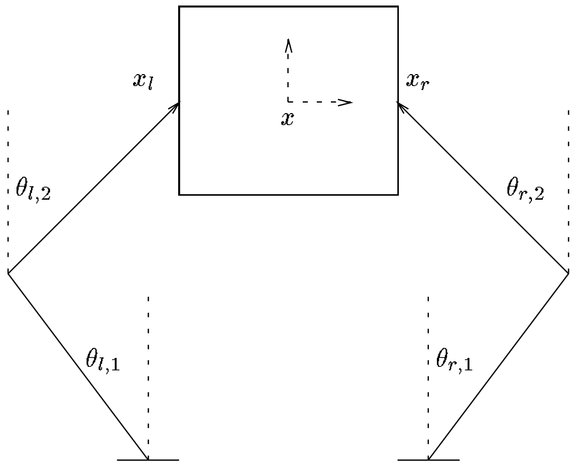


Figure 4.5: Planar two-fingered hand. Contacts are assumed to be maintained throughout the motion. For this particular system the box position and orientation, x, form a generalized set of coordinates for the system.

4.3 Example

To make the use of the primitives more concrete we present an example of a planar hand grasping a box (Figure 4.5) using a complicated control structure. We shall assume the existence of the following functions

M_b	box inertia matrix in Cartesian coordinates
M_l, M_r	inertia matrix for the left and right fingers
C_b, C_l, C_r	Coriolis/centrifugal vector for box and fingers
f	finger kinematics function, $f : (\theta_l, \theta_r) \mapsto (x_l, x_r)$
g	grasp kinematics function, $g : (x_l, x_r) \mapsto x_b$
J	finger Jacobian, $J = \frac{\partial f}{\partial \theta}$
G	grasp map, consistent with g
`rd_left`, **`rd_right`**	read the current joint position and velocity
`wr_left`, **`wr_right`**	generate a desired torque on the joints

where $\theta_l, \theta_r, x_l, x_r$, and x_b are defined as in Figure 4.5.

Consider the control structure illustrated in Figure 4.6. This control structure is obtained by analogy with biological systems in which controllers run at several different levels simultaneously. At the lowest level we use simple PD control laws attached directly to the individual fingers. These PD controllers mimic the stiffness provided by muscle coactivation in a biological system [42]. Additionally, controllers at this level might be used to represent spinal reflex actions. At a somewhat higher level, the fingers are attached and considered as a single unit with relatively complicated dynamic attributes and Cartesian configuration. At this point we employ a feedforward controller (computed torque with no error correction) to simplify these dynamic properties, as viewed by higher levels of the brain. With respect to these higher levels, the two fingers appear to be two Cartesian force generators represented as a single composite robot.

Up to this point, the representation and control strategies do not explicitly involve the box, a payload object. These force generators are next attached to the box, yielding a robot with the dynamic properties of the box but capable of motion due to the actuation in the fingers. Finally, we use a computed torque controller at the very highest level to allow us to command motions of the box without worrying about the details of muscle actuation. By this controller we simulate the actions of the cerebellum and brainstem to coordinate motion and correct for errors.

In terms of the primitives that we have defined, we build this structure from the bottom up

```
left  = DEFINE($M_l$, $C_l$, 0, 0, rd_left, wr_left)
pd_left =  CONTROL(left, pd)
right = DEFINE($M_r$, $C_r$, 0, 0, rd_right, wr_right)
pd_right = CONTROL(right, pd)
    fingers = ATTACH($J$, I, $f$, nil, pd_left, pd_right)
    ff_fingers = CONTROL(fingers, feed-forward)
    box  = DEFINE($M_b$, $C_b$, 0, 0, nil, nil)
        hand = ATTACH(I, $G$, $g$, box, ff_fingers)
        ct_hand = CONTROL(hand, computed-torque)
```

It is helpful to illustrate the flow of information to the highest level control law. In the evaluation of x_b and $\dot{x}_b$, the following sequence occurs through calls to the *rd* attribute:

```
hand: asks for current state, $x_b$ and $\dot{x}_b$
    finger: ask for current state, $x_f$ and $\dot{x}_f$
        left: read current state, $\theta_l$ and $\dot{\theta}_l$
        right: read current state, $\theta_r$ and $\dot{\theta}_r$
    finger: $x_f, \dot{x}_f \leftarrow f(\theta_l, \theta_r), J(\dot{\theta}_l, \dot{\theta}_r)$
hand: $x_b, \dot{x}_b \leftarrow g(x_f), G^T \dot{x}_f$
```

When we write a set of hand forces using the *wr* attribute, it causes a similar chain of events to occur.

The structure in Figure 4.6 also has interesting properties from a more traditional control viewpoint. The low level PD controllers can be run at high servo rates (due to their simplicity) and allow us to tune the response of the system to reject high frequency disturbances. The Cartesian feedforward controller permits a distribution of the calculation of nonlinear compensation terms at various levels, lending itself to multiprocessor implementation. Finally, using a computed torque controller at the highest level gives the flexibility of performing the controller design in the task space and results in a system with linear error dynamics.

4.4 Extensions to the basic primitives

Having presented the primitives for non-redundant robot systems in which we ignore internal forces, we now describe the modifications necessary to include both internal motion and internal forces in the primitives. As before, these extensions are based on the dynamic equations given in Chapter 3 and rely on the fact that the equations of motion of this class of systems can be expressed in a unified way. Recall that the fundamental constraint used to generate the equations of motion had the form

$$J\dot{\theta} = G^T\dot{x} \tag{4.9}$$

Internal motion and force can be thought of as manifestations of redundancies in the manipulator, and both can be used to improve performance. A common use of redundant motion in robotics is to specify a cost function and use the redundancy of the manipulator to attempt to minimize this cost function [65]. If we extend our definition of the wr function so that it takes not only an "external" trajectory, but also an internal trajectory (which might be represented as a cost function or directly as a desired velocity in the internal motion directions) then this internal motion can be propagated down the graph structure. A similar situation occurs with internal or constraint forces.

The matrices $J(q)$ and $G(q)$ in equation (4.9) embody the fundamental properties of the constrained system. We begin by assuming that $J(q)$ and $G(q)$ are both full row rank. The null space of $J(q)$ corresponds to motions which do not affect the configuration of the object, i.e., internal motions. Likewise, the null space of $G(q)$ describes internal forces—the set of forces which cause no motion of the object. A complete trajectory for a robot must specify not only external motion and force for a robot but also the internal motion and force which lie in these subspaces.

Internal forces

To allow internal forces to be specified and controlled, we must first add them to the rd and wr attributes. This is done by simply adding an extra value to the list of values returned by rd and adding an extra argument to wr. Thus the wr attribute is called as

$$wr(x_e, \dot{x}_e, \ddot{x}_e, F_e, F_i) \tag{4.10}$$

where F_i is the desired internal force.

Internal forces are "created" by the `attach` primitive. The internal force directions for a constraint are represented by an orthonormal matrix $H(\theta)$ whose rows form a basis for the null space of $G(\theta)$. Since any of the daughter robots may itself have an internal force component, the internal force vector for a robot created by attach consists of two pieces: the internal forces created by this constraint and the combined internal forces for the daughter robots. We shall refer to these two components as $F_{i,1}$ and $F_{i,2}$, respectively. The force transformations which describe this relationship are

$$\tau_R = \left(\begin{array}{c} \tau_{R,e} \\ \hline \tau_{R,i} \end{array}\right) = \left(\begin{array}{cc|c} J^T G^+ & J^T H^T & 0 \\ \hline 0 & 0 & I \end{array}\right) \left(\begin{array}{c} F_e \\ F_{i,1} \\ \hline F_{i,2} \end{array}\right) \tag{4.11}$$

where $\tau_{R,e}$ is the vector of external forces for the daughter robots and $\tau_{R,i}$ is the vector of internal forces. This equation is analogous to equation (3.9) in Section 3.3. Note that $\tau_{R,i}$ is identical to $F_{i,2}$, thus internal force specifications required by the daughter robots are appended to the internal force specification required due to the constraint. Expanding equation (4.11) we see the appropriate definition for the new wr attribute generated by `attach` is

$$\tilde{wr}(x_e, \dot{x}_e, \ddot{x}_e, F_e, F_i) := wr_R(\cdots, J^T G^+ F_e + J^T H^T F_{i,1}, F_{i,2}) \tag{4.12}$$

The inclusion of internal forces in the rd attribute is similar. The sensed forces from the robots, τ_R, are simply split into external and internal components and converted to the appropriate internal and external forces for the new robot. This is equivalent to inverting equation (4.11):

$$F = \left(\begin{array}{c} F_e \\ F_{i,1} \\ \hline F_{i,2} \end{array}\right) = \left(\begin{array}{c|c} GJ^{-T} & 0 \\ HJ^{-T} & 0 \\ \hline 0 & I \end{array}\right) \left(\begin{array}{c} \tau_{R,e} \\ \hline \tau_{R,i} \end{array}\right) \tag{4.13}$$

It follows that

$$\tilde{rd}() := (\cdots, GJ^{-T}\tau_{R,e}, \begin{pmatrix} HJ^{-T}\tau_{R,e} \\ \tau_{R,i} \end{pmatrix}) \tag{4.14}$$

Internal forces are resolved by the `control` primitive. In principle, a controller can specify any number of the internal forces for a robot. Internal forces which are not resolved by a controller are left as internal forces for the newly defined robot. In practice, controllers will often be placed immediately above the attached robots since internal forces are best interpreted at this level. Unlike external motions and forces, internal forces are not subject to coordinate change and so leaving such forces unresolved causes higher level controllers to use low level coordinates.

Internal motions

Internal motions are also created by the `attach` primitive, this time due to a non-square Jacobian matrix. As before, we must add arguments to the *rd* and *wr* attributes of robots to handle the extra information necessary for motion specification. We only assume that the redundant velocities and accelerations are defined, so we add only those quantities to *rd* and *wr*. Since the notation becomes quite cumbersome, we won't actually define the *rd* and *wr* primitives, but just specify the internal and external motion components.

Given a constraint which contains internal motions, the `attach` primitive must again properly split the motion among the robots attached to the object. Define $K(\theta)$ to be a matrix whose rows span the null space of $J(\theta)$. Then we can rewrite our constraint as

$$\left(\begin{array}{c|c} J & 0 \\ K & 0 \\ \hline 0 & I \end{array}\right) \left(\begin{array}{c} \dot{\theta}_{R,e} \\ \hline \dot{\theta}_{R,i} \end{array}\right) = \left(\begin{array}{cc|c} G^T & 0 & 0 \\ 0 & I & 0 \\ \hline 0 & 0 & I \end{array}\right) \left(\begin{array}{c} \dot{x}_e \\ \dot{x}_{i,1} \\ \hline \dot{x}_{i,2} \end{array}\right) \tag{4.15}$$

Defining $\bar{J}$ and $\bar{G}$ as the extended Jacobian and grasp matrices,

$$\bar{J} = \begin{pmatrix} J \\ K \end{pmatrix} \qquad \bar{G}^T = \begin{pmatrix} G^T & 0 \\ 0 & I \end{pmatrix} \tag{4.16}$$

we see that $\bar{J}$ is full rank and so we can use it to define $\mathcal{A} = \bar{J}^{-1}\bar{G}^T$ in equations (4.4–4.8). This then defines the dynamics attributes created by `attach`. Note that the dimension of the constrained subspace (where internal forces act) is unchanged by this extension.

The input and output attributes are described in a manner similar to those used for internal forces. For *wr* the external component of the motion is given by

$$\begin{aligned} \dot{\theta}_{R,e} &= \mathcal{A}\begin{pmatrix} \dot{x}_e \\ \dot{x}_i \end{pmatrix} && (4.17) \\ &= J^{+}G^T\dot{x}_e + K^T\dot{x}_i && (4.18) \end{aligned}$$

$\ddot{\theta}_{R,e}$ is defined similarly. $\theta_{R,e}$ is only defined if an inverse kinematic function, h^{-1}, is given. Otherwise that information is not passed to the daughter robots. As before, if the robots themselves have internal motions then these should be split off and passed unchanged to the lower level robots.

The *rd* attribute is defined by projecting robot motions into an object motion component and an internal motion component. That is

$$\begin{aligned} x_e &= h(\theta_{R,e}) && (4.19) \\ \dot{x}_e &= G^{+}J\dot{\theta}_{R,e} && (4.20) \\ \dot{x}_i &= \begin{pmatrix} K\dot{\theta}_{R,e} \\ \dot{\theta}_{R,i} \end{pmatrix} && (4.21) \end{aligned}$$

$\ddot{x}_e$ and $\ddot{x}_i$ are obtained by differentiating the expression for $\dot{x}_e$ and $\dot{x}_i$.

Controllers must also be extended to understand redundant motion. This is fundamentally no different than control of an ordinary manipulator except that position information is not available in redundant directions. Thus the computed torque law would become

$$F = M(q)\begin{pmatrix} \ddot{x}_{e,d} + K_v\dot{e}_e + K_p e_e \\ \ddot{x}_{i,d} + K_v\dot{e}_i \end{pmatrix} + C(q,\dot{q})\begin{pmatrix} \dot{x}_e \\ \dot{x}_i \end{pmatrix} + N(q,\dot{q}) \tag{4.22}$$

Motion specification for such a control law would be in terms of a position trajectory $x_e(\cdot)$ and a velocity trajectory $\dot{x}_i(\cdot)$. If a controller actually resolves the internal motion (by specifying $\dot{x}_{i,d}(\cdot)$ based on a pseudo inverse calculation for example), then the internal motion will be masked from higher level controllers; otherwise it is passed on.

Control laws commonly use the position of the object as part of the feedback term. This may not always be available for systems with non-integrable constraints (such as grasping with rolling contacts). If the object position cannot be calculated from θ then we must retrieve it from some other source. One possibility is to use an external sensor which senses x directly, such as a camera or tactile array. The function to "read the sensor" could be assigned to the payload *rd* function and `attach` could use this information to return the payload position when queried. Another possible approach is to integrate the object velocity (which is well defined) to bookkeep the payload position.

Some care must also be taken with the evaluation of dynamic attributes for robots which do not have well defined inverse kinematic functions. There are some robot control laws which use feedforward terms that depend on the desired output trajectory, e.g., $M(x_d)\ddot{x}_d$. The advantage of writing such control laws is that this expression can be evaluated offline, increasing controller bandwidth. This calculation only makes sense if the desired configuration, q_d, can be written as a function of x_d and more generally if q can be written as a function of x. One solution to this problem is to only evaluate dynamic attributes of a robot at the current configuration. Assuming each robot in the system can determine its own position, these attributes are then well defined. For all the control laws presented in this paper, M, C and N are always evaluated at q, the current configuration.

4.5 Discussion

Working from a physiological motivation we have developed a set of robot description and control primitives consistent with Lagrangian dynamics. Starting from a description of the inertia, sensor, and actuator properties of individual robots, these primitives allow for the construction of a composite constrained motion system with control distributed at all levels. Robots, as dynamical systems, are recursively defined in terms of daughter robots. The resulting hierarchical system can be represented as a tree structure in a graph theoretic formalism, with sensory data fusion occurring as information flows from the leaves of the tree (individual robots and sensors) toward the root, and data expansion as relatively simple motion commands at the root of the tree flow down through contact constraints and kinematics to the individual robot actuators.

One of the major future goals of this research is to implement the primitives presented here on a real system. This requires that efforts be made toward implementing primitives in as efficient fashion as possible. The first implementation choice is deciding when computation should occur. It is possible that the entire set of primitives could be implemented off-line. In this case, a controller-generator would read the primitives and construct suitable code to control the system. A more realistic approach is to split the computation burden more judiciously between on-line and off-line resources. Symbolically calculating the attributes of the low level robots and storing these as precompiled functions might enable a large number of systems to be constructed using a library of daughter robot systems. Although the expressions employed are continuous time, in practice digital computers will be relied upon for discrete time implementations. This raises the issue of whether lower computation rates may be practical for higher level robots/controllers.

In addition to implementation issues, there are still several theoretical issues which we hope to address. We would like to have stability proofs for classes of control hierarchies, e.g. any hierarchy with a computed torque controller at the highest level and only feedforward controllers below it can be shown to be exponentially stable. There is also no provision in the primitives for dynamics which can not be written in the form of equation (3.8). Adaptive identification and control techniques may be useful in cases where unmodeled dynamics substantially affect system performance.

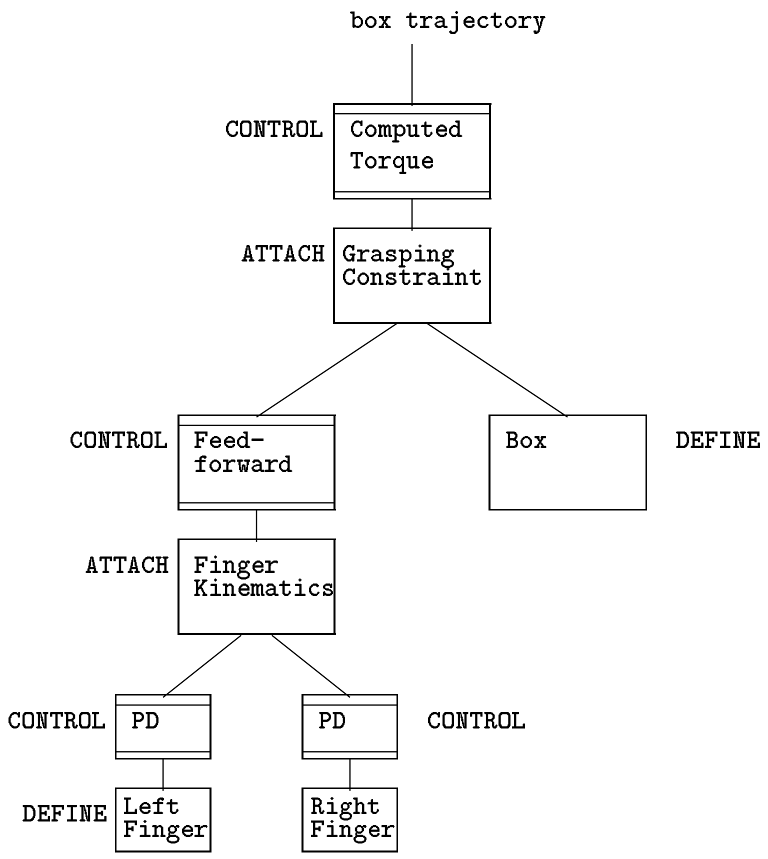


Figure 4.6: Multi-level computed torque and stiffness (PD). Controllers are used at each level to provide a distributed control system with biological motivation, desirable control properties, and computational efficiency.

Chapter 5

Nonholonomic motion planning

The preceding chapters gave detailed methods for generating controllers which can track a given trajectory for a robot system with kinematic constraints. These trajectories were specified as an output trajectory for the system, for example the desired location of a grasped object as a function of time. In this form, the kinematic constraints are automatically satisfied by the controller, which will command joint motions that achieve the desired output trajectory. In our earlier analysis, we concentrated only on the outputs, ignoring the individual joint angles.

Consider a more complicated problem where we wish to move from one *configuration* of the system to another. In the multifingered hand example, a configuration normally corresponds to the location of the grasped object *and* the values of the joint angles. In planning a trajectory between two configurations, we must obey the kinematic constraints on the system at all times. For many systems, it is not possible to move between two arbitrary configurations—the kinematic constraints limit the set of reachable configurations to a space of lower dimension.

We are interested in studying the case in which the kinematic constraints do not restrict the reachable configurations. An example of such a system, somewhat easier to visualize than a multifingered hand, is an automobile. The kinematics of an automobile are constrained because the front and rear wheels are only allowed to roll and spin, but not slide sideways. As a consequence, the car itself is not capable of sliding sideways, or rotating in place. Despite this, we know from our own experience that we can park an automobile at any position and orientation. This is an example of a nonholonomic motion planning problem.

This chapter is an introduction to nonholonomic path planning. The basic tools used in the study of nonholonomic systems are those of differential geometry and control theory. We provide a concise description of some of the more useful concepts and techniques and use them to generate a partial classification for nonholonomic systems. We then proceed to study the literature in nonholonomic motion planning using the formalism we introduce. Our own methods of nonholonomic motion generation are described in detail in a subsequent chapter.

5.1 Introduction and motivation

Path planning for robots has a rich history. The traditional difficulty in planning robot trajectories is the avoidance of obstacles. This problem is referred to as the piano mover's problem, in which we attempt to move an object (the piano) through a cluttered environment. The problem is solved by investigating the free configuration space of the piano—all configurations for which the piano does not intersect an obstacle. If the start and goal locations of the piano lie in the same connected component of the free configuration space, the motion planning problem is solvable. Algorithms for efficiently computing solutions to this problem are numerous; an overview can be found in [16].

The problem which we investigate is fundamentally different. In addition to constraints imposed by the environment, we wish to include kinematic constraints in the system. In particular, we are interested in multirobot systems with constraints of the form

$$J(\theta, x)\dot{\theta} = G(\theta, x)\dot{x} \tag{5.1}$$

where θ and x are the configurations of the manipulator(s) and the object being manipulated. These are constraints on the *velocities* of the system. In some cases, the constraints may be explicitly integrable, giving constraints of the form

$$h(x, \theta) = 0$$

If this is possible, motion of the system is restricted to a level surface of h. Such a constraint is said to be *holonomic*. By choosing coordinates for the surface, configuration space methods can be applied.

A constraint is said to be *nonholonomic* if it cannot be written as an algebraic constraint in the configuration space. There are many types of nonholonomic constraints, corresponding to different physical situations. For example:

$h(q, \dot{q}) = 0$	State space constraint
$\omega(q)\dot{q} = 0$	Linear velocity constraint
$h(q) \leq 0$	Configuration inequality constraint
$h(q, \dot{q}) \leq 0$	State space inequality constraint

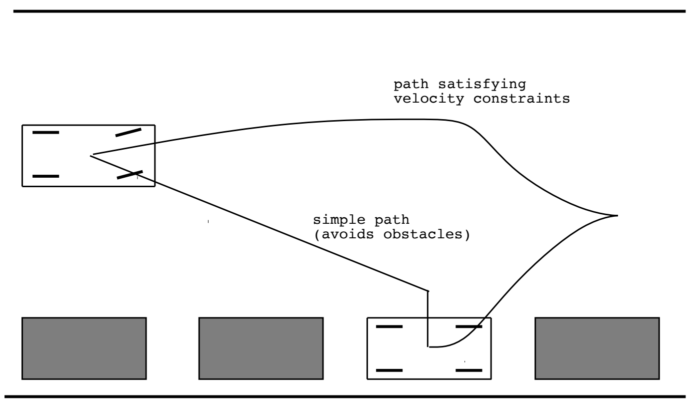


Figure 5.1: Paths generated by conventional path planners may ignore nonholonomic constraints. The straight line path in the figure indicates the path that a conventional path planner might generate. The curved path is one which satisfies the kinematic constraints of the car.

Even in the context of grasping there are several different types of constraints present. For example, unidirectional force constraints can be represented as inequality constraints in the state space—motion into an object is not allowed, but motion away from an object is unrestricted. When contact is maintained, the linear velocity constraint in equation (5.1) is valid. We shall limit ourselves to the special class of linear velocity constraints (sometimes called *Pfaffian* constraints).

Any holonomic constraint can be represented as a linear velocity constraint simply by differentiating the holonomic constraint with respect to time. As the automobile example illustrates, the converse is not true. The implication is that although the system velocities are constrained, motion is not restricted to a surface in the configuration space. Part of the goal of this chapter is to understand when a velocity constraint is actually holonomic. In the sequel, we will refer to nonintegrable linear velocity constraints simply as nonholonomic constraints.

Conventional path planners implicitly assume that arbitrary motion in the configuration space is allowed as long as obstacles are avoided. If a system contains nonholonomic constraints, many of these path planners cannot be directly applied. If we attempt to ignore the constraint, the paths generated by a path planner may not be feasible (see Figure 5.1). For this reason, it is important to understand how to efficiently compute paths for nonholonomic systems.

It will be convenient for us to convert problems with nonholonomic constraints into another form. Consider the problem of constructing a path $x(t) \in$

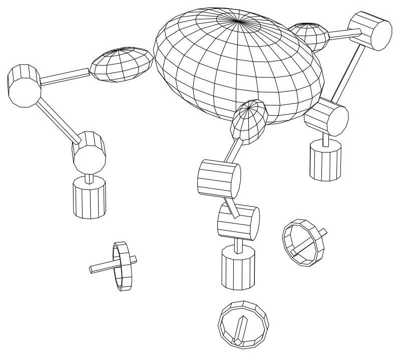


Figure 5.2: Three fingered hand grasping an object. The small circles below each finger indicate the internal forces applied by each finger. (Figure courtesy of John Hauser)

$\mathbb{R}^n$ between a given x_0 and x_1 subject to k constraints which are linear in $\dot{x}$:

$$\omega_i(x)\dot{x} = 0 \qquad i = 1, \cdots, k$$

We assume the ω_i's are smooth and linearly independent over the ring of smooth functions. Formally, these constraints are exterior differential one-forms on $\mathbb{R}^n$. This type of constraint arises in many mechanical systems, including systems with rolling contact and systems which conserve angular momentum. Specific examples of such systems are given in Section 5.3.

Rather than use the machinery of exterior differential systems, we convert the problem to one in control theory. Roughly speaking, we would like to convert the constraint specification from describing the directions in which we can't move to those in which we can. Formally, we choose a basis for the right null space of the constraints, denoted by $g_i(x) \in \mathbb{R}^n$, $i = 1, \cdots, n-k$. The path planning problem can be restated as finding an input function, $u(t) \in \mathbb{R}^m$, such that the control system

$$\dot{x} = g_1(x)u_1 + \cdots + g_m(x)u_m$$

is driven from x_0 to x_1. It will be shown that if the ω_i's are smooth and linearly independent (over the ring of smooth functions), then the g_i's inherit these properties.

Dynamic finger repositioning

As an example of this conversion, we consider the grasping control problem with rolling constraints. Such a system is shown in Figure 5.2. In Chapter 2 we derived the kinematic equations of contact for a single body in contact with a set of fingers. The constraints have the form

$$J(x_o,\theta)\dot{\theta} = G^T(x_o,\theta)\begin{pmatrix} v_o \\ \omega_o \end{pmatrix}$$

If the grasp is stable (G is onto) then we can solve uniquely for (v_o,ω_o) as

$$\begin{pmatrix} v_o \\ \omega_o \end{pmatrix} = G^{+}(x_o,\theta)J(x_o,\theta)\dot{\theta}$$

Letting $u=\dot{\theta}$, and writing $\dot{x}_o$ for (v_o,ω_o) we have

$$\begin{pmatrix} \dot{x}_o \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} G^{+}(x_o,\theta)J(x_0,\theta) \\ I \end{pmatrix} u \tag{5.2}$$

This is the control system associated with the grasping constraint. The fundamental question is to what extent we can steer x_o and θ to arbitrary values by clever choice of u.

It is instructive to rewrite equation (5.2) in a different set of coordinates. In particular, we would like to investigate the motion of the contact points rather than the finger joint angles. Recall that $g_i = x_o x_{f_i}^{-1}$ represents the position and orientation of the i^{th} finger ($x_{f_i} \in SE(3)$) relative to the body ($x_o \in SE(3)$). $\eta_i = (\alpha_{o_i}, \alpha_{f_i}, \psi_i)$ is the vector of the i^{th} contact coordinates with $\alpha_{o_i} \in \mathbb{R}^2$ representing the contact point in the surface coordinates of the object, $\alpha_{f_i} \in \mathbb{R}^2$ the contact point in finger surface coordinates, and ψ_i the angle between the surface frames. The contact kinematics can be written as

$$\dot{\eta}_i = B_i(x_o,\eta_i)\begin{pmatrix} v_t \\ \omega_t \end{pmatrix}$$

where $B_i(x_o,\eta_i)$ represents the contact kinematics,

$$\begin{pmatrix} v_t \\ \omega_t \end{pmatrix} = \begin{bmatrix} M_o^{-T}\frac{\partial c_o}{\partial \alpha_o}^T & 0 \\ 0 & M_o^{-T}\frac{\partial c_o}{\partial \alpha_o}^T \end{bmatrix}\begin{bmatrix} I & \mathcal{S}(n_o) \\ 0 & \mathcal{S}(c_o) \end{bmatrix}\begin{pmatrix} v_i \\ \omega_i \end{pmatrix}$$

and (v_i,ω_i) are the twist coordinates for $\dot{g}_i g_i^{-1}$:

$$\begin{pmatrix} v_i \\ \omega_i \end{pmatrix} = (\dot{g}_i {g_i}^{-1})^{\wedge}$$

By virtue of the fact that $g_i = x_o x_f^{-1}$, v_i and ω_i are linear functions of (v_o,ω_o) and (v_{f_i},ω_{f_i}) so that we may write

$$\dot{\eta}_i = B_i'(x_o,\eta_i)\begin{pmatrix} v_o \\ \omega_o \\ v_{f_i} \\ \omega_{f_i} \end{pmatrix}$$

Combining these equations for each finger and using the Jacobian of the finger kinematics to express (v_{f_i}, ω_{f_i}) as a function of $\dot{\theta}$, we have

$$\dot{\eta} = B''(x_o, \eta) \begin{pmatrix} v_o \\ \omega_o \\ \dot{\theta} \end{pmatrix} \tag{5.3}$$

We can now rewrite equation equation (5.2) using equation (5.3). Let $\dot{\theta} = \dot{\theta}_1 + \dot{\theta}_2$ with $\dot{\theta}_1 \in \mathcal{R}(J^T)$ and $\dot{\theta}_2 \in \mathcal{N}(J)$. Further, suppose that the fingers do not have more than 6 degrees of freedom and hence x_f (locally) uniquely determines θ. In this case, θ can be written as a function of x_o and η using Theorem 2.1 to eliminate x_f. Finally we set

$$\begin{aligned} u_1 &= G^+(x_o, \theta) J(x_o, \theta) \dot{\theta}_1 \\ u_2 &= K(x_o, \theta) \dot{\theta}_2 \end{aligned}$$

where the columns of $K^T(x_o, \theta)$ span the null space of $J(x_o, \theta)$. Using these definitions, equations (5.2) and (5.3) can be combined and written as

$$\begin{aligned} \dot{x}_o &= u_1 \\ \dot{\eta} &= B_1(x_o, \eta) u_1 + B_2(x_o, \eta) u_2 \end{aligned} \tag{5.4}$$

Equation (5.4) allows us to study the problem of body manipulation and finger repositioning by studying the problem of steering the states of the control system (5.4). This is related to the control problem we studied in Chapter 3. The u_1 input is precisely the desired object velocity $\dot{x}_d$ and u_2 corresponds to the internal motion of the system (since u_2 does not affect the location of the grasped object). Thus by specifying u_1 and u_2 as functions of time, we can use the controllers of the previous two chapters to achieve that motion.

The general case of finding $u_1(t)$ and $u_2(t)$ such that the object and the fingers move from an initial to final position (while maintaining contact) can be very difficult. We point out two interesting special cases:

1. If the hand has no redundant degrees of freedom (i.e., B_2 is not present) then it might be possible to move to an arbitrary location/grasp using only u_1. Moving just the contact location requires a carefully chosen closed loop path in x_o.

2. If we have redundant degrees of freedom, then we can move the fingers along the object while keeping the object position fixed ($\dot{x}_o = u_1 = 0$). In this case we use only the vector fields in B_2 to move the fingers.

In the second case, it is sufficient to study the control of a single finger since the fingers are decoupled if the object is held fixed. This situation has been studied by Li and Canny [59] and we review some examples from that paper in the next chapter.

5.2 Nonlinear control theory

This section collects a variety of results from differential geometry and nonlinear control theory which will prove useful in studying nonholonomic systems. There are several good references for the material presented here, although no single book is adequate. For basic definitions and concepts in differential geometry, see Boothby [9] or Spivak [88]. A good introduction to nonlinear control theory which includes many of the necessary differential geometric concepts can be found in Nijmeijer and van der Schaft [75].

We begin with a brief review of differential geometry for the purpose of fixing notation.

Manifolds, vector fields, and distributions

A *manifold* is a set M which is locally diffeomorphic to $\mathbb{R}^n$. We parameterize the manifold using a *local coordinate chart.* A chart is a pair (ϕ, U) where ϕ is a function which maps points in the open set $U \subset M$ to an open subset of $\mathbb{R}^n$. We require that overlapping charts (ϕ, U) and (ψ, V) be related in the sense that $\psi^{-1}\phi$ is a diffeomorphism where it is defined. Common examples of manifolds include S^1, the circle in $\mathbb{R}^2$, and S^2, the sphere in $\mathbb{R}^3$.

To any point on a manifold, we associate a *tangent space*, T_xM. This tangent space is the best linear approximation to the manifold at a point, and hence has the same dimension as the underlying manifold. T_xM can be identified with $\mathbb{R}^n$ and in a given set of coordinates we can represent a point in the tangent space as the point x and a vector $v \in \mathbb{R}^n$. This representation corresponds to the mental picture of attaching a tangent plane to the point $x \in M$.

A *vector field* on M is a function which assigns to each point on $x \in M$ a tangent vector $f(x) \in T_xM$. In local coordinates, we represent f as a column vector whose elements depend on x,

$$f(x) = \begin{pmatrix} f_1(x) \\ \vdots \\ f_n(x) \end{pmatrix}$$

Alternatively, if $(x_1, \cdots, x_n)$ is a set of local coordinates for M, we write

$$f(x) = f_1(x)\frac{\partial}{\partial x_1} + \cdots + f_n(x)\frac{\partial}{\partial x_n}$$

The symbol $\frac{\partial}{\partial x_i}$ is to be thought of as a basis element for the tangent space with respect to a given set of local coordinates. A vector field is smooth if each $f_i(x)$ is smooth; this can be shown to be independent of choice of coordinates.

To any vector field we define the *flow* of a vector field to represent the action of integration along a vector field. Specifically, $\phi_t^f : M \to M$ satisfies

$$\frac{d}{dt}\phi_t^f(x) = f(\phi_t^f(x)) \qquad x \in M$$

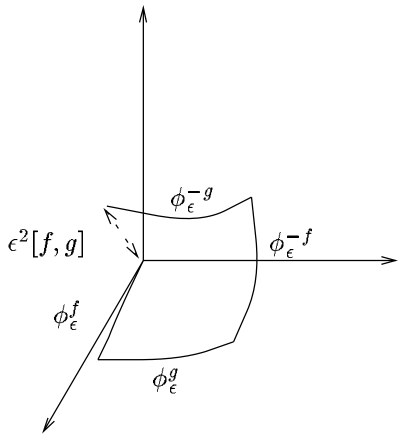


Figure 5.3: A Lie bracket motion

A vector field is *complete* if its flow is defined for all t. All differentiable vector fields are locally complete. For each given t, ϕ_t^f is a local diffeomorphism of M onto itself and $\phi_t^f \circ \phi_s^f = \phi_{t+s}^f$ for all t, s.

Given two vector fields f and g, we define the *Lie bracket* as

$$[f,g] = \frac{\partial g}{\partial x} f - \frac{\partial f}{\partial x} g$$

We can interpret this quantity using flows. Consider the flow depicted in Figure 5.3:

$$\phi_\epsilon^{-g} \circ \phi_\epsilon^{-f} \circ \phi_\epsilon^{g} \circ \phi_\epsilon^{f} = \epsilon^2[f,g] + o(\epsilon^2) \tag{5.5}$$

The Lie bracket is the infinitesimal motion that results from flowing around a square defined by two tangent vectors. If $[f,g]=0$ then f and g commute and it can be shown that the right hand side of equation (5.5) is identically zero; i.e., we return to the starting point. A *Lie product* is a nested set of Lie brackets, for example,

$$[[f,g],[f,[f,g]]]$$

A *distribution* assigns a subspace of the tangent space to each point in M in a smooth way. A special case is a distribution defined by a set of smooth vector fields, $g_1, \cdots, g_m$. In this case we define the distribution as

$$\Delta = \text{span}\{g_1, \cdots, g_m\}$$

where we take the span over the set of smooth real-valued functions on M. At any point the distribution is a linear subspace of the tangent space

$$\Delta_x = \text{span}\{g_1(x), \cdots, g_m(x)\} \subset T_x M$$

A distribution is *involutive* if it is closed under the Lie bracket:

$$\Delta \text{ involutive} \quad \Longleftrightarrow \quad \forall f, g \in \Delta, \quad [f, g] \in \Delta$$

For a finite dimensional distribution it suffices to check that the basis elements are contained in the distribution. The *involutive closure* of a distribution, denoted $\bar{\Delta}$, is the closure of Δ under bracketing. That is, $\bar{\Delta}$ is the smallest distribution containing Δ such that if $f, g \in \bar{\Delta}$ then $[f, g] \in \bar{\Delta}$.

A submanifold $N \subseteq M$ is an *integral manifold* of Δ if $\Delta_x = T_x N$ at every x. That is, the distribution spans the tangent space of the submanifold at every point. A distribution is *integrable* if there exists a manifold $N \subseteq M$ which is an integral manifold for Δ. Integral manifolds are related to involutive distributions by the following theorem:

Theorem 5.1 (Frobenius) *A distribution is integrable if and only if it is involutive.*

The following interpretation of Frobenius' theorem is also useful. If Δ is an m-dimensional involutive distribution, then locally there exist $n - m$ functions $h_i \colon M \to \mathbb{R}$ such that integral manifolds of Δ are given by the level surfaces of $h = (h_1, \cdots, h_{n-m})$. These level surfaces form a *foliation* of M. A single level surface is called a *leaf of the foliation.*

Associated with the tangent space $T_x M$ is the dual space $T_x^* M$, the set of linear functions on $T_x M$. Just as we defined vector fields on $T_x M$, on $T_x^* M$ we can define a *one-form*: for each $x \in M$, $\omega(x) \in T_x^* M$. In local coordinates we represent a smooth one form as

$$\omega(x) = \omega_1(x) dx_1 + \cdots + \omega_n(x) dx_n$$

where each ω_i is smooth. The symbols dx_i represent the basis dual to the basis $\frac{\partial}{\partial x_i}$ on $T_x M$ and are defined as

$$dx_i(\frac{\partial}{\partial x_j}) = \delta_{ij}$$

where δ_{ij} is the Kronecker delta. A one-form acts on a vector field to give a real-valued function on M,

$$\omega \cdot f = \left(\sum_i \omega_i dx_i \right) \cdot \left(\sum_j f_j \frac{\partial}{\partial x_j} \right) = \sum_i \omega_i f_i$$

Nonlinear controllability

We begin the study of controllability by converting a set of linear velocity constraints into a control system. Consider the problem of constructing a path $x(t) \in M$ between a given x_0 and x_1 subject to the constraints

$$\omega_i(x)\dot{x} = 0 \qquad i = 1, \cdots, k$$

The ω_i's are linear functions on the tangent spaces of M, i.e., one-forms. We assume that the ω_i's are smooth and linearly independent over the ring of smooth functions.

Proposition 5.2 *There exist vector fields $g_j(x)$, $j = 1, \cdots, n-k$ that annihilate the $w_i(x)$'s ($w_i(g_j) = 0$) such that the g_i's are smooth and linearly independent over the ring of smooth functions.*

Proof. Since the ω's are linearly independent, we can choose local coordinates such that

$$\omega_i = dx_i + \sum_{l=k+1}^{n} \alpha_{il} dx_l \qquad i = 1, \cdots, k$$

Set

$$g_j = \frac{\partial}{\partial x_{j+k}} + \sum_{l=1}^{k} -\alpha_{l(j+k)} \frac{\partial}{\partial x_l} \qquad j = 1, \cdots, n-k$$

The g_j's are linearly independent. Furthermore they annihilate the constraint since

$$\begin{aligned} \omega_i \cdot g_j &= (dx_i + \textstyle\sum_{l=k+1}^{n} \alpha_{il} dx_l) \cdot (\frac{\partial}{\partial x_{j+k}} + \sum_{l=1}^{k} -\alpha_{l(j+k)} \frac{\partial}{\partial x_l}) \\ &= \alpha_{i(j+k)} - \alpha_{i(j+k)} = 0 \end{aligned}$$

Smoothness of $\{g_j\}$ follows directly from smoothness of α_{ij} and hence $\{\omega_i\}$. □

We now restrict our attention to control systems of the form

$$\Sigma: \qquad \dot{x} = g_1(x)u_1 + \cdots + g_m(x)u_m \qquad \begin{array}{l} x \in M \\ u \in U \subset \mathbb{R}^m \end{array} \tag{5.6}$$

In view of the previous proposition, we assume that the g_i's are smooth, linearly independent vector fields on M and that their flows are defined for all time (i.e., g_i is *complete*). We wish to determine conditions under which we can steer from $x_0 \in M$ to an arbitrary $x_1 \in M$ by appropriate choice of $u(\cdot)$. We review the formulation of Hermann and Krener [39], specialized to systems having the form of equation (5.6).

A system Σ is controllable if for any $x_0, x_1 \in M$ there exists a $T > 0$ and $u: [0,T] \to U$ such that Σ satisfies $x(0) = x_0$ and $x(T) = x_1$. A system is *small-time locally controllable* at x_0 if we can reach nearby points in arbitrarily small amounts of time and stay near to x_0 at all times. Given an open set $V \subseteq M$, define

$$\begin{aligned} \mathcal{R}^V(x_0, T) = \{&x \in M : \text{there exists } u: [0,T] \to U \text{ that steers } \Sigma \text{ from } x(0) = x_0 \text{ to} \\ &x(T) = x_1 \text{ and satisfies } x(t) \in V \text{ for } 0 \le t \le T\} \end{aligned}$$

$\mathcal{R}^V(x_0, T)$ is the set of states which are reachable from x_0 in time T that also remain in V. We also define

$$\mathcal{R}^V(x_0, \le T) = \bigcup_{0 < \tau \le T} \mathcal{R}^V(x_0, \tau)$$

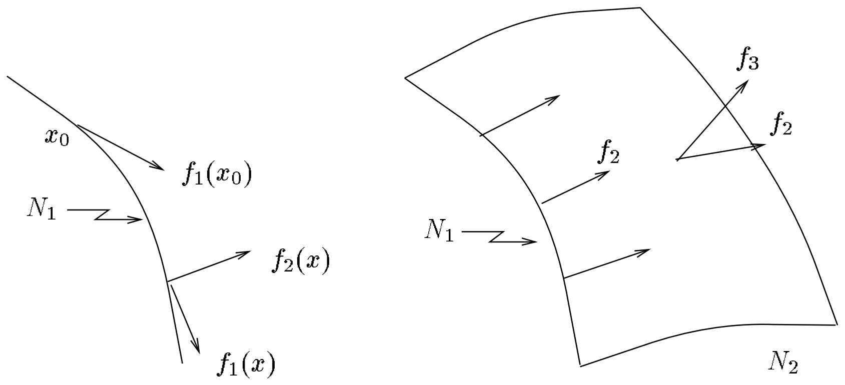


Figure 5.4: Proof of local accessibility. At each step we can find a vector field which is not in N_k.

A system is small-time locally controllable (*locally controllable* for brevity) if $\mathcal{R}^V(x_0, \leq T)$ contains a neighborhood of x_0 for all neighborhoods V of x_0 and $T > 0$. Define $\Delta = \{g_1, \cdots, g_m\}$ and let $\bar{\Delta}$ be the involutive closure of Δ. We wish to establish the following implications for a given system Σ, in a neighborhood of a point:

$$\bar{\Delta}_x = T_x M \;\Rightarrow\; \text{int}\,\mathcal{R}^V(x_0, \leq T) \neq \{\} \iff \Sigma \text{ is locally controllable}$$

Theorem 5.3 *If $\bar{\Delta}_x = T_x M$ for all x is some neighborhood of x_0, then for any $T > 0$ and neighborhood V of x_0,* $\text{int}\,\mathcal{R}^V(x_0, \leq T) \neq \{\}$.

Proof. The proof is by recursion. Choose $f_1 \in \Delta$. For $\epsilon_1 > 0$ sufficiently small,

$$N_1 = \{\phi_{t_1}^{f_1}(x_0) : 0 < t_1 < \epsilon_1\}$$

is a manifold of dimension 1 which contains points arbitrarily close to x_0; without loss of generality, we can take $N_1 \subset V$. Assume $N_k \subset V$ is a k-dimensional manifold. If $k < n$, there exists $x \in N_k$ and $f_{k+1} \in \Delta$ such that $f_{k+1} \notin T_x N_k$. If this were not so then $\Delta_x \in T_x N_k$ for any x in some open set $W \subset N_k$, which would imply $\bar{\Delta}|_W \subset TN_k$. This cannot be true since $\dim \bar{\Delta}_x = n > \dim N_k$. For ϵ_{k+1} sufficiently small

$$N_{k+1} = \{\phi_{t_{k+1}}^{f_{k+1}} \circ \cdots \circ \phi_{t_1}^{f_1}(x_0) : 0 < t_i < \epsilon_i,\ i = 1, \cdots, k+1\}$$

is a $k+1$ dimensional manifold. Since ϵ can be made arbitrarily small, we can assume $N_{k+1} \subset V$.

If $k = n$, $N_k \subset V$ is an n-dimensional manifold and by construction $N_k \subset \mathcal{R}^V(x_0, \leq \epsilon_1 + \cdots + \epsilon_n)$. Hence $\mathcal{R}^V(x_0, \epsilon)$ contains an open set. By restricting each $\epsilon_i \leq T/n$, we can find such an open set for any $T > 0$. This proof is illustrated in Figure 5.4. □

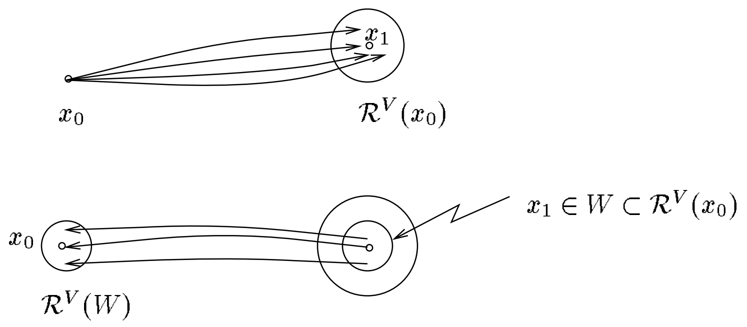


Figure 5.5: Proof of local controllability. To show $\mathcal{R}^V(x_0)$ contains a neighborhood of the origin, we move to any point x_1 and map a neighborhood of x_1 to a neighborhood of x_0 by reversing our original path.

Theorem 5.4 int $\mathcal{R}^V(x_0, \leq T) \neq \{\}$ *for all neighborhoods* V *of* x_0 *and* $T > 0$ $\Longleftrightarrow$ Σ *is locally controllable at* x_0*.*

Proof. The sufficiency follows from the definition of locally controllable. To prove necessity, we need to show that $\mathcal{R}^V(x_0, \leq T)$ contains a neighborhood of x_0. Choose a piecewise constant $u\colon [0, T/2] \to U$ such that u steers x_0 to some $x_1 \in \mathcal{R}^V(x_0, \leq T/2)$ and $x(t) \in V$. Let ϕ^u be the flow corresponding to this input (as given in the proof of the previous theorem). Since Σ is symmetric, we can flow backwards from x_1 to x_0 using $u'(t) = u(T/2 - t)$. The flow corresponding to u' is $(\phi^u)^{-1}$. By continuity of the flow, there exists $W \subset \mathcal{R}^V(x_0, T/2)$ such that $x_1 \in W$ and $\phi_t^{-1}(W) \subset V$ for all t. Furthermore, $(\phi^u_{T/2})^{-1}(W)$ is a neighborhood of x_0. It follows that $\mathcal{R}^V(x_0, \leq T)$ contains a neighborhood of x_0 since we can concatenate the inputs which steer x_0 to $x_1 \in W$ with u' to obtain an open set containing x_0. This is illustrated in Figure 5.5. □

Nonlinear stabilization

We now consider a more general control system which contains a *drift* vector field f:

$$\dot{x} = f(x) + g(x)u \qquad x \in M, u \in U \tag{5.7}$$

Again f and $\{g_i\}$ are smooth vector fields and $\{g_i\}$ has full rank. Suppose that our goal is to move the system (5.7) from one equilibrium point, (x_0, u_0), to another, (x_1, u_1). One particularly simple method of accomplishing this is to stabilize the system about x_1. That is, we choose a feedback law $u = k(x)$ such that

$$\dot{x} = f(x) + g(x)k(x) \tag{5.8}$$

is asymptotically stable at x_1. This feedback law solves the path planning problem (locally), although the resulting trajectory may not reach x_1 in finite time. In practice this is a reasonable limitation.

For the case where $f(x) \equiv 0$, we cannot use this simple method unless $m = n$ (same number of inputs as outputs). As an indication of this, we note that the linearization of the control system (5.7) about an equilibrium point $(x_0, 0)$ is given by

$$\dot{x} = \frac{\partial f}{\partial x}(x_0)x + g(x_0)u =: Ax + Bu$$

If $f(x) \equiv 0$ then $A = 0$ and the linearized system is not controllable unless $B = g(x_0)$ has rank n. Indeed, the linearized system is constrained to lie on the hyperplane defined by the columns of B.

Although the linearized system is degenerate, there is still a possibility of stabilizing the system using nonlinear feedback. We now show that there is *no* smoothly stabilizing feedback law for a nonholonomic system without drift. The proof relies on the following theorem, due to Brockett [11]. Define B_ϵ to be a ball of radius $\epsilon > 0$ containing the origin.

Theorem 5.5 *$\dot{x} = a(x)$ is asymptotically stable only if $a(B_\epsilon)$ contains a neighborhood of the origin for all $\epsilon > 0$.*

Proof. The proof given here is due to Sontag [87]. Let $\Phi(t, x)$ be the flow of a point x along a. Construct a homotopy $H\colon B_\epsilon \times [0,1] \to B_\epsilon$

$$\begin{aligned} H(x,s) &= \tfrac{1}{s}(\Phi(\tfrac{s}{1-s}, x) - x) \\ H(x,0) &= \lim_{s\to 0} \frac{d}{ds}\Phi(\frac{s}{1-s}, x) = f(x) \\ H(x,1) &= -x \end{aligned}$$

Therefore a is homotopic to $-x$ and hence their degrees with respect to zero are equal:

$$\deg\, a = \deg(-x) = (-1)^n$$

Since $\deg\, a \neq 0$, it follows that $a(x) = p$ has at least one solution for all p in a neighborhood of 0. $a(B_\epsilon)$ therefore contains a neighborhood of 0. □

Corollary 5.5.1 *There exists a smooth control law $u = k(x)$ which stabilizes (5.7) to an equilibrium point 0 only if the image of $F(x,u) = f(x) + g(x)u$ contains a neighborhood of the origin for every $U \ni (0,0)$.*

Corollary 5.5.2 *If $f(x) \equiv 0$ and $m < n$, there exists no smooth control law which stabilizes (5.7) to an equilibrium point.*

Proof. Let $v \neq 0$ be orthogonal to $g(x_0)$. Choose local coordinates such that

$$g_i(x) = \frac{\partial}{\partial x_i} + \sum_{j=m+1}^{n} a_{ij}(x)\frac{\partial}{\partial x_j}$$

In these coordinates $v = \sum_{j=m+1}^{n} v_j \frac{\partial}{\partial x_j}$ and hence $v \notin \text{span}\{g_i\}$. Corollary 5.5.1 guarantees the lack of a smoothly stabilizing control law. □

Corollary 5.5.2 implies that we cannot solve the nonholonomic motion planning problem by using smooth state feedback to stabilize the system about the goal. It also limits the results that can be obtained by any method which uses smooth feedback. In particular, any motion planning algorithm which is based upon smooth feedback will fail to solve the problem at some points. This is a local result, in the sense that given *any* neighborhood of the goal, there exists points arbitrarily close to the goal which cannot be steered to the goal by the algorithm.

Finally we note that the lack of stabilizing controllers holds not only for smooth (C^∞) control laws but for C^1 control laws as well. The degree theory proof of Theorem 5.5 relies only on the continuity of $a(x)$ and the existence of the flow Φ. Therefore any controller which stabilizes a nonholonomic system to a point must necessarily lose differentiability at points arbitrarily close to the equilibrium point.

5.3 Classification

We now develop some concepts which allow us to classify nonholonomic systems. A more complete treatment can be found in the work of Vershik [30, 93]. Basic facts concerning Lie algebras are taken from Varadarajan [92]. Let $\Delta = \text{span}\{g_1, \cdots, g_m\}$ be the distribution associated with the control system (5.6). Define $G_1 = \Delta$ and

$$G_i = G_{i-1} + [G_1, G_{i-1}]$$

where

$$[G_1, G_{i-1}] = \text{span}\{[g, h] : g \in G_1, h \in G_{i-1}\}$$

The set of all G's defines the *filtration* associated with a distribution. Each G_i is spanned by the input vector fields plus the vector fields formed by taking up to $i-1$ Lie brackets. The Jacobi identity implies $[G_i, G_j] \subset [G_1, G_{i+j-1}] \subset G_{i+j}$.

A filtration is *regular* in a neighborhood U of x_0 if

$$\text{rank } G_i(x) = \text{rank } G_i(x_0) \qquad \forall x \in U$$

We say a system is regular if the corresponding filtration is regular. If a filtration is regular, then at each step of its construction, G_i either gains dimension or the construction terminates. If rank G_{i+1} = rank G_i then G_i is involutive and hence $G_{i+j} = G_i$ for all $j \geq 0$. Clearly rank $G_i \leq n$ and hence if a filtration is regular, then there exists an integer $p < n$ such that $G_i = G_{p+1}$ for all $i \geq p+1$. We refer to p as the *degree of nonholonomy* of the distribution.

For a regular system, Chow's theorem is particularly easy to prove.

Theorem 5.6 *For a regular system, a path exists between two arbitrary points in an open set $U \subset M$ if and only if $G_p(x) = T_xM \approx \mathbb{R}^n$ for all $x \in U$.*

Proof. (Necessity) Suppose that $G_p(x) = \mathbb{R}^k$ for $k < n$. Since the system is regular, G_p is involutive and Frobenius' theorem implies that G_p is integrable. Let N be the integrable submanifold for G_p, of dimension k. Since all trajectories of the system are confined to N, any $x \in M$ such that $x \notin N$ is not reachable.

(Sufficiency) Given two points $x_0, x_1 \in M$, we can connect the points with a line $x_0 + t(x_1 - x_0)$. At each point on this line, the reachable states form an open set (since the system is controllable). Since the line has finite length, it is a compact set (in the relative topology) and hence we can pick a finite subcover of reachable sets. Since a path exists between each intersecting pair of this finite subcover, a path between x_0 and x_1 exists by concatenating path segments. □

A system (or distribution) satisfying the conditions of the theorem is said to be *maximally nonholonomic.* This version of Chow's theorem is considerably weaker than our previous version, which holds for non-regular systems. If a regular system is not maximally nonholonomic, then by Frobenius' theorem we can restrict ourselves to a manifold on which the system is maximally nonholonomic.

It is also useful to record the dimension of each G_i. For a regular system, we define the *growth vector* $r \in \mathbb{Z}^{p+1}$ as

$$r_i = \text{rank } G_i$$

We define the *relative growth vector* $\sigma \in \mathbb{Z}^{p+1}$ as $\sigma_i = r_i - r_{i-1}$ and $r_0 := 0$. The growth vector for a system is a convenient way to represent information about the associated control Lie algebra. For a distribution with finite rank, the growth vector is bounded from above at each step. To properly determine this bound, we must determine the rank of G_i taking into account skew-symmetry and the Jacobi identity. A careful calculation [85] gives

$$\bar{\sigma}_i = \frac{1}{i}\left((\bar{\sigma}_1)^i - \sum_{j|i,j<i} j\bar{\sigma}_j\right) \qquad i > 1 \tag{5.9}$$

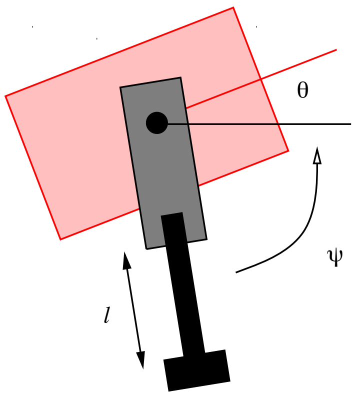
where $\bar{\sigma}_i$ is the maximum relative growth at the i^{th} stage and $j|i$ means all integers j such that j divides i. If $\sigma_i = \bar{\sigma}_i$ for all i, we say Δ has *maximum growth.*

Examples

To illustrate the classification of nonholonomic systems, we present several detailed examples. These examples are used in later chapters as a basis for testing planning algorithms.

Example 1 – Hopping robot

As our first example, we consider a hopping robot as shown in Figure 5.3. This robot consists of a body with an actuated leg that can rotate and extend. We



are interested in studying the robot when it is in free flight. The "constraint" on the system is conservation of angular momentum.

The (ψ, l, θ) be the body angle, leg extension and leg angle of the robot. For simplicity, we take the body mass to be 1 and concentrate the mass of the leg, m_l, at the foot. The upper leg length is also taken to be 1, with l representing the extension of the leg past this point. Since we control the leg angle and extension directly, we choose them as our inputs. The angular momentum of the robot is given by

$$\dot{\theta} + m_l(l+1)^2(\dot{\theta} + \dot{\psi}) = 0$$

Thus our equations become

$$\begin{aligned} \dot{\psi} &= u_1 \\ \dot{l} &= u_2 \\ \dot{\theta} &= -\tfrac{m_l(l+1)^2}{1+m_l(l+1)^2} u_1 \end{aligned}$$

In vector field form we have

$$\begin{aligned} g_1 &= \tfrac{\partial}{\partial \psi} - \tfrac{m_l(l+1)^2}{1+m_l(l+1)^2} \tfrac{\partial}{\partial \theta} \\ g_2 &= \tfrac{\partial}{\partial l} \\ g_3 = [g_1, g_2] &= \tfrac{2m_l(l+1)}{(1+m_l(l+1)^2)^2} \tfrac{\partial}{\partial \theta} \end{aligned}$$

In a neighborhood of $l = 0$, $\{g_1, g_2, g_3\}$ is full rank and hence the hopping robot has degree of nonholonomy 3 with growth vector $(2, 3)$.

Example 2 – Automobile

Figure 5.6 shows an automobile with front and rear tires. The rear tires are aligned with the car while the front tires are allowed to spin about the vertical axes. To simplify the derivation, we model the front and rear pairs of wheels as

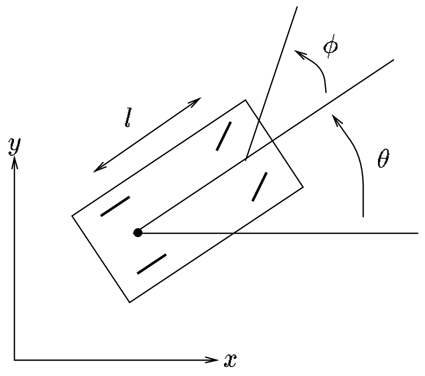


Figure 5.6: Front wheel drive cart. The configuration of the cart is determined by the Cartesian location of the back wheels, the angle the car makes with the horizontal and the steering wheel angle relative to the car body. The two inputs are the velocity of the front wheels (in the direction the wheels are pointing) and the steering velocity. The rear wheels of the cart are always aligned with the cart body and are constrained to move along the line in which they point or rotate about their center.

single wheels at the midpoints of the axles. The constraints on the system arise by allowing the wheels to roll and spin, but not slip.

Let (x, y, ϕ, θ) denote the configuration of the car, parameterized by the location of the rear wheel(s), the angle of the car body with respect to the horizontal (θ), and the steering angle with respect to the car body (ϕ). The constraints for the front and rear wheels are formed by writing the sideways velocity of the wheels:

$$\begin{aligned} \tfrac{d}{dt}(x + l\cos\theta)\cdot\sin(\theta+\phi) - \tfrac{d}{dt}(y + l\sin\theta)\cdot\cos(\theta+\phi) &= 0 \\ \dot{x}\sin\theta - \dot{y}\cos\theta &= 0 \end{aligned}$$

Written as one forms we have

$$\begin{aligned} \omega_1 &= \sin(\theta+\phi)dx - \cos(\theta+\phi)dy - l\cos\phi d\theta \\ \omega_2 &= \sin\theta dx - \cos\theta dy \end{aligned}$$

Converting this to a control system gives

$$\begin{aligned} \dot{x} &= \cos\theta\, u_1 \\ \dot{y} &= \sin\theta\, u_1 \\ \dot{\phi} &= u_2 \\ \dot{\theta} &= \tfrac{1}{l}\tan\phi\, u_1 \end{aligned}$$

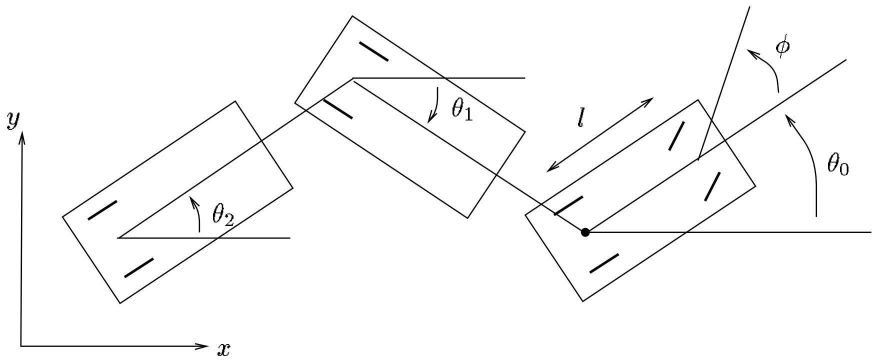


Figure 5.7: Front wheel drive cart with trailers. The trailer configuration is described the the angle the trailer makes with the horizontal, ψ. The rear wheels of the trailer are fixed and constrained to move along the line in which they point or rotate about their center. The inputs to the system are the inputs to the tow car: the driving velocity (of the front wheels) and the steering velocity. This system is an example of a fourth degree system; higher degree systems can be generated by adding extra trailers.

To calculate the growth vector, we build the filtration

$$\begin{aligned}
g_1 &= \cos\theta\frac{\partial}{\partial x} + \sin\theta\frac{\partial}{\partial y} + \frac{1}{l}\tan\phi\frac{\partial}{\partial\theta} \\
g_2 &= \frac{\partial}{\partial\phi} \\
g_3 = [g_1, g_2] &= \frac{-1}{l\cos^2\phi}\frac{\partial}{\partial\theta} \\
g_4 = [g_1, g_3] &= \frac{-\sin\theta}{l\cos^2\phi}\frac{\partial}{\partial x} + \frac{\cos\theta}{l\cos^2\phi}\frac{\partial}{\partial y} \\
g_5 = [g_2, g_3] &= \frac{-2\tan\phi}{l\cos^2\phi}\frac{\partial}{\partial\theta}
\end{aligned}$$

$\{g_1, g_2, g_3, g_4\}$ are linearly independent when $\phi \neq \pm\pi$. Thus the system has degree of nonholonomy 2 with growth vector $r = (2, 3, 4)$ and relative growth vector $\sigma = (2, 1, 1)$. The system is regular away from $\phi = \pm\pi$, where g_1 is undefined.

Example 3 – Car with N trailers

Figure 5.7 shows a car with N trailers attached. We attach the hitch of each trailer to the center of the rear axle of the previous trailer. The wheels of the individual trailers are aligned with the body of the trailer. The constraints are again based on allowing the wheels only to roll and spin, but not slip. The dimension of the state space is $4 + N$ with 2 controls.

We parameterize the configuration by the states of the automobile plus the angles of each of the trailers with respect to the horizontal. For consistency we will write θ_0 for the angle of the car. Calculation of the constraints becomes tedious since we have to write the velocity of the wheels of each trailer, which depend on all previous trailers. Instead, we choose to use the same inputs as the automobile and calculate the effect on the trailer angles.

At each trailer, we can write the hitch velocity as the sum of two components: the velocity in the direction the trailer is pointing and its perpendicular. The perpendicular component causes the trailer to spin. Letting v_{i-1} be the forward velocity of the previous trailer, we have

$$\begin{aligned}\dot{\theta}_i &= \tfrac{1}{d_i}\sin(\theta_{i-1}-\theta_i)v_{i-1}\\ v_i &= \cos(\theta_{i-1}-\theta_i)v_{i-1}\end{aligned}$$

Aggregating these equations gives

$$\begin{aligned}\dot{x} &= \cos\theta_0\, u_1\\ \dot{y} &= \sin\theta_0\, u_1\\ \dot{\phi} &= u_2\\ \dot{\theta}_0 &= \tfrac{1}{l}\tan\phi\, u_1\\ \dot{\theta}_i &= \tfrac{1}{d_i}\left(\prod_{j=1}^{i-1}\cos(\theta_{j-1}-\theta_j)\right)\sin(\theta_{i-1}-\theta_i)\, u_1\end{aligned} \tag{5.10}$$

The filtration corresponding to the N trailer problem is very complex. For small values of N, controllability can be verified directly. For the general case, a very detailed and well-organized calculation by Laumond [57] shows that the system is controllable with degree of nonholonomy $N+2$ and relative growth vector $\sigma = (2, 1, \cdots, 1)$.[1]

[1]Laumond uses a slightly different system, obtained by ignoring ϕ and choosing u_1 and $u_1 \tan\phi$ as inputs. Since setting $u_1 = 0$ allows us to steer ϕ independently, controllability for the system given here follows from Laumond's result.

5.4 Methods in nonholonomic path planning

We now survey some of the techniques which have been used in planning paths for nonholonomic systems. There are two basic classes of algorithms currently available: motion planning and path planning. The distinction is in the presence or absence of obstacles. For our purposes, we shall call an algorithm a *motion* planning algorithm if it attempts to generate feasible trajectories without accounting for obstacles explicitly. A *path* planning algorithms is one in which obstacles are included in an integral way. As we shall see, many motion planning algorithms can be extended to account for obstacles if they satisfy certain local properties.

Optimal control

Perhaps the most well-formulated method for finding trajectories of a general control system is to use optimal control. By attaching a cost to each trajectory, we can limit our search to trajectories which maximize the cost function. Typical cost functions might be the length of the path (in some appropriate metric), the control cost, or the time required to execute the trajectory.

A very general approach to the optimal control problem was developed by Pontryagin [78] using the maximum principle. The maximum principle gives a set of differential equations which are satisfied by all extremal curves of a cost function. It is a generalization of the Euler-Lagrange equations which allows for input constraints, state space constraints, and free or fixed end conditions.

The application of the maximum principle is straightforward. We outline the basic approach here, for the special case of nonholonomic path planning. Let $L(x, \dot{x}, t)$ be the infinitesimal cost associated with a feasible path $x(\cdot)$. The cost associated with $x(\cdot)$ is

$$\rho(x) = \int_0^T L(x, \dot{x}, t) dt$$

Define

$$\mathcal{H}(x, p, u) = \dot{p}^T g(x) u - L(x, g(x) u, t)$$

where $p \in \mathbb{R}^n$ is a set of *costate* variables. The maximum (or minimum) principle states that we can achieve the extremal curve by using pointwise minimization on $\mathcal{H}$. Let

$$H(x, p) = \min_u \mathcal{H}(x, p, u)$$

Then the extremal curves for the fixed end point problem satisfy

$$\begin{aligned} \dot{x} &= \frac{\partial H}{\partial p} \qquad x(0) = x_0, \quad x(1) = x_1 \\ \dot{p} &= -\frac{\partial H}{\partial x} \end{aligned}$$

Even for relatively low dimensional problems, the differential equations generated by applying the maximum principle are not solvable in closed form. Thus we do not have an explicit characterization of the extremal curves. One method of getting around this is to simulate the differential equations using different initial conditions for the costate variables. This technique is called *shooting.* Unfortunately, since there are as may costates as states, the cost of this search is $o(\frac{1}{\epsilon^n})$ where ϵ is the grid size for the initial conditions and n is the number of states.

If the system has bounds on the magnitudes of the inputs, we can ask to solve the motion planning problem in *minimum time.* It is well-known that for many problems, the time-optimal paths consist of saturating the inputs at all times (this is often referred to as *bang-bang* control). The inputs may change between one set of values and another at a possibly infinite number of *switching times.* Choosing an optimal trajectory is then equivalent to choosing the switching times and the values of the inputs between switching times.

Piecewise constant inputs

Related to the bang-bang trajectories of optimal control, it is also possible to steer nonholonomic systems using piecewise constant inputs. The motivation in using piecewise constant controls is that we do not need to bookkeep the evolution of the costate variables. Rather, we simply search or otherwise choose a sequence of constant inputs which steer the system as desired.

Perhaps the most naive way of using constant inputs is to pick a time interval and generate a graph by applying all possible sequences of inputs. Each node on the graph corresponds to a configuration and branches indicate the choice of a fixed control over the time interval. The size of the graph grows as m^d where m is the number of input combinations considered at each step and d is the depth of the graph. Since we do not know how long to search, the memory required by such an algorithm can be very large. Also, we are likely not to hit our goal exactly, so some post-processing must be done if exact maneuvering is needed.

An improved version of this basic approach has been used by Barraquand and Latombe [4]. Part of their approach is to grid the state space and keep track of parallelepipeds which have already been reached by some trajectory. In this way, they must bookkeep at most $\sigma(\frac{1}{\epsilon^n})$ points. For long motions this method is vastly superior to the naive method originally considered. Unfortunately, their algorithm only works in sufficiently small dimensional state spaces ($n = 4$ seems to be a practical upper limit).

Recently a very elegant and general motion planning method using piecewise constant inputs has been developed by Lafferriere and Sussmann [54]. They consider as motivation the case of a *nilpotent* system. Briefly, a system is nilpotent of order k if all the brackets with more than k terms vanish. We shall study these more closely in the next chapter. The advantage of nilpotent Lie algebras is that certain computations are greatly simplified, as we shall see. The main tool in their method is the Baker-Campbell-Hausdorff formula.

The Baker-Campbell-Hausdorff formula describes the composition of flows as a single flow:

$$\phi^{X_1} \circ \phi^{X_2} = \phi^{X_1 + X_2 + \frac{1}{2}[X_1, X_2] + \cdots}$$

Their method consists of first converting a desired motion into a flow of the form:

$$\phi^{\alpha_1 X_1 + \alpha_2 X_2 + \alpha_3 [X_1, X_2] + \cdots}$$

They use the Baker-Campbell-Hausdorff formula to choose piecewise constant inputs that achieve this flow. Since the system is nilpotent, the Baker-Campbell-Hausdorff formula has a finite number of terms and it can be shown that the process terminates.

The technique used by Lafferriere and Sussmann is an important one which we shall see again in the next chapter. They begin by choosing inputs which steer the system in the input directions to their final values. They then construct inputs which are closed loops in the input directions but result in motion in the first level of Lie brackets. This is precisely the motion we used in equation (5.5). The error terms can all be written in terms of higher level brackets. We then proceed to find motions which are closed in the input and first-level bracket directions that move the second level bracket directions to their desired values. In the case that the system is nilpotent, we will eventually reach our goal since at some point the remainder terms in the Baker-Campbell-Hausdorff expansion will be zero.

The restriction of using a nilpotent system can be relaxed. Lafferriere and Sussmann describe a method which works for general nonholonomic systems. The technique is to ignore the higher order brackets and pretend that the system is nilpotent. It can be shown that if the initial and final points are close, then the resulting piecewise constant input moves the original system closer to the goal by at least half. By breaking the path up into small pieces, we can move arbitrarily close to the goal with repeated applications of the algorithm.

Unfortunately, the paths generated for non-nilpotent systems can consist of large numbers of segments. One possible solution is to pick a change of coordinates and state feedback such that the new system is nilpotent. This is much like the technique of exact linearization—given a nonlinear system, we convert it to a linear system via feedback transformation and then apply linear design techniques. We return to this topic in the next chapter.

Canonical paths

A third approach to solving the nonholonomic path planning problem is by choosing a family of paths which can be used to produce desired motions. For example, we might consider paths for a car which cause a net rotation of any angle and a translation in the direction that the car is facing. We can then move to any configuration by reorienting the car, driving forward, and again reorienting the car. The path used to cause a net rotation might consist of a set of parameterized piecewise constant inputs or a heuristic trajectory.

The set of canonical paths used for a given problem is usually specific to that problem. In some cases the paths may be derived from some unifying principle. For example, if we could solve the optimal control problem in closed form, these optimal paths would form a set of canonical paths. In the case of time-optimal control, we might consider paths corresponding to saturated inputs as canonical paths, but since it is not clear how to combine these paths to get a specific motion, we distinguish these lower level paths from canonical paths.

One system which has generated several different families of canonical paths is the automobile. Laumond introduced a set of canonical paths to achieve a crabwise (sideways) motion and rotation for a mobile robot with restricted turning radius [55]. Using a similar system description, Dubins was able to show that the optimal length paths for a car driving at constant velocity had a particularly nice form: three segments with each segment being a curve at maximum turning radius or a straight segment [27]. Reeds and Sheep have extended Dubins work to consider a car which is capable of driving forwards and backwards [80]. They show that the shortest paths are composed of at most five segments of arcs or lines with at most two changes in direction.

Canonical paths have also been used by Li and Canny to study the motion of a spherical fingertip on an object [59]. They use triangular patches on the sphere to generate motion on the object. Although this path is not optimal, it does consists of piecewise constant input segments. Similar paths have been used by Li, Montgomery and Raibert to steer the hopping robot in midair [61], although those methods might be more properly considered piecewise constant paths.

Gradient and metric methods

For conventional robot manipulators, path planning can be performed with the use of potential fields, as described by Khatib [51]. Given a start and goal position in configuration space, one can simulate a potential well by attaching a large negative potential to the goal and positive potentials to obstacles. The robot configuration is modeled as a positive point charge in this potential field and the resulting motion of the particle gives a path for the system. If the particle reaches the goal then we have planned a collision free path.

Variations of the potential field method overcome some of its limitations. The work of Koditschek and Rimon [53] derives potential fields for a given configuration space in which there is only a single local minimum: the goal position. Furthermore, these potential functions give a control law for the mechanical system. Thus we are given a path implicitly in the form of a control law which causes the system to move from start to goal.

These methods and others rely on the use of a metric to determine how far away obstacles are. The use the gradient of the metric to determine locally which way to move. For a nonholonomic system, there is no natural choice for the distance between two points. Since we cannot move directly in some directions, a planner based on the usual Euclidean metric will not generate feasible paths. There is a class of metrics for nonholonomic systems called *Carnot-Caratheodory* metrics. These are also referred to as singular metrics or sub-Riemannian metrics in the literature. For a system which is maximally nonholonomic, we define the distance between two points as

$$d(x_0, x_1) = \text{length of the shortest feasible path connecting } x_0 \text{ to } x_1$$

(the length of a path being taken with respect to the configuration space metric). An equivalent formulation, more natural from the point of view of control theory, is to define

$$d(x_0, x_1) = \min_u \int_0^T ||u||^2 dt \tag{5.11}$$

where u steers the system from x_0 to x_1. A very thorough discussion of Carnot-Caratheodory spaces can be found in a paper by Strichartz [90].

Algorithms based on metric properties have only recently begun to appear. This may be due to their singular nature—algorithms can fail to converge when all feasible motions cause the robot to move away from the goal. Laumond et al. have used a modified gradient to optimize feasible paths in the presence of obstacles [58]. Li and Gurvits have proposed a regularization based approach to motion planning in which they construct a metric which weights constraint directions by $\frac{1}{\epsilon}$ [62]. As $\epsilon \rightarrow 0$ the cost of moving in these directions grows. By iteratively solving for minimum length paths and letting $\epsilon \rightarrow 0$, they claim to construct asymptotically feasible paths (the method has yet to be implemented and tested).

Another example of gradient methods applied to nonholonomic problem is the work of Nakamura and Mukherjee [72]. They attempt to steer a system by choosing a Lyapunov function which attains its minimum at the goal configuration and then deriving a control law which causes the Lyapunov function to decrease. As noted in their paper, they are not able to find controllers which cause the Lyapunov function to be *strictly* decreasing. Consequently, there exist points for which their control law does not steer the system to the goal. This is to be expected since their controller is implemented using smooth feedback and, by Corollary 5.5.2, we know that it is not possible to asymptotically stabilize the goal in this manner.

Carnot-Caratheodory metrics are closely related to optimal control; the length in equation (5.11) has the same form as the cost functions used in the maximum principle. This connection has been studied by Brockett [10], who was able to show that for a certain class of systems with degree of nonholonomy 1, the minimum length paths correspond to using sinusoidal inputs at integrally related frequencies. His analysis is the starting point for the methods presented in the next chapter; a detailed discussion is presented there.

Obstacle avoidance

We now wish to consider methods of nonholonomic motion planning in the presence of obstacles. That is, we would like to find a *feasible* path between two points x_0 and x_1 which avoids obstacles. As we mentioned earlier, canonical path planners assume arbitrary motions are possible in the interior of the free configuration space. This assumption is not valid for a nonholonomic system. However, the problem can be decomposed in two steps:

1. generate a trajectory which ignores the constraints, but avoids obstacles
2. generate a feasible trajectory which is arbitrarily close to the obstacle avoiding trajectory

The proof that we can decompose the problem in this way is based on the following lemma, which follows directly from the definition of local controllability:

Lemma 5.7 *Let Σ be a maximally nonholonomic system (controllable). Given any $\epsilon > 0$, there exists $\delta > 0$ such that for any $x_1 \in B_\delta(x_o)$ there is a feasible path $x(\cdot)$ satisfying $x(0) = x_0$, $x(1) = x_1$, and $x(t) \in B_\epsilon(x_o)$ for all t.*

To construct a feasible path from a non-feasible one, we locally approximate the given path. In the presence of obstacles we can use Lemma 5.7 to insure that a feasible path exists which does not intersect the obstacles.

Theorem 5.8 *Given any path $x_u(\cdot)$ between x_0 and x_1 which is in the interior of the free space, there exists a feasible path between x_0 and x_1.*

Proof. Cover the path with a ball at each point which is contained in the interior of the free space. Since the path has finite length, there exists a finite subcover. Let ϵ_0 be the size of the smallest ball in the finite subcover. At each point in the path we can now place a ball of radius δ_0 which stays inside the free configuration space. Choosing a finite subcover of this new set of balls, we can choose intermediate points which generate a feasible path. This path approximates the original path. $\Box$

Much stronger versions of Theorem 5.8 are available. The work of Gershkovich [31, 30] shows that locally the set of reachable states for a regular system are contained in a parallelepiped with sides whose length is exponential in the level of bracketing necessary to move in that direction. Furthermore, a smaller parallelepiped of the same shape is contained within the reachable states.

Theorem 5.8 guarantees that we can approximate any trajectory by a feasible one. However, the method that we use to generate our nonholonomic motions may not do so in such a way as to avoid obstacles. In order to insure that a method is complete, we must insure that Lemma 5.7 holds when we search only over paths in a given class. We say a nonholonomic motion planning algorithm is *locally controllable* if Lemma 5.7 holds. Any algorithm which is locally controllable can automatically be extended to work in the presence of obstacles. This property has also been studied by Laumond [45, 58]

As an example, we again consider the automobile. Suppose we use the canonical paths formed by reorienting the car to point to the goal position, driving to that position, and again reorienting the car to be aligned with the desired final configuration. It is easy to see that this method is not locally controllable: to move in a sideways direction we must spin $\frac{\pi}{2}$ radians regardless of how far we desire to move. A similar argument shows that the Dubins generated paths are also not locally controllable. However, the Reeds-Shepp paths *are* locally controllable. A proof of this was given in [6] and an algorithm which uses the two-step approach described above has been implemented [56].

Even if an algorithm is locally controllable, the resulting paths can be intuitively unappealing. The final path has a strong dependence on the initial path chosen by the obstacle avoidance planner. There are two methods that can be applied in an attempt to improve the final paths. The first is to give the obstacle avoidance planner information about directions in which it is difficult to move. A second is to use post-processing to smooth the path while keeping it feasible. This latter method was implemented in [56] by attempting to replace sections of the feasible paths with shorter segments which did not intersect obstacles. Although the method was heuristic, the results were quite good. An example of the algorithm is shown in Figure 5.8.

A slightly different approach was used by Barraquand and Latombe in avoiding obstacles for car-like robots [4]. They grid the state space and move between parallelepipeds by considering piecewise constant trajectories integrated for unit time. Given a starting point, they generate the configurations reached by driving forwards and backwards with the steering wheel turned hard left, hard right, or straight. If the new location intersects an obstacle or lies in a parallelepiped which has already been visited, it is discarded. Otherwise, it is added to the list of points to be expanded and the corresponding parallelepiped is marked as visited. In this way obstacles are incorporated at the same time as feasible trajectories are generated and are used to truncate the search graph.

Other methods

There are many other methods which have been applied to nonholonomic problems, often in the context of mobile robots.

Jacobs used the optimal paths of Dubins to develop an algorithm for steering a mobile robot in the presence of polygonal obstacles [46]. The basic principle was that between contact points, the trajectories for the robot had the form of an path. By searching for paths over the edges of obstacles, Jacobs was able to generate an efficient planner which could find globally optimal paths for mobile robots which do not reverse direction.

Bloch and McClamroch have studied the problem of stabilizing a nonholonomic system with drift to a manifold of equilibrium points [7, 8]. They consider the use of smooth control laws, and hence the dimension of the stabilizable manifolds is bounded from below. This is an extension of the smooth stabilizability result given in Corollary 5.5.2.

Another related set of work is planning trajectories for robots with dynamic constraints. Typically these constraints are on the maximum torques that can be generated by the manipulator, but it may be possible to extend those methods to systems with nonholonomic constraints. Representative papers include [17, 38] and the references therein.

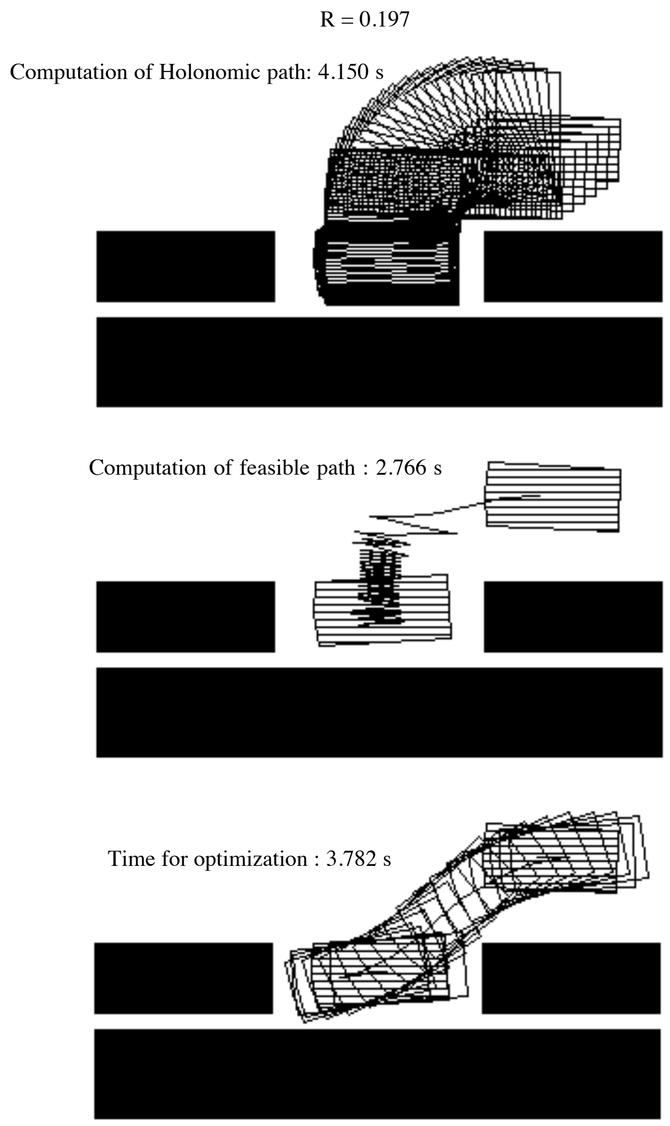

R = 0.197
Computation of Holonomic path: 4.150 s
Computation of feasible path : 2.766 s
Time for optimization : 3.782 s

Figure 5.8: A sample path calculated by the planner of Jacobs et al. [56]. (Figure courtesy of Paul Jacobs)

Chapter 6

Steering nonholonomic systems using sinusoids

In this chapter we investigate methods for steering systems with nonholonomic constraints between arbitrary configurations. Early work by Brockett derives the optimal controls for a set of canonical systems in which the tangent space to the configuration manifold is spanned by the input vector fields and their (first order) Lie brackets. Using Brockett's result as motivation, we derive suboptimal trajectories for systems which are not in canonical form and consider systems in which it takes more than one level of bracketing to achieve controllability. These trajectories use sinusoids at integrally related frequencies to achieve motion at a given bracketing level. Examples and simulation results are presented.

We consider systems of the form

$$\dot{x} = g_1(x)u_1 + \cdots + g_m(x)u_m \qquad x \in \mathbb{R}^n, \quad u \in \mathbb{R}^m \tag{6.1}$$

with $\{g_i\}$ a set of smooth, linearly independent vector fields in some neighborhood of the origin. We also assume that the system is regular (as defined in Section 5.3) and hence has a well-defined degree of nonholonomy and growth vector.

6.1 First degree systems

Control systems in which the first level of brackets together with the input vector fields span the tangent space at each configuration arise in many areas. In classical mechanics, systems with growth vector $r = (n-1, n)$ are called contact structures [1]. A version of the Darboux theorem asserts that for these systems the corresponding constraint can be written as

$$dx_3 = x_2 dx_1$$

(using the notation of exterior differential forms). In $\mathbb{R}^3$ and using control system form, this becomes

$$\begin{array}{rcl} \dot{x}_1 & = & u_1 \\ \dot{x}_2 & = & u_2 \\ \dot{x}_3 & = & x_2 u_1 \end{array} \tag{6.2}$$

Brockett considered a more general version of this system [10]; we review his results here. Consider a control system as in equation (6.1) that is maximally nonholonomic with growth vector $(m, n) = (m, \frac{m(m+1)}{2})$. We would like to find an input $u(t)$ on the interval 0 to 1 which steers the system between an arbitrary initial and final configuration and minimizes

$$\int_0^1 |u|^2 dt$$

This problem is related to finding the geodesics associated with a singular Riemannian metric (Carnot-Caratheodory metric).

To solve the problem, Brockett considers a class of systems which have a special canonical form. An equivalent form, which is more useful for our purposes, is

$$\begin{array}{rcll} \dot{x}_i & = & u_i & i = 1, \cdots, m \\ \dot{x}_{ij} & = & x_i u_j & i < j \end{array} \tag{6.3}$$

We see that if $m = 2$, this is exactly the contact system (6.2). It can be shown that the input vector fields and their pairwise brackets span $\mathbb{R}^n$ and hence the system is controllable with degree of nonholonomy equal to 1.

To find the optimal input between two points, we construct the Lagrangian

$$L(x, \dot{x}) = \sum_{i=1}^{m} \dot{x}_i^2 + \sum_{i,j} \lambda_{ij} (\dot{x}_{ij} - x_i u_j) \tag{6.4}$$

Here we have used the fact that $u_i = \dot{x}_i$. The λ_{ij}'s are the Lagrangian multipliers associated with the constraint imposed by the control system. Substituting equation (6.4) into the Euler-Lagrange equation

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{x}} - \frac{\partial L}{\partial x} = 0$$

it can be shown that the input must satisfy

$$u = e^{\Lambda t} u_0$$

where Λ is a constant skew-symmetric matrix. Thus the inputs are sinusoids at various frequencies. Unfortunately, even for very simple problems, determining Λ and u_0 given an initial and final configuration is very difficult.

A great deal of simplification occurs if we consider moving between configurations where $x_i(1) = x_i(0)$. In this instance the eigenvalues of Λ must be

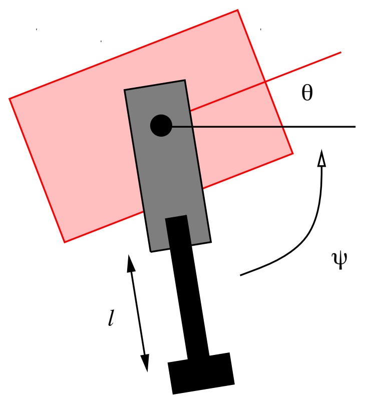


Figure 6.1: A simple hopping robot. The robot consists of a leg which can both rotate and extend. The configuration of the mechanism is given by the angle of the body and the angle and length (extension) of the leg.

multiples of 2π and Brockett showed that the optimal inputs are sinusoids at integrally related frequencies, namely $2\pi, 2 \cdot 2\pi, \cdots, \frac{m}{2} \cdot 2\pi$. This simplifies the problem tremendously and for many examples reduces the search to that of finding u_0. We use this result to propose the following algorithm for steering systems of this type:

Algorithm

1. Steer the x_i's to their desired values using any input and ignoring the evolution of the x_{ij}'s.

2. Using sinusoids at integrally related frequencies, find u_0 such that the input steers the x_{ij}'s to their desired values. By the choice of input, the x_i's are unchanged.

The resulting trajectories are suboptimal but easily computable and have several nice properties which we will explore.

Example 1

We consider as an example a kinematic hopping robot, as shown in Figure 6.1. This example has been studied by Li, Montgomery and Raibert [61] using holonomy methods. We wish to reorient the body of robot while in midair and bring the leg rotation and extension to a desired final value. The kinematic equations of the robot (in center of mass coordinates) can be written as

$$\begin{aligned} \dot\psi &= u_1 \\ \dot l &= u_2 \\ \dot\theta &= -\tfrac{m_l(l+1)^2}{1+m_l(l+1)^2} u_1 \end{aligned}$$

where we have used units such that the mass of the body is 1 and the length of the leg at zero extension is also 1. The last equation is a consequence of conservation of angular momentum. Expanding the equation using a Taylor series about $l = 0$

$$\dot\theta = -\frac{m_l}{1+m_l}\,\dot\psi - \frac{2m_l}{(1+m_l)^2}\, l u_1 + o(l) u_1$$

This suggests a change of coordinates, $\alpha = \theta + \frac{m_l}{1+m_l}\psi$ to put the equations in the form

$$\begin{aligned} \dot\psi &= u_1 \\ \dot l &= u_2 \\ \dot\alpha &= \tfrac{2m_l}{(1+m_l)^2} l u_1 + o(l) u_1 = f(l) u_1 \end{aligned}$$

This equation has the same form locally as the canonical system in equation (6.3).

Using this as justification, we attempt to use our proposed algorithm to steer the full nonlinear system. Since we control the ψ and l states directly, we first steer them to their desired values. Then using sinusoids in the ψ and l inputs,

$$\begin{aligned} u_1 &= a_1 \sin\omega t \\ u_2 &= a_2 \cos\omega t \end{aligned}$$

we steer θ to its desired value. By construction, this last motion does not affect the final values of ψ and l. To include the effect of nonlinearity in the first vector field, harmonic analysis can be used. Since l is periodic, we expand f using its Fourier series,

$$f(\frac{a_2}{\omega}\sin\omega t) = \beta_1 \sin\omega t + \beta_2 \sin 2\omega t + \cdots$$

Integrating $\dot\alpha$ over one period, only the first term in the expansion contributes to the net motion

$$\begin{aligned} \alpha(\tfrac{2\pi}{\omega}) &= \alpha(0) + \int_0^{\frac{2\pi}{\omega}} \left(\frac{a_1\beta_1}{\omega}\sin^2\omega t + \frac{a_1\beta_2}{\omega}\sin\omega t \sin 2\omega t + \cdots \right) dt \\ &= \alpha(0) + \tfrac{\pi a_1 \beta_1}{\omega^2} \end{aligned}$$

Figure 6.2 shows the trajectory for the last motion segment; ψ and l return to their initial values but α (and hence θ) experiences a net change. To compute the required input amplitudes, we plot β_1 as a function of a_2 and choose a_2 such that $\frac{a_1\beta_1}{\pi\omega} = \theta_1 - \theta_0$. Using this procedure, we can (locally) steer between any two configurations.

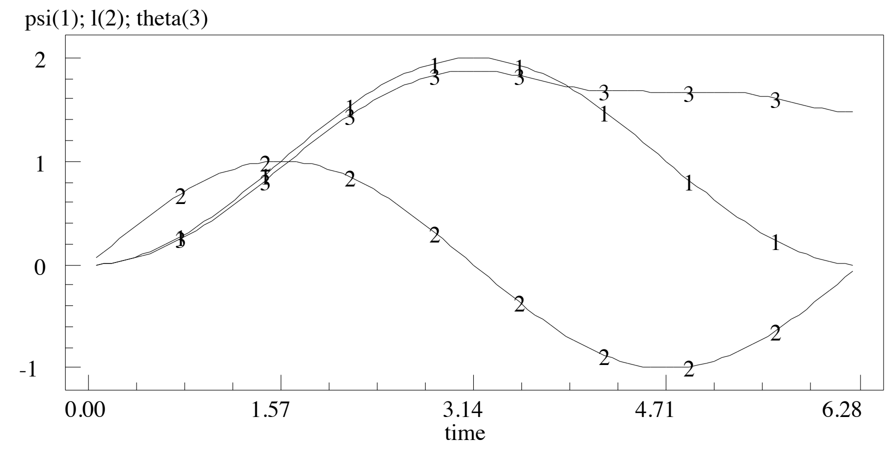


Figure 6.2: Nonholonomic motion for a hopping robot. Using sinusoidal inputs, the leg angle and extension return to their starting values but the body angle goes a net rotation.

6.2 Second degree systems

We next consider systems in which the first level of bracketing is not enough to span $\mathbb{R}^n$. We begin by trying to extend the previous canonical form to the next higher level of bracketing. Consider a system which can be expressed as

$$\begin{aligned} \dot{x}_i &= u_i && i = 1, \cdots, m \\ \dot{x}_{ij} &= x_i u_j && i < j \\ \dot{x}_{ijk} &= x_{ij} u_k && \text{(mod Jacobi identity)} \end{aligned} \tag{6.5}$$

Because Jacobi's identity imposes relations between certain brackets, not all x_{ijk} combinations are possible. This is analogous to limiting the x_{ij}'s to those for which $i < j$, reflecting skew-symmetry of the Lie bracket.[1] Using the calculation in equation (5.9) shows that this system has relative growth vector $(m, \frac{m(m-1)}{2}, \frac{(m+1)m(m-1)}{3})$. Constructing the Lagrangian (with the same integral cost function) and substituting into the Euler-Lagrange equations does not result in a constant set of Lagrange multipliers. As a consequence, we cannot solve the optimal control problem in closed form.

We can however extend and apply our previous algorithm as follows:

Algorithm

1. Steer the x_i's to their desired values. This causes drift in all other states.

[1]This is made more explicit in Section 6.3.

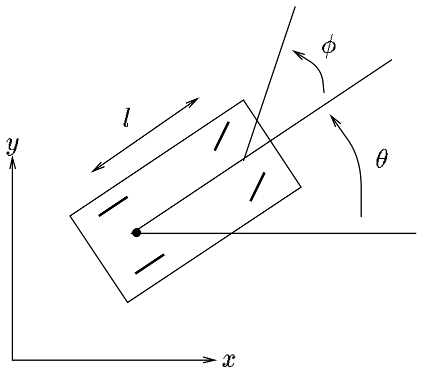


Figure 6.3: Front wheel drive cart. The configuration of the cart is determined by the Cartesian location of the back wheels, the angle the car makes with the horizontal and the steering wheel angle relative to the car body. The two inputs are the velocity of the front wheels (in the direction the wheels are pointing) and the steering velocity.

2. Steer the x_{ij}'s to their desired values using integrally related sinusoidal inputs. If the i^{th} input has frequency w_i then x_{ij} will have frequency components at $w_i \pm w_j$. By choosing inputs such that we get frequency components at zero, we can generate motion in the desired states.

3. Use sinusoidal inputs a second time to move all previously steered states in a closed loop and generate motion only in the x_{ijk} directions. This requires careful choice of the input frequencies so that $w_i \pm w_j \neq 0$ but $w_i \pm w_j \pm w_k$ has zero frequency components.

Example 2

To illustrate the algorithm, we consider the motion of a front wheel drive car as shown in Figure 6.3. The kinematics of this mechanism where derived in the last chapter and can be written as

$$\begin{aligned} \dot{x} &= \cos\theta \cos\phi \, u_1 \\ \dot{y} &= \sin\theta \cos\phi \, u_1 \\ \dot{\phi} &= u_2 \\ \dot{\theta} &= \tfrac{1}{l} \sin\phi \, u_1 \end{aligned} \tag{6.6}$$

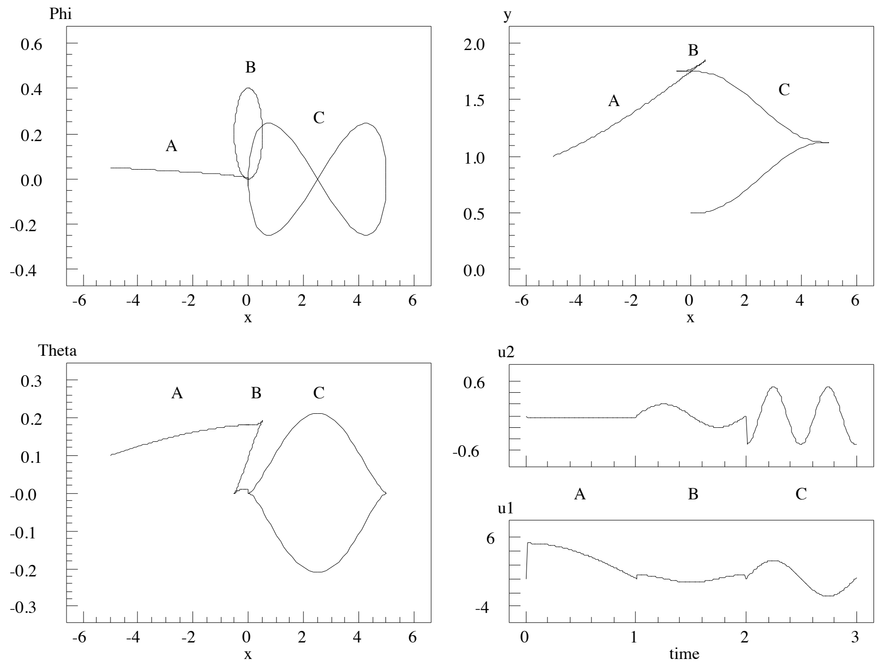


Figure 6.4: Sample trajectories for a car. The trajectory shown is a three stage path which moves the unicycle from $x = -5$, $y = 1$, $\theta = 0.05$, $\phi = 1$) to (0, 0.5, 0, 0). The first three figures show the states versus x; the bottom right figures show the inputs as functions of time.

In this form, u_1 does not control any state directly. We use a change of coordinates and a change of input to put the equations in the form

$$\begin{aligned} \dot{x} &= v_1 & v_1 &= \cos\theta\cos\phi u_1 \\ \dot{\phi} &= v_2 & v_2 &= u_2 \\ \dot{\alpha} &= \tan\phi\, v_1 & \alpha &= \sin\theta \\ \dot{y} &= \tfrac{\alpha}{\sqrt{1-\alpha^2}}\, v_1 \end{aligned}$$

As before, the linear portion of the nonlinearities matches the canonical system and we can include the effects of the nonlinearities using Fourier series techniques.

An example of the algorithm applied to the car is given in Figure 6.4. The first portion of the path, labeled A, drives the x and ϕ states to their desired values using a constant input. The second portion, labeled B, uses a periodic input to drive θ while bringing the other two states back to their desired values. The last step brings y to its desired value and returns the other three states to their correct values. The Lissajous figures that are obtained from the phase portraits of the different variables are quite instructive. Consider the portion of the curve labeled C. The upper left plot contains the Lissajous figure for x, ϕ (two loops); the lower left plot is the corresponding figure for x, θ (one loop) and the open curve in x,y shows the increment in the y variable. The very powerful implication here is that the Lie bracket directions correspond to rectification of harmonic periodic motions of the driving vector fields and the harmonic relations are determined by the degree of the Lie bracket corresponding to the desired direction of motion. This point has also been made rather elegantly by Brockett [13] in the context of the rectification of mechanical motion.

6.3 Higher order canonical systems

We now study more general examples of nonholonomic systems and investigate the use of sinusoids for steering such systems. As before, we begin by studying a class of canonical examples and then attempt to extend the analysis to non-canonical systems.

To construct systems with a given number of inputs and degree of nonholonomy, it is necessary to introduce some additional machinery. As mentioned earlier, in constructing canonical systems we must observe the fundamental restrictions imposed by the Lie bracket: skew-symmetry and the Jacobi identity. Our search for a set of vector fields which have a given degree of nonholonomy is equivalent to searching for a basis for an abstract, finite-dimensional Lie algebra. One such basis is P. Hall basis [35, 85].

P. Hall Bases

Given a set of generators $\{X_1, \cdots, X_m\}$, we define the length of a Lie product recursively as

$$\begin{array}{rcll} l(X_i) & = & 1 & \qquad i = 1, \cdots, m \\ l([A, B]) & = & l(A) + l(B) & \end{array}$$

where A and B are themselves Lie products. A Lie algebra is *nilpotent* if there exists an integer k such that all Lie products of length greater than k are zero. k is called the degree of nilpotency. A nilpotent Lie algebra is finite-dimensional. A *P. Hall basis* is an ordered set of Lie products $H = \{B_i\}$ satisfying

(PH1) $X_i \in H$, $i = 1, \cdots, m$

(PH2) If $l(B_i) < l(B_j)$ then $B_i < B_j$

(PH3) $[B_i, B_j] \in H$ if and only if

- **(a)** $B_i, B_j \in H$ and $B_i < B_j$ *and*
- **(b)** either $B_j = X_k$ for some k or
  $B_j = [B_l, B_r]$ with $B_l, B_r \in H$ and $B_l \leq B_i$

The proof that a P. Hall basis is a basis for the free Lie algebra generated by $\{X_1, \cdots, X_m\}$ can be found in [35, 85].

A P. Hall basis with degree of nilpotency k can be constructed from a set of generators using the definition. The simplest approach is to construct all possible Lie products with length less than k and use the definition to eliminate elements which fail to satisfy one of the properties. In practice, the basis can be built in such a way that only (PH3) need be checked. A Mathematica program which performs this calculation is given in the appendix.

Examples

A basis for the nilpotent Lie algebra of degree 3 generated by $\{X, Y, Z\}$ is

$$\begin{array}{llll} X & Y & Z & \\ [X,Y] & [X,Z] & [Y,Z] & \\ [X,[X,Y]] & [X,[X,Z]] & [Y,[X,Y]] & [Y,[X,Z]] \\ [Y,[Y,Z]] & [Z,[X,Y]] & [Z,[X,Z]] & [Z,[Y,Z]] \end{array}$$

Note that $[X,[Y,Z]]$ does not appear since

$$[X,[Y,Z]] + [Y,[Z,X]] + [Z,[X,Y]] = 0$$

and two of the three terms are already present.

A larger example, which we will use in the sequel, is a basis for a Lie algebra of degree 5 with 2 generators:

$$\begin{array}{rl} B_1 - B_2: & X \quad Y \\ B_3: & [X,Y] \\ B_4 - B_5: & [X,[X,Y]] \quad [Y,[X,Y]] \\ B_6 - B_8: & [X,[X,[X,Y]]] \quad [Y,[X,[X,Y]]] \quad [Y,[Y,[X,Y]]] \\ B_9 - B_{14}: & [X,[X,[X,[X,Y]]]] \quad [Y,[X,[X,[X,Y]]]] \quad [Y,[Y,[X,[X,Y]]]] \\ & [Y,[Y,[Y,[X,Y]]]] \quad [[X,Y],[X,[X,Y]]] \quad [[X,Y],[Y,[X,Y]]] \end{array}$$

Note that B_{13} and B_{14} have the form $[B_3, B_4]$ and $[B_3, B_5]$, requiring careful checking of the condition (PH3).

Maximum growth canonical systems

Using a P. Hall basis, it is possible to construct vector fields which have maximum growth; at each level of bracketing the dimension of the filtration grows by the maximum possible amount. More specifically, we wish construct a set of vector fields $\{X_i\}$ such that when the vector fields are substituted into the expressions for the P. Hall basis elements, the resulting set of vector fields is linearly independent. The method of construction used here is due to Grayson and Grossmann [34]. We present only the 2-input case since this is of the most interest to us.

An important property of a P. Hall Basis is that each basis element has a unique representation as a set of nested Lie products

$$B_i = [B_{i_1}, [B_{i_2}, \cdots [B_{i_l}, X_j] \cdots]] \tag{6.7}$$

Given a P. Hall basis element $B = [B_i, B_j]$, we convert it into this form by recursively expanding B_j. We associate with each such basis element a vector $\alpha_i \in \mathbb{Z}^n$ which indicates the number of times each basis element occurs in the expansion (6.7). Thus $\alpha_i(k)$ is the number of times B_k appears in the expansion for B_i. From the properties of a P. Hall basis, it is clear that $\alpha_i(k) = 0$ if $k \geq i$.

Given a P. Hall basis $H = \{B_1, \cdots, B_n\}$ we construct a vector field on $\mathbb{R}^n$ using coordinates $h \in \mathbb{R}^n$. Assume $B_i = X_i$ for $i = 1, \cdots, m$. Given α_i associated with B_i, $i > m$, we define

$$\begin{aligned} h^{\alpha_i} &= \prod_j h_j^{\alpha_i(j)} \\ \alpha_i! &= \prod_j \alpha_i(j)! \end{aligned}$$

Theorem 6.1 (Grayson and Grossman) *Fix $k \geq 1$ and let n be the rank of the free, nilpotent Lie algebra of order k with 2 generators. Then*

$$X_1 = \frac{\partial}{\partial h_1} \qquad X_2 = \frac{\partial}{\partial h_2} + \sum_{i=3}^{n} \frac{h^{\alpha_i}}{\alpha_i!} \frac{\partial}{\partial h_i}$$

generate a free, nilpotent Lie algebra (of vector fields) of order k at the origin.

The vector fields generated by this theorem are extensions of the canonical forms we have seen for degree of nonholonomy 1 and 2. The degree of nonholonomy for these vector fields is identical to the order of nilpotency. One way to interpret and gain insight into this formula is to note that a Lie product

$$[B_{i_1}, [B_{i_2}, \cdots [B_{i_k}, X_2]]]$$

corresponds to a vector field obtained by taking the derivative of the components of X_2 with respect to $h_{i_1}, h_{i_2}, \cdots, h_{i_k}$. The coefficients of X_2 are chosen such that taking this derivative leaves 1 in the $\frac{\partial}{\partial h_i}$ term.

Example

Consider the two input example given previously, but with order of nilpotency 4 instead of 5. The system generated by Theorem 6.1 is

$$\begin{aligned}
\dot{h}_1 &= u_1 && X \\
\dot{h}_2 &= u_2 && Y \\
\dot{h}_3 &= h_1 u_2 && [X, Y] \\
\dot{h}_4 &= \tfrac{1}{2} h_1^2 u_2 && [X, [X, Y]] \\
\dot{h}_5 &= h_1 h_2 u_2 && [Y, [X, Y]] \\
\dot{h}_6 &= \tfrac{1}{6} h_1^3 u_2 && [X, [X, [X, Y]]] \\
\dot{h}_7 &= \tfrac{1}{2} h_1^2 h_2 u_2 && [Y, [X, [X, Y]]] \\
\dot{h}_8 &= \tfrac{1}{2} h_1 h_2^2 u_2 && [Y, [Y, [X, Y]]]
\end{aligned}$$

We can now ask ourselves if it is possible to steer these canonical systems using simple sinusoids. Although the form of the system is different from that we used in Section 6.2, the same approach can be used to steer h_1 through h_5.

That is, sinusoids at the same frequency and proper phase give motion in h_3 and sinusoids at frequency 1 and 2 give motion in h_4 and h_5 (switching the input frequency switches between h_4 and h_5). This can be verified by direct calculation.

Steering in the $h_6 - h_8$ directions is more difficult. Consider the effect of using two simple sinusoids as inputs, $u_1 = a\cos\omega_1 t$ and $u_2 = b\sin\omega_2 t$. In order to prevent motion in lower level brackets, we must have $\omega_1 \neq \pm\omega_2$, $\omega_1 \neq \pm 2\omega_2$, $\omega_2 \neq \pm 2\omega_1$. Assuming these relationships hold, we get the following frequency components in the derivatives of the dynamic system:

$$\begin{array}{llll} h_6: & \omega_1 \pm \omega_2 & 3\omega_1 \pm \omega_2 & \\ h_7: & \omega_1 \quad 2\omega_1 & 2\omega_2 & 2\omega_1 \pm 2\omega_2 \\ h_8: & \omega_2 \quad \omega_1 \pm \omega_2 & \omega_1 \pm 3\omega_2 & \end{array}$$

By choosing frequencies such that the derivative has a term at frequency 0, we get motion in that coordinate. Thus $\omega_2 = 3\omega_1$ gives motion in h_6 (only) and $\omega_1 = 3\omega_2$ gives motion in h_8 (only).

Based on these calculations, it would appear that choosing $2\omega_1 = 2\omega_2$ would give motion in h_7. This is in fact the case, but we *also* get motion in the h_3 direction. It is not possible to get motion *only* in the h_7 direction using simple sinusoids. A direct calculation verifies that adjusting the phasing of the inputs does not resolve this dilemma. It may still be possible to steer the system using combinations of sinusoids at different frequencies for each input or using more complicated periodic functions (such as elliptic functions).

Rather than explore the use of more complicated inputs for steering nonholonomic systems, we consider instead a simpler class of systems. The justification for changing the class of systems is simple—most of the systems encountered as examples do not have the complicated structure of our canonical example. Thus there may be a simpler class of systems which is both steerable using simple sinusoids and representative of systems in which we are interested.

Chained canonical systems

Consider a two input system of the following form:

$$\begin{array}{rclcrcl} \dot{x}_0 & = & u_x & \qquad & \dot{y}_0 & = & u_y \\ \dot{x}_1 & = & y_0 u_x & & (y_1 & := & x_1) \\ \dot{x}_2 & = & x_1 u_x & & \dot{y}_2 & = & y_1 u_y \\ \dot{x}_3 & = & x_2 u_x & & \dot{y}_3 & = & y_2 u_y \\ & \vdots & & & & \vdots & \\ \dot{x}_{n_x} & = & x_{n_x-1} u_x & & \dot{y}_{n_y} & = & y_{n_y-1} u_y \end{array} \tag{6.8}$$

or more compactly

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = Xu_x + Yu_y \qquad \begin{array}{rcl} X & = & \frac{\partial}{\partial x_0} + y_0\frac{\partial}{\partial x_1} + \sum_{i=2}^{n} x_{i-1}\frac{\partial}{\partial x_i} \\ Y & = & \frac{\partial}{\partial y_0} + \sum_{j=2}^{n} y_{j-1}\frac{\partial}{\partial y_j} \end{array}$$

where $y_1 := x_1$ to account for skew-symmetry of the Lie bracket. This system has similar form to that considered in section 6.2, but we have extended the degree of nilpotency of the system. It is convenient to denote iterated Lie products as $\mathrm{ad}_X^k Y$:

$$\mathrm{ad}_X Y = [X,Y] \qquad \mathrm{ad}_X^k Y = [X, \mathrm{ad}_X^{k-1}Y] = [X,[X,\cdots,[X,Y]\cdots]]$$

Lemma 6.2 *For the vector fields in equation (6.8)*

$$\begin{array}{rcll} ad_X^k Y & = & (-1)^k\frac{\partial}{\partial x_k} & \\ ad_Y^k X & = & (-1)^{k+1}\frac{\partial}{\partial y_k} & \quad k > 1 \end{array}$$

Proof. By induction. Since the first level of brackets is irregular, we begin by expanding $[X,Y]$ and $[X,[X,Y]]$.

$$\begin{array}{rcl} [X,Y] & = & \left(\frac{\partial}{\partial x_0} + y_0\frac{\partial}{\partial x_1} + \sum x_{i-1}\frac{\partial}{\partial x_i}\right)\left(\frac{\partial}{\partial y_0} + \sum y_{j-1}\frac{\partial}{\partial y_j}\right) - \\ & & \left(\frac{\partial}{\partial y_0} + \sum y_{j-1}\frac{\partial}{\partial y_j}\right)\left(\frac{\partial}{\partial x_0} + y_0\frac{\partial}{\partial x_1} + \sum x_{i-1}\frac{\partial}{\partial x_i}\right) \\ & = & 0 - \frac{\partial}{\partial x_1} \\ [X,[X,Y]] & = & X(-\frac{\partial}{\partial x_1}) + \frac{\partial}{\partial x_1}(X) = 0 + \frac{\partial}{\partial x_2} \end{array}$$

Now assume that $\mathrm{ad}_X^k Y = (-1)^k\frac{\partial}{\partial x_k}$. Then

$$\mathrm{ad}_X^{k+1}Y = [X, \mathrm{ad}_X^k Y] = (-1)^k\left(X(\frac{\partial}{\partial x_k}) - \frac{\partial}{\partial x_k}(X)\right) = (-1)^{k+1}\frac{\partial}{\partial x_{k+1}}$$

The proof for $\mathrm{ad}_Y^k X$ is identical using the facts $[Y,X] = -[X,Y]$ and $y_1 := x_1$. □

Theorem 6.3 *System (6.8) is maximally nonholonomic (controllable).*

Proof. There are $2n-1$ coordinates in (6.8) and the $2n-1$ Lie products

$$\{X, Y, \mathrm{ad}_X^i Y, \mathrm{ad}_Y^j X\} \qquad i > 1, \quad j > 2$$

are independent using Lemma 6.2. We require $j > 2$ since $\mathrm{ad}_Y X = -\mathrm{ad}_X Y$ and hence those Lie products can never be independent. □

To steer this system, we use sinusoids at integrally related frequencies. Roughly speaking, if we use $u_x = \sin t$ and $u_y = \cos kt$ then $\dot{x}_1$ will have components at frequency $k-1$, $\dot{x}_2$ at frequency $k-2$, etc. $\dot{x}_k$ will have a component at frequency zero and when integrated we get motion in x_k while all previous variables return to their starting values. In the y variables, all frequency components will be of the form $m \cdot k \pm 1$ and hence we get no motion for $k > 1$. (For $k = 1$, y_1 and x_1 are the same variable). We make this precise with the following algorithm.

Algorithm

1. Steer x_0 and y_0 to their desired values.

2. For each x_k, $k \geq 1$, steer x_k to its final value using

$$u_x = a \sin t \qquad u_y = b \cos kt$$

where a and b satisfy

$$x_k(2\pi) - x_k(0) = \frac{(a/2)^k b}{k!} \cdot 2\pi$$

3. For each y_k, $k \geq 2$, steer y_k to its final value using

$$u_x = b \cos kt \qquad u_y = a \sin t$$

where a and b satisfy

$$y_k(2\pi) - y_k(0) = \frac{(a/2)^k b}{k!} \cdot 2\pi$$

Theorem 6.4 *Algorithm 6.3 can steer (6.8) to an arbitrary configuration.*

Proof. The proof is constructive. It suffices to consider only step 2 since step 3 can be proved by switching x and y in what follows. We must show 2 things:

1. moving x_k does not affect x_j, $j < k$

2. moving x_k does not affect y_j, $j = 1, \cdots, n_y$

To verify that using $u_1 = a \sin t$, $u_2 = b \cos t$ produces motion only in x_k, we integrate the x states. If x_{k-1} has terms at frequency ω_i, then x_k has corresponding terms at $\omega_i \pm 1$ (by expanding products of sinusoids as sums of sinusoids). Since the only way to have $x_i(2\pi) \neq x_i(0)$ is to have x_i have a component at frequency zero, it suffices to keep track only of the lowest frequency component in each variable; higher components will integrate to zero. Direct computation starting from the origin yields

$$\begin{aligned}
x_0 &= a(1-\cos t)\\
x_1 &= \int \frac{1}{2}\frac{ab}{k}\sin kt \sin t = \frac{ab}{k(k-1)}\sin(k-1)t + \frac{1}{2}\frac{ab}{k(k+1)}\sin(k+1)t\\
x_2 &= \tfrac{1}{2^k}\tfrac{a^2b}{k(k-1)(k-2)}\sin(k-2)t + \cdots\\
&\vdots\\
x_k &= \int \left(\frac{a^k b}{2^{k-1}k!}\sin^2 t + \cdots\right) dt = \frac{a^k b}{2^{k-1}k!}\frac{t}{2} + \cdots
\end{aligned}$$

$x_k(2\pi) = x_k(0) + \frac{(a/2)^k b}{k!}\pi$ and all earlier x_i's are periodic and hence $x_i(2\pi) = x_i(0)$, $i < k$. If the system does not start at the origin, the initial conditions generate extra terms of the form $x_{i-1}(0)u_2$ in the i^{th} derivative and this integrates to zero, giving no net contribution.

To show that we get no motion in the y variables, we show that all frequency components in the y's have the form $mk \pm 1$ where m is some integer. This is true for $y_1 := x_1$ from the calculation above. Assume it is true for y_i

$$\begin{aligned}
\dot{y}_{i+1} &= y_i u_2\\
&= \sum_m \alpha(m)\sin(mk \pm 1)t \cdot \cos kt\\
&= \sum_m \frac{\alpha(m)}{2}(\sin((m+1)k \pm 1)t + \sin((m-1)k \pm 1)t)
\end{aligned}$$

Hence y_{i+1} has components at frequency $m'k \pm 1$ and therefore $y_i(2\pi) = y_1(0)$. □

To include systems with more than two inputs, we replicate the structure of (6.8) for each additional input. Let h_{ij}^k represent the motion corresponding to the Lie product $\mathrm{ad}_{X_i}^k X_j$. In the two input case, $x_k = h_{21}^k$ and $y_k = h_{12}^k$. Consider the following system on $\mathbb{R}^n$:

$$\begin{aligned}
\dot{h}_j^0 &= u_j & j &= 1, \cdots, m\\
\dot{h}_{ij}^1 &= h_i^0 u_j & i &> j \text{ and } h_{ji}^1 := h_{ij}^1\\
\dot{h}_{ij}^k &= h_{ij}^{k-1} u_j
\end{aligned} \tag{6.9}$$

Theorem 6.5 *The system (6.9) is maximally nonholonomic and can be steered using sinusoids.*

Proof. The system (6.9) can be rewritten

$$\dot{h} = X_1 u_1 + \cdots + X_m u_m$$

with

$$X_j = \frac{\partial}{\partial h_j^0} + \sum_{\substack{i=1\\ i<j}}^{m} h_i^0 \frac{\partial}{\partial h_{ij}^1} + \sum_k \sum_i h_{ij}^{k-1}\frac{\partial}{\partial h_{ij}^k}$$

Given any two X_i, X_j, their Lie product expansions only involve terms of the form h_{ij}^k for some k. But this is precisely the vector fields from Lemma 6.2 and hence

$$\mathrm{ad}_{X_i}^k X_j = (-1)^{k+1} \frac{\partial}{\partial h_{ij}^k}$$

Taking these terms for all possible i, j, k we get a set of independent Lie products just as in the proof of Theorem 6.3.

To show that the system can be steered using sinusoids, pick any $i, j \in \{1, \cdots, m\}$, $i < j$. Fix $u_l = 0$ for all $l \neq i, j$. The resulting system is identical to (6.8) can be steered using algorithm 6.3. By choosing all possible combinations of i and j, we can move to any position. □

We define systems having the form of equation (6.9) as *chained canonical systems*. More generally, we shall say a system is a *chained system* if the Lie algebra generated by the input vector fields is spanned by the vector fields of the form $\mathrm{ad}_{g_i}^k g_j$. In the canonical case, for a fixed i, j, we refer to the vector fields $\mathrm{ad}_{g_i}^k g_j$ and their corresponding coordinates h_{ij}^k as a *chain*. Each chain in the canonical system can be steered using sinusoidal inputs in the appropriate input channels.

Non-canonical chained systems

We would like to extend the class of systems which we can steer by including systems which have similar structure to equation (6.8), but with additional nonlinearities. The following example illustrates the limitations of using sinusoidal inputs for this purpose. Consider the system

$$\begin{aligned} \dot{x}_0 &= u_1 \\ \dot{y}_0 &= u_2 \\ \dot{x}_1 &= (y_0 + \epsilon y_0^2) u_1 \\ \dot{x}_2 &= (x_1 + \epsilon x_1^2) u_1 \\ \dot{x}_3 &= x_2 u_1 \end{aligned}$$

This satisfies our definition of a chained system with a single chain: $\mathrm{ad}_{g_1}^k g_2$, $k = 0, 1, 2, 3$ together with g_2 forms a basis for $\mathbb{R}^5$.

If we apply inputs $u_1 = \sin t$ and $u_2 = \cos 3t$, we get the following motion, starting from $x = 0$

$$\begin{aligned} x_0(2\pi) &= y_0(2\pi) = x_1(2\pi) = 0 \\ x_2(2\pi) &= -\tfrac{7}{1440}\epsilon^2 \\ x_3(2\pi) &= \tfrac{\pi}{24} + 2.5 \times 10^5 \epsilon^2 \end{aligned}$$

The reason for this perturbation in x_2 is that the (small) nonlinear terms cause zero frequency components to appear in $\dot{x}_2$. Hence we cannot use simple sinusoids to steer this system as before.

Nonetheless, there are many special instances where sinusoids are an important tool. For example, we were able to steer the automobile with sinusoids, despite the nonlinearities. Since the automobile had degree of nonholonomy 2, the problems present in the previous example do not occur. Another example is a system which has the chained canonical form until the last coordinate. In this case, harmonic analysis is needed when finding the motion at the last step of the algorithm and zero frequency terms do not appear in any previous coordinates.

It may also be possible to use feedback transformation to convert certain systems into chained canonical form. This is similar to the technique used in nonlinear control to convert a nonlinear system into a linear one by using a change of coordinates and state feedback. Similar efforts have been used by Lafferriere and Sussmann [54] to convert systems into nilpotent form for use with their planning algorithm. It is interesting to note that in several of their examples, the converted systems where also in chained canonical form.

Finally, sinusoids may be useful for steering systems which are not locally in canonical form. The minimum structure necessary to attempt motion generation using sinusoids is a *triangular system.* A system is triangular if we can find a set of coordinates $h = (h^1, h^2, \cdots, h^p) \in \mathbb{R}^{m_1 \times m_2 \times \cdots \times m_p} = \mathbb{R}^n$ such that

$$\begin{aligned} \dot{h}^1 &= v & v \in \mathbb{R}^{m_1} \\ \dot{h}^2 &= f^2(h^1)v \\ \dot{h}^3 &= f^3(h^1, h^2)v \\ &\vdots \\ \dot{h}^p &= f^p(h^1, \cdots, h^{p-1})v \end{aligned}$$

The triangular form was necessary in our examples to insure that the differential equations driven by sinusoidal inputs could be integrated in a stepwise fashion. We now investigate a specific example in which this triangular form allows us to achieve interesting motions using sinusoids.

6.4 Dynamic finger repositioning, revisited

We now reconsider the case of a multifingered hand grasping an object. The equations of motion are given by equation (5.4), which we reproduce here:

$$\begin{aligned} \dot{x}_o &= u_1 \\ \dot{\eta} &= B_1(x_o, \eta)u_1 + B_2(x_o, \eta)u_2 \end{aligned} \tag{6.10}$$

Recall that these equations were obtained by attaching a controller to the system and letting u_1 reflect the desired object velocity and u_2 parameterize the internal motion of the system. We consider the special case when the object position is held fixed and hence $u_1 = 0$.

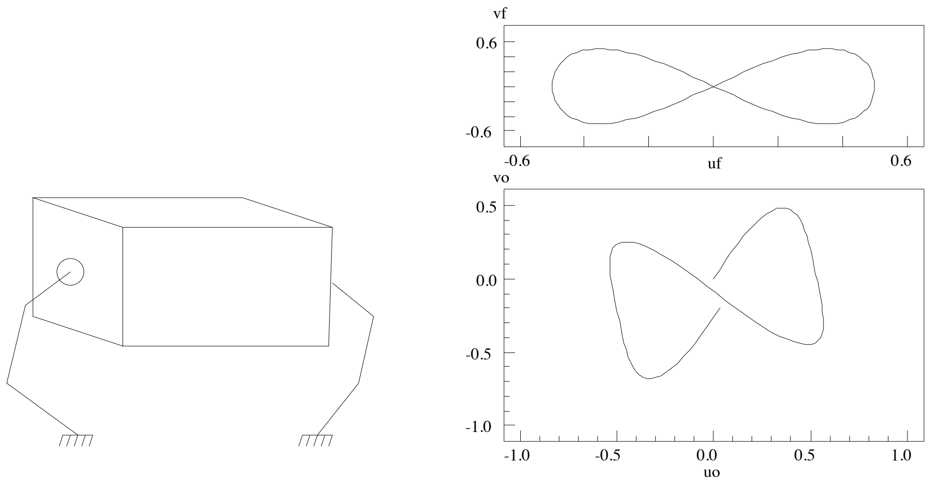


Figure 6.5: Steering applied to a multifingered hand. We consider the motion of a finger with a spherical tip on a polyhedral object (left). The plots to the right show a trajectories which move a finger down the side of an object. The location of the contact on the finger is unchanged (upper graph), while the location of the contact on the face of the object undergoes a net y displacement (lower graph).

Consider the case of a single spherical finger rolling on a plane. The control kinematics were derived in Section 2.3:

$$\dot{\eta} = \begin{pmatrix} \dot{u}_f \\ \dot{v}_f \\ \dot{u}_o \\ \dot{v}_o \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \rho\cos\psi \\ -\rho\sin\psi \\ 0 \end{pmatrix} \omega_1 + \begin{pmatrix} 0 \\ \sec u_f \\ -\rho\sin\psi \\ -\rho\cos\psi \\ -\tan u_f \end{pmatrix} \omega_2 \tag{6.11}$$

For simplicity, we assume that we control ω_1 and ω_2 directly. It can be verified that the system has relative growth vector $(2, 1, 2)$ and that by a change of input variables we can put this system into strictly triangular form. Furthermore, the approximate version of this system is given by

$$\begin{array}{rrcl} u_f\colon & \dot{x}_1 & = & u_1 \\ v_f\colon & \dot{x}_2 & = & u_2 \\ -\psi\colon & \dot{x}_3 & = & x_1 u_2 \\ u_o - \rho u_f\colon & \dot{x}_4 & = & x_3 u_2 \\ v_o + \rho v_f\colon & \dot{x}_5 & = & x_3 u_1 \end{array} \tag{6.12}$$

This is identical to the canonical form in equation (6.5). Using the same techniques as before, we can construct paths using integrally related sinusoids and apply these sinusoids to the full nonlinear system in equation (6.11). An example of a path which moves a finger vertically down the side of a planar object is shown in Figure 6.5.

A more challenging example considered by Li and Canny is that of moving a spherical finger on a spherical object. It may be verified that the system is controllable except when the object and finger radii are identical (in this case the rolling constraint becomes holonomic). The contact kinematics, from equation (2.16) are

$$\dot{\eta} = \begin{pmatrix} \frac{1}{1+\rho} \\ 0 \\ \frac{\rho}{1+\rho}\cos\psi \\ -\frac{\rho}{1+\rho}\sin\psi\sec u_o \\ \frac{\rho}{1+\rho}\sin\psi\tan u_o \end{pmatrix} \omega_1 + \begin{pmatrix} 0 \\ \frac{1}{1+\rho}\sec u_f \\ -\frac{\rho}{1+\rho}\sin\psi \\ -\frac{\rho}{1+\rho}\cos\psi\sec u_o \\ \frac{\rho}{1+\rho}(\cos\psi\tan u_o - 1/\rho\ \tan u_f) \end{pmatrix} \omega_2 \quad (6.13)$$

We see that this system is not strictly triangular (ψ depends on u_o) and hence requires a more sophisticated approach. Motion for this particular system can be constructed using the techniques described by Li and Canny due to the special choice of object and finger shapes. Motion planning for more general choices of finger and object shapes is still unsolved.

6.5 Discussion

There are many questions in the use of sinusoids for generating trajectories for nonholonomic questions. The full class of systems to which the techniques presented here can be applied is not known. In particular, the use of feedback transformations and the necessary conditions for converting a system into chained canonical form is of great interest. This appears to be related to the problem of nilpotentization of a set of vector files, which has been by Hermes, Lundell, and Sullivan [40].

Another fundamental problem of great practical importance is obstacle avoidance. In many situations, such as parallel parking a car, obstacles play an important role in choosing a good trajectory. The use of sinusoidal inputs allows some freedom in shaping a trajectory based on the presence of obstacles. By varying the amplitude, phase, and number of cycles of the input sinusoids, different trajectories can be generated which result in the same net motion. Figure 6.6 shows three possible trajectories for parallel parking a car.

Finally, we note that all of the trajectories we have studied have been fundamentally open loop. Errors in the initial conditions, model mismatch, and sensor noise will all degrade performance. A fundamental property of feedback control is robustness with respect to these disturbances. We are currently investigating methods for generating controllers for nonholonomic systems which respect the fundamental limitations mentioned in the introduction. An effective

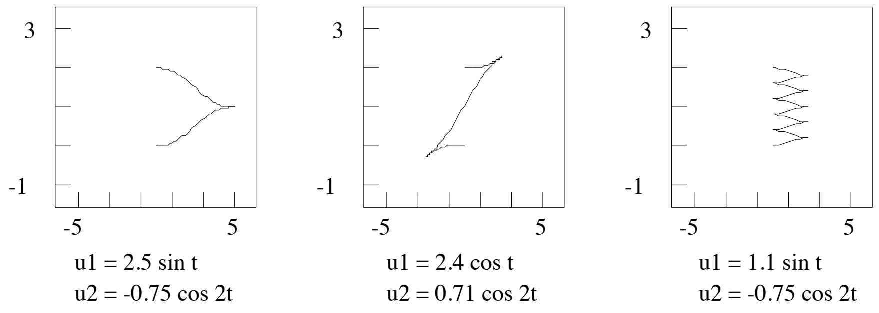


Figure 6.6: Alternative trajectories for parallel parking

strategy may be to design controllers which perform some type of trajectory tracking rather than stabilization to a point. This point is discussed more fully in the conclusion.

Nonholonomic path planning represents a fusion of some of the newest ideas in control theory, classical mechanics, and differential geometry with some of the most challenging practical problems in robot motion planning. Furthermore, the class of systems to which the theory is relevant is broad: mobile robots, space-based robots, multifingered hands, and even such systems as a one-legged hopping robot. The techniques presented here indicate one possible method for generating efficient and computable trajectories for some of these nonholonomic systems in the absence of obstacles.

Appendix I – PhilipHall.m

The following file is a Mathematica program to calculate a Philip Hall basis of a given degree with a set of generators.

```
(*
 * PhilipHall.m - generate a Philip Hall basis
 *
 * Richard M. Murray
 * July 12, 1990
 *
 * Example: PhilipHallBasis[{X,Y,Z}, 4]
 *)

(* Define laws for expanding Lie brackets *)
Bracket[X_List, Y_List] := Flatten[Map[ Bracket[X, #] &, Y]];
Bracket[X_List, Y_] := Map[ Bracket[#, Y] &, X];
Format[Bracket[X_, Y_]] :=
  StringJoin["[", ToString[X], ", ", ToString[Y], "]"];


(* Define the degree of a Lie bracket expression *)
degree[Bracket[A_, B_]] := degree[A] + degree[B];
degree[X_Symbol] := 1;

(* Define the ordering on the Lie algebra; use default math ordering *)
leq[A_, B_] := degree[A] < degree[B] || OrderedQ[{A, B}];
lt[A_, B_] := leq[A, B] && Not[SameQ[A, B]];

(*
 * Philip Hall criteria
 *
 * These are from Serre.  Basically only PH3 needs to be checked since
 * all others are automatically satisfied by our construction.
 *
 *)

PHallCheck[Bracket[A_, Bracket[C_, D_]]] := lt[A, Bracket[C, D]] && leq[C, A]
PHallCheck[Bracket[A_, B_]] := lt[A, B]
PHallCheck[X_] := (degree[X] == 1);


(* Return a list of elements which satisfy P. Hall criteria *)
PHallFilter[list_List] :=
    Flatten[Map[ If[PHallCheck[#], #, {}]&, list]]

(* Generate a Philip Hall basis given a list of generators and the order *)
PhilipHallBasis[X_List, k_] :=
  Block[
    {basis = Sort[X], list},

    (* Go through and generate candidate elements up to the desired order *)
    For[order = 2, order <= k, ++order,
      (* Take the bracket of our basis so far *)
      (*! This could be done more intelligently !*)
    list = Bracket[basis, basis];

    (* Now go through and get rid of things that arent the desired order *)
    list = Flatten[Map[If[degree[#] == order, #, {}]&, list]];

    (* Now go through and find elements which satisfy PH criteria *)
    basis = Union[Join[basis, PHallFilter[ Union[list] ]]] ];
  basis
];
```

Chapter 7

Conclusion

In this dissertation we have studied two aspects of the control of constrained mechanical systems. The first is closed loop control of robotic systems and the second is motion planning for nonholonomic robotic systems.

The dynamics of a constrained robot system can be formulated in a unified fashion. That is, we may write the dynamics of a large class of constrained robot systems as

$$\begin{aligned} &M(q)\ddot{x} + C(q,\dot{q})\dot{q} + N(q,\dot{q}) = F \\ &J(q)\dot{\theta} = G^T(q)\dot{x} \end{aligned}$$

where $q = (\theta, x)$, and θ and x represent the configurations of the manipulators and the configuration of the object being manipulated. The important properties of this system are that $M(q)$ is uniformly positive definite and $\dot{M} - 2C$ is skew-symmetric.

Using this class of systems, we can formulate control laws which are provably stable. In particular, we have shown stability for a very general control law which controls position and stiffness in complementary directions. This control law has the form

$$F = M(q)\begin{pmatrix} \ddot{x}_{1d} + \lambda_1 \dot{e}_1 \\ \ddot{x}_{2d} \end{pmatrix} + C(q,\dot{q})\begin{pmatrix} \dot{x}_{1d} + \lambda_1 \dot{e}_1 \\ \dot{x}_{2d} \end{pmatrix} + \begin{pmatrix} M_{11}\lambda_2(\dot{e}_1 + \lambda_1 e_1) \\ K_v \dot{e}_2 + K_p e_2 \end{pmatrix} + N(q,\dot{q})$$

where x_1 represents the position directions and x_2 represents the stiffness directions. In the x_1 coordinates, this control law is a modified version of computed torque, while in the stiffness directions this is a modified version of PD control. Thus we have a unified control law which is provably stable over a large class of systems.

In addition to formulating the hybrid position/stiffness controller, the form of the dynamics of a system with velocity constraints is sufficiently structured to allow us to build complicated controllers using three simple primitives—**DEFINE**, **ATTACH**, and **CONTROL**. These primitives allow us to construct controllers which share properties of biological motor control systems, that is, to distribute control at different levels.

Given controllers for constrained systems, we can formulate the planning problem. In the simplest case, we simply control the "output" of the system, parameterized by x. In addition to x, a consistent task specification must also include specification of internal forces and internal motions.

The nonholonomic motion planning problem is introduced when we wish to move between configurations of the *entire* system, rather than just between values of the output function. An example of such a task is dynamic finger repositioning, where we wish to move the location of the fingers on the object. If the constraints are nonholonomic, the fingers can be moved without releasing the object or allowing it to slip.

The techniques used for steering nonholonomic systems apply not only to systems with contact constraints, but any system with linear velocity constraints. The methods developed here, using integrally related sinusoidal inputs, can be used to control a number of specific systems, including systems which obey conservation of angular momentum as well a systems with rolling contact constraints.

Future work

There are many problems, both general and specific, which remain open. The techniques and terminology introduced in this dissertation provide a starting point for many of these areas of research. We discuss a few of these areas in more detail below.

Robotic control

One of the primary challenges in robotic control is the integration of large numbers of manipulators performing a coordinated task. The primitives given in Chapter 4 allow specification of controllers which can be used for such tasks. A typical task consists of many different phases, each potentially requiring a different type of control. An example of this is a hand grabbing an object, in which the initial move and grab phases require position control of the fingers, while the manipulation phase requires control of the composite grasping system.

For each phase of the task, the control primitives can be used to specify hierarchical control structures appropriate to the motion being performed. Methods for smoothly changing from one control structure to another, for example when contact is being made with an object, need to be developed. If we represent the system dynamics as a finite state machine, with each state corresponding to a different set of constraints on the overall system, we can associate with each state a controller defined in terms of the control primitives. In this context, we are interested in what control structure should be used during the transition from one state to another. It particular, it seems important to insure that the controller does not force the system to return to the previous state and begin an oscillation between states and the associated controllers. The idea of representing the system as a graph of dynamical systems is from a suggestion by Brockett [15] and is worthy of more study.

At the level of a single control structure, it would be very useful to determine broad stability properties for the system with controllers at various levels. This is particularly difficult when there are constraints in the system, since some low-level controllers may not be aware of the constraints and the resulting model mismatch could cause instability. A weak version of the desired result is the observation that if we place a computed torque controller at the highest level and feedfoward controllers at lower levels of the hierarchy, the commanded torques are identical to those obtained by a single computed torque controller at the top of the hierarchy. Thus the overall system is stable. Similar results with PD or natural controllers distributed throughout the structure seem plausible, and there has been some work in this area in other contexts, such as decentralized control [84]. By using the structure inherent in controllers constructed using the primitives, it may be possible to adapt these results and strengthen them.

Finally, an important step in the progress of the control primitives is their implementation on a working system. Such an implementation would provide a testbed for different control structures (similar to that presented in [69]) and for transition controllers which operate between phases of a task.

Nonholonomic motion planning

Most current nonholonomic motion planners rely on special system structure to generate efficient motions. In some cases the structure is very specific, as evidenced by the large number of path planners for car-like robots using the special form of the kinematics for that system. More general path planners, such as the one proposed by Lafferriere and Sussmann [54], require that either the system be nilpotent or that an iterative procedure be used. In the non-nilpotent case, the iterative algorithm generates very complex paths which can steer arbitrarily close to the goal only at the cost of additional complexity. The results of Chapter 6 are somewhat complimentary—the methods can easily be applied to certain systems which are not nilpotent, but the general case requires a restrictive canonical form.

Research in efficient motion planning for general nonholonomic systems can proceed in many ways. More understanding of conditions under which a distribution can be represented by a nilpotent or chained basis would clarify the extent to which particular algorithms can be applied. On the other hand, new approaches using metric or other properties of nonholonomic distributions might lead to path planners which work for more general classes of systems. Computational approaches such as those proposed by Barraquand and Latombe [3] might also be extended to handle higher dimensional systems with very few structure requirements.

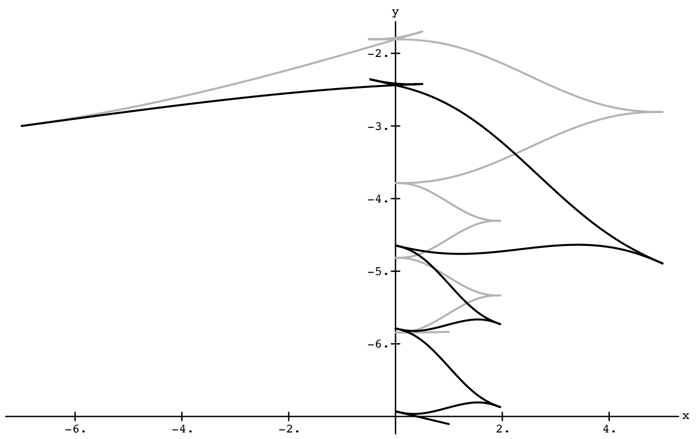


Figure 7.1: Affects of initial condition errors on open loop paths. The gray line shows a parking maneuver for an automobile. The solid path is the trajectory which is followed when the initial steering wheel angle of the car is off by 0.5 radians (approximately 3 degrees).

As mentioned at the end of Chapter 6, the work in nonholonomic motion planning thus far has been primarily in the generation of open loop trajectories. Closed loop control of nonholonomic systems is very difficult, in part because of fundamental restrictions which prohibit the existence of smooth feedback controllers which asymptotically stabilize a point. Nonetheless, it is vital to introduce closed loop control for these systems to account for initial condition and modeling errors, noise, and other effects that are encountered in any real implementation. Figure 7.1 shows an example of the effects of initial condition errors on parallel parking maneuvers for an automobile.

A possible approach to the control of nonholonomic systems is the study of controllability along a reference trajectory. If we are given a desired state trajectory, we would like to construct a controller which stabilizes the system to this trajectory. The simplest example of such a controller is a control law for steering a car down the road. While the car is moving, it is quite easy to linearize the system and design linear feedback controllers which cause the car to stay aligned with a given trajectory. In fact, if the car is moving at a constant velocity, $u_1 = v_c$, then we can write

$$\begin{aligned} \dot{x} &= g_1(x)v_c + g_2(x)u_2 \\ &= f(x) + g_2(x)u_2 \end{aligned}$$

Furthermore, this system is completely controllable as a nonlinear system. Methods for extending these results to more complicated systems are currently being pursued.

The development of closed loop controls may allow simplifications in plan-

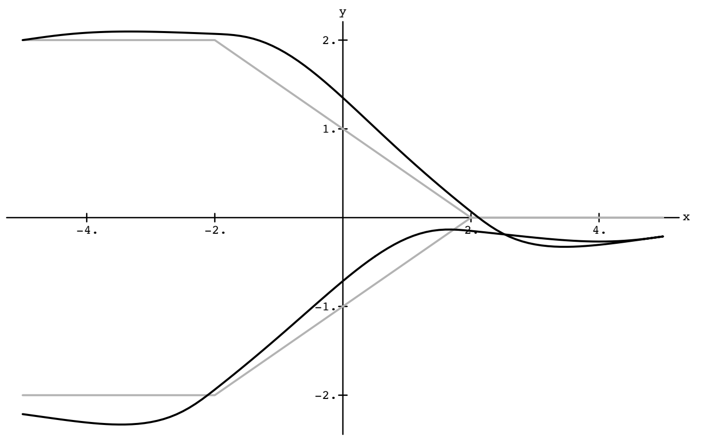


Figure 7.2: Parallel parking maneuver using piecewise feasible segments (gray lines) and closed loop control.

ning for nonholonomic systems. Rather than attempt to find an input which steers us between the initial and desired locations, we might construct a piecewise feasible trajectory which connects the two points. We then apply the linear controller about the piecewise feasible segments to implicitly define the input u. To illustrate this approach, we consider a parallel parking maneuver as shown in Figure 7.2. This controller was constructed by using piecewise linear state feedback for each feasible segment.

Finally, we consider the problem of planning for systems with a nonzero drift vector field:

$$\dot{x} = f(x) + g(x)u$$

The planning problem for this system is to steer between two equilibrium points of the system using u. If the equilibrium points lie on a connected manifold and the system is controllable at each point along the manifold, this problem can be solved for very general systems (see [36] for a specific example). However, if the start and goal position are not connected by an equilibrium manifold, it is not clear how to proceed. Although the existence of a trajectory is guaranteed by the appropriate controllability conditions (see [75]), construction of a trajectory for systems with drift is still an open problem.

Planning for nonholonomic systems with drift occurs in many contexts. For example, if we extend the model of our car to use the torques on the rear wheels and steering wheel as input, a drift vector field, $f(x)$, is introduced into the system description. It is easy to verify that the resulting nonlinear system is not controllable through its linearization, precisely because of the nonholonomic nature of the system. Thus this problem falls between the strictly nonholonomic methods considered here and the large body of work in exact and approximate linearization techniques.

Bibliography

[1] V.I. Arnold. *Mathematical Methods of Classical Mechanics.* Springer-Verlag, second edition, 1989.

[2] J. Baillieul. Kinematic programming alternatives for redundant manipulators. In *IEEE International Conference on Robotics and Automation*, pages 722–728, 1985.

[3] J. Barraquand and J-C. Latombe. Motion planning with many degrees of freedom and dynamic constraints. In *International Symposium on Robotics Research*, pages 74–83, 1989.

[4] J. Barraquand and J-C. Latombe. On nonholonomic mobile robots and optimal maneuvering. In *4th International Symposium on Intelligent Control*, Albany, NY, 1989.

[5] A. K. Bejczy. Robot arm dynamics and control. Technical Report 33-699, Jet Propulsion Laboratory, 1974.

[6] André Bellaiche, Jean-Paul Laumond, and Paul Jacobs. Controllability of car-like robots and complexity of the motion planning problem with nonholonomic constraints. In *International Symposium on Intelligent control*, Bangalore, India, 1991.

[7] A. M Bloch and N. H. McClamroch. Control of mechanical systems with classical nonholonomic constraints. In *IEEE Control and Decision Conference*, pages 201–205, 1989.

[8] A. M Bloch and N. H. McClamroch. Controllability and stabilizability properties of a nonholonomic control system. In *IEEE Control and Decision Conference*, 1990.

[9] William M. Boothby. *An Introduction to Differentiable Manifolds and Riemannian Geometry.* Academic Press, second edition, 1986.

[10] R. W. Brockett. Control theory and singular Riemannian geometry. In *New Directions in Applied Mathematics*, pages 11–27. Springer-Verlag, New York, 1981.

[11] R. W. Brockett. Asymptotic stability and feedback stabilization. In R.W. Brockett, R.S. Millman, and H.J. Sussman, editors, *Differential Geometric Control Theory*, pages 181–191. Birkhauser, 1983.

[12] R. W. Brockett. Robotic manipulators and the product of exponentials formula. In P. A. Fuhrman, editor, *Mathematical Theory of Networks and Systems*. Springer-Verlag, 1984.

[13] R. W. Brockett. On the rectification of vibratory motion. *Sensors and Actuators*, 20(1–2):91–96, 1989.

[14] R.W. Brockett. On the computer control of movement. Technical Report CICS-P-31, Center for Intelligent Control Systems, Harvard University, November 1987.

[15] R.W. Brockett. Personal communication, 1990.

[16] J.F. Canny. *The Complexity of Robot Motion Planning.* M.I.T. Press, Cambridge, 1988.

[17] John Canny, Ashutosh Rege, and John Reif. An exact algorithm for kinodynamic planning in the plane. Extended Abstract, April 1990.

[18] G. Cesareo, F. Nicolo, and S. Nicosia. DYMIR: A code for generating dynamic model of robots. In *IEEE International Conference on Robotics and Automation*, pages 115–120, 1984.

[19] Dayton Clark. Hic: An operating system for hierarchies of servo loops. In *IEEE International Conference on Robotics and Automation*, pages 1004–1008, 1989.

[20] A. Cole. *Dextrous Manipulation for Multifingered Hands: Planning and Control.* PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1989.

[21] Arlene B.A. Cole, John E. Hauser, and S. Shankar Sastry. Kinematics and control of multifingered hands with rolling contact. *IEEE Transactions on Circuits and Systems*, 34(4):398–404, 1989.

[22] John J. Craig. *Introduction to Robotics: Mechanics and Control.* Addison-Wesley, second edition, 1989.

[23] M.R. Cutkosky, R.D. Howe, and A.P. Witkin. Object-oriented modeling of robot hands. *ASME*, pages 177–186, 1986.

[24] D.C. Deno, R.M. Murray, K.S.J. Pister, and S.S. Sastry. Control primitives for robot systems. In *IEEE International Conference on Robotics and Automation*, 1990.

[25] D.C. Deno, R.M. Murray, K.S.J. Pister, and S.S. Sastry. Finger-like biomechanical systems. Technical Report ERL M90/39, Electronics Research Laboratory, University of California, Berkeley, 1990. (contains Mathematica implementation).

[26] D.C. Deno, R.M. Murray, K.S.J. Pister, and S.S. Sastry. Primitives for robot control. In *Mathematical Theory of Networks and Systems*, 1990.

[27] L.E. Dubins. On curves of minimal length with a constraint on average ccurvature , and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79:497–516, 1957.

[28] Victor Eng. *The MDL Programmer's Reference Manual.* Harvard Robotics Laboratory, 1988. First Revision.

[29] K.S. Fu, Raphael C. Gonzalez, and C. S. George Lee. *Robotics: control, sensing, vision, and intelligence.* McGraw-Hill, Inc., 1987.

[30] V. Gershkovich and A. Vershik. Nonholonomic manifolds and nilpotent analysis. *Journal of Geometry and Physics*, 5(3):407–452, 1988.

[31] V. YA. Gershkovich. Two-sided estimates of metrics generated by absolutely nonholonomic distributions on Riemannian manifolds. *Soviet Math. Dokl.*, 30(2):506–510, 1984.

[32] Claude Ghez. Introduction to the motor system. In Eric R. Kandel and James H. Schwartz, editors, *Principles of Neural Science, 2nd Ed.*, chapter 33. Elsevier, 1985.

[33] Claude Ghez. Voluntary movement. In Eric R. Kandel and James H. Schwartz, editors, *Principles of Neural Science, 2nd Ed.*, chapter 38. Elsevier, 1985.

[34] Matthew Grayson and Robert Grossman. Models for free nilpotent Lie algebras. Technical Memo PAM-397, Center for Pure and Applied Mathematics, University of California, Berkeley, 1987. (to appear in *J. Algebra*).

[35] M. Hall. *The Theory of Groups.* Macmillan, 1959.

[36] John Hauser and Richard M. Murray. Nonlinear controllers for nonintegrable systems: the acrobot example. In *American Control Conference*, 1990.

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

[38] Greg Heinzinger, Paul Jacobs, John Canny, and Brad Paden. Time-optimal trajectories for a robot manipulator: A provably good approximation algorithm. In *IEEE International Conference on Robotics and Automation*, 1990. (to appear in *IEEE J. Robotics and Automation*).

[39] Robert Hermann and Arthur J. Krener. Nonlinear controllability and observability. *IEEE Transactions on Circuits and Systems*, AC–22:728–740, 1977.

[40] Henry Hermes, Albert Lundell, and Dennis Sullivan. Nilpotent bases for distributions and control systems. *Journal of Differential Equations*, 55:385–400, 1984.

[41] G. Hinton. Some computational solutions to Bernstein's problems. In H. T. A. Whiting, editor, *Human Motor Actions — Bernstein Reassessed*, chapter 4b. Elsevier Science Publishers B.V., 1984.

[42] N. Hogan. Stable execution of contact tasks using impedance control. In *IEEE International Conference on Robotics and Automation*, 1987.

[43] Neville Hogan, Emilio Bizzi, Ferdinando A. Mussa-Ivaldi, and Tamar Flash. Controlling multijoint motor behavior. In Kent B. Pandolf, editor, *Exercise and Sport Sciences Reviews*, chapter 6, pages 153–190. Macmillan, 1987.

[44] Ping Hsu, John Hauser, and Shankar Sastry. Dynamic control of redundant manipulators. *Journal of Robotics Systems*, 6(2):133–148, 1989.

[45] Paul Jacobs, Jean-Paul Laumond, and Michel Taix. A complete iterative motion planner for a car-like robot. In *Journees Geometrie Algorithmique*, INRIA, 1990.

[46] Paul E. Jacobs. *Planning Robot Motion with Dynamic Constraints.* PhD thesis, University of California at Berkeley, Dept. of Electrical Engineering and Computer Science, 1989.

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

[48] J. Kerr and B. Roth. Analysis of multifingered hands. *International Journal of Robotics and Control*, 4(4):3–17, Winter 1986.

[49] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE RA Journal*, RA-3(1):43–53, February 1987.

[50] O. Khatib. Augmented object and reduced effective inertia in robot systems. In *American Control Conference*, 1988.

[51] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics and Control*, 5(1):90–99, 1986.

[52] D. Koditschek. Natural motion for robot arms. In *IEEE Control and Decision Conference*, pages 733–735, 1984.

[53] D.E. Koditschek and E. Rimon. Robot navigation functions on manifolds with boundary. *Advances in Applied Mathematics*, 1988. (to appear).

[54] G. Lafferriere and H.J. Sussmann. Motion planning for controllable systems without drift: a preliminary report. Technical Report SYCON-90-04, Rutgers Center for Systems and Control, June 1990.

[55] J-P. Laumond. Feasible trajectories for mobile robots with kinematic and environment constraints. In *Intelligent Autonomous Systems.* North Holland, 1987.

[56] J-P. Laumond, P. Jacobs, M. Taix, and R. Murray. Fast and exact trajectory planning for mobile robots and other systems with non-holonomic constraints. Technical Report 90318, LAAS/CNRS, Toulouse, France, 1991.

[57] Jean-Paul Laumond. Nonholonomic motion planning versus controllability via the multibody car system example. Technical Report STAN-CS-90-1339, Department of Computer Science, Stanford University, October 1990. (preprint).

[58] Jean-Paul Laumond, Michel Taix, and Paul Jacobs. A motion planner for car-like robots based on a mixed global/local approach. In *IEEE International Workshop on Intelligent Robots and Systems*, 1990.

[59] Z. Li and J. Canny. Motion of two rigid bodies with rolling constraint. *IEEE Transactions on Robotics and Automation*, 6(1):62–71, 1990.

[60] Z. Li, P. Hsu, and S. Sastry. On kinematics and control of multifingered hands. In *International Journal of Robotics and Control*, 1988.

[61] Z. Li, R. Montgomery, and M. Raibert. Dynamics and optimal control of a legged robot in flight phase. In *IEEE International Conference on Robotics and Automation*, pages 1816–1821, 1989.

[62] Zexiang Li and Lenoid Gurvits. Theory and applications of nonholonomic motion planning. (preprint), July 1990.

[63] Zexiang Li and Shankar Sastry. A unified approach for the control of multifingered robot hands. *Contemporary Mathematics*, 97:217–239, 1989.

[64] Z.X. Li and S. Sastry. Task oriented optimal grasping by multifingered robot hands. *IEEE RA Journal*, 4(1):32–44, 1988.

[65] A. Liegeois. Automatic supervisory control of the configuration and behaviior of multibody mechanisms. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-7(12), 1977.

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

[67] B. Mishra, J.T. Schwartz, and M. Sharir. On the existence and synthesis of multifinger positive grips. Technical Report No. 259, New York University, Courant Institute of Mathematical Sciences, November 1986.

[68] D. J. Montana. The kinematics of contact and grasp. *International Journal of Robotics and Control*, 7(3):17–32, June 1988.

[69] R. Murray. Experimental results in planar grasping. Master's thesis, University of California, Berkeley, 1988.

[70] R. Murray and S. Sastry. Control experiments in planar manipulation and grasping. In *IEEE International Conference on Robotics and Automation*, 1989.

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

[72] Yoshihiko Nakamura and Ranjan Mukherjee. Bi-directional approach for nonholonomic path planning of space robots. In *International Symposium on Robotics Research*, pages 101–112, 1989.

[73] V. Nguyen. The synthesis of stable force-closure grasp. Master's thesis, Massachusetts Institute of Technology, 1986.

[74] V-D. Nguyen. Constructing force-closure grasps. *International Journal of Robotics and Control*, 7(3):3–16, 1988.

[75] H. Nijmeijer and A.J. van der Schaft. *Nonlinear Dynamical Control Systems*. Springer-Verlag, 1990.

[76] B. Paden. *Kinematics and Control Robot Manipulators*. PhD thesis, University of California, Berkeley, 1986.

[77] B. Paden and R. Panja. Globally asymptotically stable 'PD+' controller for robot manipulators. *International Journal of Control*, 47(6):1697–1712, 1988.

[78] L.S. Pontryagin, V.G. Boltyanskii, R.V. Gamkrelidze, and E.F. Mishchenko. *The Mathematical Theory of Optimal Processes*. Wiley-Interscience, 1962. (translated from Russian).

[79] M. Raibert and J. Craig. Hybrid position/force control of manipulators. *ASME Journal of Dynamic Systems, Measurement and Control*, 102:126–133, June 1981.

[80] J.A. Reeds and L.A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 1990. (to appear).

[81] R. M. Rosenberg. *Analytical Dynamics of Discrete Systems.* Plenum Press, New York, 1977.

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

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

[84] N. R. Sandell, Jr., P. Varaiya, M. Athans, and M. G. Safonov. Survey of decentralized control methods for large scale systems. *IEEE Transactions on Automatic Control*, AC-23:108–128, 1978.

[85] J-P. Serre. *Lie Algebras and Lie groups.* W.A. Benjamin, New York, 1965.

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

[87] E.D. Sontag. Feedback stabilization of nonlinear systems. In *Mathematical Theory of Networks and Systems.* Birkhauser, 1989.

[88] Michael Spivak. *A Comprehensive Introduction to Differential Geometry*, volume One. Publish or Perish, Inc., Houston, second edition, 1979.

[89] M. W. Spong and M. Vidyasagar. *Dynamics and Control of Robot Manipulators.* John Wiley, 1989.

[90] Robert S. Strichartz. Sub-Riemannian geometry. *Journal of Differential Geometry*, 24:221–263, 1986.

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

[92] V.S. Varadarajan. *Lie Groups, Lie Algebras, and Their Representations.* Springer-Verlag, 1984.

[93] A. M. Vershik and V. Ya. Gershkovich. Nonholonomic problems and the theory of distributions. *Acta Applicandae Mathematicae*, 12:181–209, 1988.

[94] Stephen Wolfram. *Mathematica: A System for Doing Mathematics by Computer.* Addison-Wesley, 1989.

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