Chapter 1

Introduction

The theme of this book is building and using robot models for control. The intent is to show that model-based control leads to performance superior to control not based on carefully-constructed robot models. Thus the book naturally falls into two parts: identifying a robot model, and applying the model to control.

Building a robot model involves a mathematical formulation of its components:

- motor models for joint torque control,
- kinematic models of link lengths and of locations of joint axes, and
- inertial models of mass, center of mass, and moment of inertia for loads and links.

The parameters of these models need to be measured or estimated by appropriate procedures. The emphasis in this book is on procedures that allow the robot to calibrate itself with minimal human involvement. We fancifully envision the robot waking up in the morning, stretching to calibrate its motors, moving around a bit to identify inertial parameters, and visually observing its end effector in a few positions to identify kinematic parameters.

Controlling a robot involves both position and force variables. The role of models in position control has been extensively elaborated, but less so in force control. One contribution of this book is to show that robot

models are as important for force control as they are for position control. In addition, this book addresses the important new area of trajectory learning, where a robot fine tunes one particular trajectory through repetition. Here again the fidelity of the robot model will determine how efficiently the robot can learn.

Despite voluminous publications on the theory of robot control, ranging from PD to nonlinear control, there are almost no experimental results on performance. To be sure, complicated proofs are often given, and occasionally simulations, that supposedly validate an approach. If robot control is to become a scientific endeavor rather than just the pursuit of esoteric mathematics, it must incorporate experimentation to form a critical hypothesize-and-test loop. There simply is no other way to verify convincingly that particular control algorithms work or make a difference, or to guarantee that one is confronting real problems. Experimentation also stimulates discovery, and in fact our results on kinematic instability in force control, discussed later, serendipitously evolved from problems with an actual implementation.

What makes this book relatively unique is its experimental basis: our ideas have been tested and verified on a real robot. The experimental results in this book lend strong validity to our particular control ideas. To a large extent this lack of experimental results is due to the unavailability of high-performance manipulators that are suitable for experimentation. Commercial robots are not suitable for such reasons as high gear ratios, substantial joint friction, and slow movement. A new generation of robots based on direct drive technology is appearing that is promising for research. This book reports on experiments with our *MIT Serial Link Direct Drive Arm* (DDArm), currently one of the few manipulators available anywhere for testing advanced control strategies (Figure 1.1). We expect more of such experimentally suitable manipulators to become available soon.

Specifically, we present a number of advanced implementations in robot control, which either have not been done at all before or which provide one of the few demanding tests of a control strategy.

- Kinematic parameters of the manipulator were automatically calibrated using a motion tracking system.

- Inertial parameters of a grasped object or load were fully identified by a dynamic estimation procedure after a small number of arm movements.

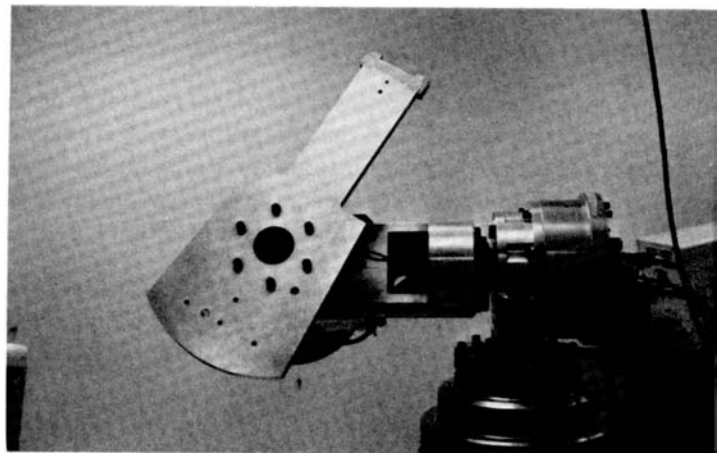

Figure 1.1: MIT Serial Link Direct Drive Arm.

- Inertial parameters of each link of the manipulator were identified in a few arm movements.
- Computed torque control and the feedforward controller have been tested for fast arm motion, where dynamic interactions are significant.
- A trajectory learning scheme was implemented that converged in three repetitions close to the repeatability level of the robot.
- A dynamically stable force controller was implemented that is stable against stiff environments but that also has a fast response and steady-state accuracy.
- Resolved acceleration force control, equivalent to the operational space method, was implemented.

In addition to the experimental results, our work has evolved a number of theoretical insights:

- We have shown that the rigid body dynamics of a manipulator can be written as linear equations in terms of the intrinsic inertial parameters, but also that certain of these parameters can only be identified in linear combinations or not at all.

- We have shown that trajectory learning schemes that do not use an accurate dynamic model of the robot converge extremely slowly.
- We have shown, along with others, that force control is essentially high-gain position control, and that the stiffness of the environment multiplies the force control gain to produce a highly underdamped system. Many robot force controllers then become dynamically unstable in the face of noise and unmodeled dynamics.
- We have found a surprising kinematic instability with certain hybrid position/force control strategies. This kind of instability is a new and important result, and occurs only because of the geometric structure of a multi-joint manipulator.

1.1 Arm Trajectory Control

By trajectory control, we intend the most general definition where position and force are simultaneously controlled. To provide a framework for arm trajectory control, a prototypical robot control architecture is outlined in Figure 1.2. The robot control problem consists of two parts:

- *planning*, where a detailed specification of manipulator position and force is given for every instance of time, and
- *control*, i.e., carrying out the plan.

For current robots, planning is usually accomplished by a human programmer, although a goal of this and much other research is to incrementally automate robot programming. Planning can be improved in several ways: the world models used in planning can be refined, better methods for solving the given task can be generated, and the planning methods themselves can be changed. These processes are assumed to be independent of improving execution of a given plan, and will not be addressed in what follows. We will take the plan as fixed, and will focus on execution and making the effector system obey a given plan more closely.

A plan is a complete specification of the motion of the robot in some coordinate system. Often the plan is expressed in *task coordinates,* for example, the Cartesian coordinates of the hand. In Figure 1.2, the variables $\mathbf{x}_d$ represent the desired position of the hand. The trajectory planner may also specify the desired hand velocity $\dot{\mathbf{x}}_d$ and acceleration $\ddot{\mathbf{x}}_d$. Whether the trajectory planner needs to do so depends on the particular controller

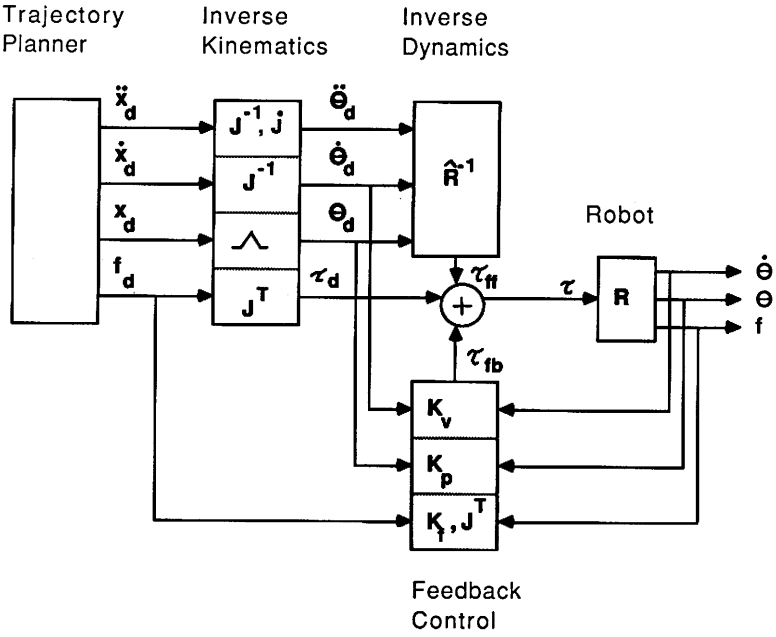

Figure 1.2: A hybrid position/force feedforward controller.

configuration. The trajectory plan may also specify the desired force $\mathbf{f}_d$, when the manipulator is in contact with the environment. The desired force may depend on the position variables, such as in stiffness control or in impedance control. One reason for planning in task coordinates is the ease of partitioning variables into position controlled $\mathbf{x}_d$ versus force controlled $\mathbf{f}_d$ variables.

In order to execute this plan, task coordinates are converted to *joint coordinates*. To indicate how this is done, it is easiest to first write the *direct kinematics* relationship between the position variables and the corresponding joint variables:

$$\mathbf{x}_d = \mathbf{f}(\boldsymbol{\theta}_d) \tag{1.1}$$

$$\dot{\mathbf{x}}_d = \mathbf{J}\dot{\boldsymbol{\theta}}_d \tag{1.2}$$

$$\ddot{\mathbf{x}}_d = \mathbf{J}\ddot{\boldsymbol{\theta}}_d + \dot{\mathbf{J}}\dot{\boldsymbol{\theta}}_d \tag{1.3}$$

The function $\mathbf{f}$ is a nonlinear transformation from the joint angles $\boldsymbol{\theta}_d$ to the endpoint positions $\mathbf{x}_d$, and depends on the kinematic parameters of the robot. The fidelity of these kinematic parameters determines how

accurately the robot may be positioned, and kinematic calibration is one of the basic model-building procedures to be discussed. The direct kinematic velocities and accelerations express linear relationships, with the aid of the Jacobian matrix $J_{ij} = \partial f_i / \partial \theta_j$. Note the slightly more involved expression for $\ddot{\mathbf{x}}_d$.

The adjective *direct* in the expression *direct kinematics* refers to the direction of the transformation, namely from the more internal variable as input (joint variables) to the more external variable as output (endpoint variables). The *inverse kinematics* transformation is required to convert the endpoint trajectory plan to joint variables, by inverting the direct kinematic relationships (1.1)-(1.3):

$$\boldsymbol{\theta}_d = \boldsymbol{\Lambda}(\mathbf{x_d}) \tag{1.4}$$

$$\dot{\boldsymbol{\theta}}_d = \mathbf{J}^{-1}\dot{\mathbf{x}}_d \tag{1.5}$$

$$\ddot{\boldsymbol{\theta}}_d = \mathbf{J}^{-1}(\ddot{\mathbf{x}}_d - \dot{\mathbf{J}}\dot{\boldsymbol{\theta}}_d) \tag{1.6}$$

The nonlinear function $\boldsymbol{\Lambda} = \mathbf{f}^{-1}$ is a one-to-many mapping, and is problematic. For a six degree-of-freedom manipulator, unless the manipulator has the proper kinematic structure, $\boldsymbol{\Lambda}$ cannot be expressed analytically (Tsai and Morgan, 1985). If there are redundancies, then some method must be chosen to resolve the redundancies (Hollerbach and Suh, 1987). In general, we now know that $\boldsymbol{\Lambda}$ cannot be a continuous function covering the whole workspace (Baker and Wampler, 1987).

The expressions (1.5) and (1.6) would not actually be used to evaluate $\dot{\boldsymbol{\theta}}_d$ and $\ddot{\boldsymbol{\theta}}_d$ for efficiency reasons, but rather customized computations would be formed that take advantage of regularities in the robot's kinematic structure (Hollerbach and Sahar, 1983). There is a large volume of literature dealing with the feasibility and computational efficiency of the inverse kinematics transformation, as well as issues of singularities and redundancies. A whole book could be devoted to inverse kinematics, and the topic is not further elaborated here. In our case, the MIT Serial Link Direct Drive Arm only has three degrees of freedom, and inverse kinematics is a relatively simple proposition.

The desired endpoint force $\mathbf{f}_d$ is directly transformed to joint torques $\boldsymbol{\tau}_d$ by the fundamental mechanical relationship:

$$\boldsymbol{\tau}_d = \mathbf{J}^T\mathbf{f}_d \tag{1.7}$$

This relation is one of *statics* rather than kinematics, but is grouped with kinematics because the transformation involves just the transpose Jacobian matrix $\mathbf{J}^T$. This relation is easy to understand, because the Jacobian

matrix is made up of the axes of rotation (Whitney, 1972) and the moment arms about them. In the present case, endpoint forces are reflected to joint torques through the moment arms contained in the Jacobian, and endpoint torques directly sum about the joint axes of rotation. In the positional transformation case, the moment arms produce linear velocities, and the axes of rotation produce angular velocities.

After the inverse kinematics and statics transformation, the next step in making the robot follow a desired trajectory is supplying appropriate commands to the actuators. The simplest approach is *feedback control*, which generates these commands by measuring the difference between where the arm is and where it is supposed to be at any instant in time and using some function (usually a linear function) of this error as the drive signal to the actuators. In Figure 1.2, feedback control is indicated by a box that contains feedback gains $\mathbf{K}_p$ for position error, $\mathbf{K}_v$ for velocity error, and $\mathbf{K}_f$ for force error, to produce an output:

$$\tau_{fb} = \mathbf{K}_p(\theta_d - \theta) + \mathbf{K}_v(\dot{\theta}_d - \dot{\theta}) + \mathbf{J}^T\mathbf{K}_f(\mathbf{f}_d - \mathbf{f}) \tag{1.8}$$

Note that the force error is computed in task coordinates, and after application of the gain $\mathbf{K}_f$ must be converted to joint torques by the transpose Jacobian $\mathbf{J}^T$.

Feedback control is useful and necessary to compensate for unpredicted disturbances. In particular, when linear feedback control is used alone, the rigid body dynamics of the manipulator are considered as disturbances. These dynamics will cause substantial trajectory errors for faster motions, unless gains in the feedback control are made correspondingly higher. Yet there are practical limits to how high gains can be set, given actuator saturation and stability problems.

To reduce the errors that need to be corrected by feedback, one approach is *feedforward control*, which uses a dynamic model $\hat{R}$ of the robot to predict actuator commands corresponding to a desired motion. This model $\hat{R}$ hopefully represents the actual robot dynamics R fairly accurately, so that the unmodeled dynamics will not cause significant perturbations. Besides the kinematic parameters, the inertial parameters of the links go into the model $\hat{R}$, and their accurate estimation is useful in reducing trajectory errors.

The *direct dynamics* R refers to the transformation from the robot input (joint torques) to the robot output (joint motion); in control terms, R represents the plant dynamics. Again, the adjective *direct* refers to the transformation from the more internal torque variables to the more external joint variables. In Figure 1.2 the calculation of driving torques

from a model of the robot and a desired trajectory is then referred to as *inverse dynamics* R^{-1}, obtained by inverting the robot plant R. Since the plant dynamics are not known exactly, only the estimated inverse $\hat{R}^{-1}$ may be used to predict the feedforward torques τ_{ff}.

In practice, there are always unexpected disturbances or modeling errors that make feedforward control imperfect, and a feedback controller is also included to compensate for unpredicted disturbances. Thus the total output τ to the robot is given by:

$$\tau = \tau_{fb} + \tau_{ff} + \tau_d \tag{1.9}$$

Note that the desired torque τ_d based on the planned endpoint force $\mathbf{f}_d$ is also included in this calculation, and can also be thought of as a feedforward command. This control scheme might be termed a *hybrid position/force feedforward controller*, and is only one way to achieve simultaneous control of position and force. Other alternatives will be considered in later sections.

All components of control can be improved using experience. One way of improving the various coordinate transformations involved in robot control is to refine the kinematic model of the robot using measurements from redundant sensing such as vision of the robot tip and joint angle sensing. To improve feedforward control the dynamic model of the robot could be refined. We could also design a better feedback controller using the past history of controller actions.

In this book we will discuss how to build and refine a model of robot dynamics to be used for predicting the appropriate actuator commands to drive the robot (feedforward control). We will discuss how to identify certain types of loads. In addition we will show the role of the robot model as the learning operator during movement practice, i.e., the robot model transforms trajectory following errors into feedforward command corrections.

1.2 Building Robot Models

The first step in any control design is the accurate modeling of the plant to be controlled. In practice, especially with the availability of automatic control design tools, this modeling step may occupy greater than 90% of the control designer's efforts. Hence, for controlling a direct drive arm, accurate modeling of the manipulator is important. The components of

Subsystem	**Input**	**Output**
Motor model	**Motor input**	**Joint torques**
Kinematic param.	**Joint angles**	**External position sensing**
Link inertial param.	**Joint torques**	**Joint movement**
Load inertial param.	Joint movement	Wrist force/torque sensing

Table 1.1: Using sensing to decouple system identification problems.

a robot that need to be modeled separately are the motor characteristics, the kinematic parameters, and the inertial parameters.

There are conceptually many different ways in which these components can be determined. For example, kinematic and inertial parameters can be determined from blueprints, and inertial parameters can be determined by taking apart the robot and weighing, counterbalancing, and swinging the pieces. The emphasis in this book is on the robot using input/output relationships between its sensors during movement to calibrate itself, with minimal human involvement. The area of engineering analysis devoted to the problem of characterizing a system from an analysis of input and output data is known as *system identification.* A particular type of system identification that identifies parameters of a known model structure is *parameter estimation.*

An insight that simplifies system identification is to use sensing to decouple different system identification problems. Different identification procedures can then be applied to the different subsystems (Table 1.1). For example, although kinematic and inertial parameters could in principle be identified with the same procedure, in practice it is easier to identify these parameters separately. Kinematic calibration can be accomplished by combining vision and joint angle sensing, while inertial parameter estimation combines joint torque sensing and joint angle sensing.

Widely different rates and types of change in system structure call for different system identification algorithms to track the changes. In the case of an arm with variable loads, arm dynamics are constant or only slowly varying, while load dynamics change rapidly as loads are picked up or put down. Arm identification can be based on a fixed model structure, while a complete load identification system must handle many different model structures, such as rigid loads, flexible loads, and time-varying loads. A wrist force/torque sensor can be used to estimate load

parameters independently of the arm dynamics.

Formulation and estimation of the model of the DDArm, consisting of motor modeling, kinematic calibration, load inertial parameter estimation, and link inertial parameter estimation, are presented in Chapters 2-5, and are briefly introduced below.

1.2.1 Motor Modeling

Motor models generally include not only the structure of the motor and amplifier, but also properties of the drive train. Although motor models can be quite complicated, they are in some sense simpler than rigid link dynamic models because motor dynamics are typically confined to a single joint. This reduces motor model identification to a single input/single output modeling problem rather than the more difficult multiple input/multiple output modeling problem.

Because direct drive arms do not have gears, there are no drive train dynamics to model. Hence motor modeling is much easier, because one does not have to contend with friction and backlash. Moreover, the joints are intrinsically stiff, so no extra dynamics are introduced by flexible gear trains. Flexibility in gear trains causes a loss of endpoint precision, and is one of the non-geometric factors that makes kinematic calibration more complicated (Whitney, Lozinski, and Rourke, 1986). Another factor is eccentricity in the gears, which introduces a periodic error in position.

The issue then reduces to how torque can be derived from the motor, given its structure and drive amplifier. The two basic alternatives for an electric motor are current measurement or external torque measurement. Of the two alternatives, current measurement is less accurate, because it is based on a model of the motor: its winding structure, commutation scheme, and magnetic pole location for electromagnetic motors. Sources of error derive from cogging torques and imperfect position measurement. With careful modeling and design, nevertheless, torque accuracies on the order of 1% are now possible in new direct-drive motors (see Chapter 10).

External torque measurement is more accurate, provided that one can design a sensor for the motor axis. The joint structure definitely becomes more complex, especially if a flexible element is introduced to permit sufficient strain to be produced for measurement (Asada, Youcef-Toumi, and Lim, 1984). Additional flexibility may introduce extra dynamics into the system, as well as potential loss of endpoint resolution.

In Chapter 2, the structure of the DDArm is discussed, with particular attention to joint torque control by the motors. Our torque measurement

scheme is based on current measurement. Additional points of discussion in that chapter include amplifier nonlinearities, significant inductances, and position inaccuracies of our motors.

1.2.2 Kinematic Calibration

No robot is ever manufactured perfectly. There will be slight variations in kinematic parameters — link lengths that are a little off nominal values or neighboring joint axes that are not quite parallel. The result can be tip inaccuracies on the order of several millimeters for common industrial robots. Kinematic calibration has come to be recognized as a necessary process for any robot, because machining has limits of precision, assembly may be imperfect, and striving for even greater precision is costly.

Other aspects of kinematic calibration, not treated here, include locating the robot with respect to an external reference frame and locating an object within the grasp of the robot. These processes obviously must be carried out whenever a robot is moved or a new object is picked up.

To proceed with kinematic calibration, one formulates how the endpoint position varies with the kinematic parameters. In (1.1), the endpoint position was written only as a function of the joint angles, namely $\mathbf{x} = \mathbf{f}(\boldsymbol{\theta})$. For kinematic calibration, this relation becomes:

$$\mathbf{x} = \mathbf{f}(\boldsymbol{\theta}, \boldsymbol{\alpha}, \mathbf{a}, \mathbf{s}) \tag{1.10}$$

where the Denavit-Hartenberg (1955) parameters $\boldsymbol{\alpha}$, the skew angles between neighboring joint axes, $\mathbf{a}$, the link lengths, and $\mathbf{s}$, the joint offsets, represent the most commonly used kinematic parameters. Taking the first difference,

$$\Delta\mathbf{x} = \frac{\partial \mathbf{f}}{\partial \boldsymbol{\theta}}\Delta\boldsymbol{\theta} + \frac{\partial \mathbf{f}}{\partial \boldsymbol{\alpha}}\Delta\boldsymbol{\alpha} + \frac{\partial \mathbf{f}}{\partial \mathbf{a}}\Delta\mathbf{a} + \frac{\partial \mathbf{f}}{\partial \mathbf{s}}\Delta\mathbf{s} \tag{1.11}$$

The term $\partial\mathbf{f}/\partial\boldsymbol{\theta}$ is just the ordinary Jacobian matrix $\mathbf{J}$, defined in (1.2). The other partial derivatives of $\mathbf{f}$ with respect to the remaining three Denavit-Hartenberg parameters are also different Jacobian matrices.

The difference $\Delta\mathbf{x}$ may be interpreted as the error in position of the endpoint, obtained by subtracting the computed from the measured position. The differences $\Delta\boldsymbol{\theta}$, $\Delta\boldsymbol{\alpha}$, $\Delta\mathbf{a}$, and $\Delta\mathbf{s}$ may be viewed as the corrections to the kinematic parameters. These corrections may be solved for by combining many measurements throughout the workspace and inverting

(1.11):

$$[\Delta\boldsymbol{\theta} \quad \Delta\boldsymbol{\alpha} \quad \Delta\mathbf{a} \quad \Delta\mathbf{s}] = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial \boldsymbol{\theta}} & \frac{\partial \mathbf{f}}{\partial \boldsymbol{\alpha}} & \frac{\partial \mathbf{f}}{\partial \mathbf{a}} & \frac{\partial \mathbf{f}}{\partial \mathbf{s}} \end{bmatrix}^{\dagger} \Delta\mathbf{x} \tag{1.12}$$

where the † indicates the generalized inverse of the enclosed matrix, giving a least squares solution for the parameter corrections. Since kinematic calibration is a nonlinear estimation problem, one must iterate this procedure by reevaluating the Jacobians at the updated parameter values.

The most difficult aspect of kinematic calibration is obtaining accurate measurements of the endpoint position. Past work in kinematic calibration has involved calibration fixtures with precision points (Hayati and Roston, 1986), special apparatuses (Stone, Sanderson, and Neuman, 1986), or external position measurements with theodolites (Whitney, Lozinski, and Rourke, 1986). Particularly the latter work achieved extremely good results with theodolites, which are manually operated surveying instruments, but one drawback is the extensive human involvement in making the theodolite measurements. Special calibration fixtures also require extensive human involvement, and do not address the issues of locating a robot or an object in the robot's grasp.

Our approach sacrifices to some extent the precision obtained with the above techniques for convenience, so that the robot could in principle calibrate itself. Recently, three-dimensional motion measuring systems have become available that locate points with accuracies on the order of a millimeter and resolutions around a quarter of a millimeter in a volume typical of robot workspaces. We have used one such system, the Watsmart System, which is a commercial opto-electronic apparatus based on triangulation of LED markers. The Watsmart system allows fast tracking of multiple LED markers, and hence the endpoint can be tracked in real time. Kinematic calibration is a straightforward, convenient application of this system, although the accuracies are not comparable to the specialized techniques mentioned above. In Chapter 3 an iterative procedure based on the linearized kinematic equations is presented, along with experimental results using the Watsmart System.

1.2.3 Load Estimation

Since a load is essentially a part of the last link, the knowledge of the inertial parameters of manipulator loads is important for accurate control of manipulators. One alternative is to use object models and precise

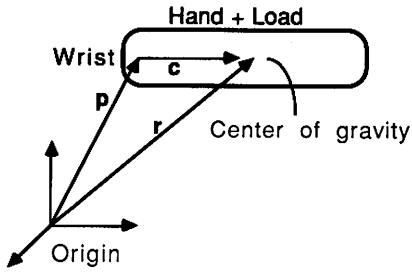

Figure 1.3: Locating the hand plus load center of gravity relative to the wrist and origin.

knowledge of grasping to predict how the load affects the inertial parameters of the last link. A more general alternative would not rely on object models or exact grasping, but would allow a robot to use sensing and an appropriate system identification procedure to identify the load itself.

The general procedure for load estimation begins by writing the Newton-Euler equations for rigid-body motion:

$$\mathbf{f} = m\ddot{\mathbf{r}} \tag{1.13}$$

$$\mathbf{n} = \mathbf{I}\dot{\omega} + \omega \times \mathbf{I}\omega \tag{1.14}$$

In Newton's equation, $\mathbf{f}$ is the net force on the body, m is its mass, and $\ddot{\mathbf{r}}$ is the acceleration of the center of gravity. In Euler's equation, $\mathbf{n}$ is the net torque, $\mathbf{I}$ is the inertia and ω is the angular velocity. Since the position of the center of gravity is not generally known, we decompose $\mathbf{r}$ into the position of the wrist $\mathbf{p}$ plus the position of the load's center of gravity relative to the wrist $\mathbf{c}$ (Figure 1.3). Then

$$\ddot{\mathbf{r}} = \ddot{\mathbf{p}} + \dot{\omega} \times \mathbf{c} + \omega \times (\omega \times \mathbf{c}) \tag{1.15}$$

Substituting into (1.13),

$$\mathbf{f} = m\ddot{\mathbf{p}} + \dot{\omega} \times m\mathbf{c} + \omega \times (\omega \times m\mathbf{c}) \tag{1.16}$$

Note that the inertial parameters of mass m, mass moment $m\mathbf{c}$, and inertia $\mathbf{I}$ appear linearly in the Newton-Euler equations (1.14) and (1.16), even though the dynamic equations as a whole are nonlinear. The nonlinearity appears in the kinematic terms, however, which are known as a result of measurement during the motion. By measuring force and

torque at the wrist, linear least squares estimation can then be used by combining a number of readings and inverting (1.14)-(1.16):

$$[m \quad mc \quad \mathbf{I}] = \boldsymbol{A}(\ddot{\mathbf{p}}, \omega, \dot{\omega})^{\dagger}[\mathbf{f} \quad \mathbf{n}] \tag{1.17}$$

where the matrix $\boldsymbol{A}$ is a function of the kinematic parameters, obtained by rearranging the Newton-Euler equations, and † once again denotes the generalized inverse.

Some previous work in load estimation has specified special test motions, where one joint moves at a time to identify inertia components. The method we present can identify these parameters simultaneously as a result of a single motion. Also, some investigators specify the use of joint torques rather than wrist force/torque sensing, but as mentioned above this couples the dynamics of the arm into the dynamics of the load and makes the estimation less accurate. Finally, most of the previous work has not involved experimentation, so that investigators have not been able to say how well the various load inertial parameters can be estimated.

Our method, presented in Chapter 4, identifies the load inertial parameters after a single motion. Experimental results are presented both on a PUMA 600 robot and on the DDArm.

1.2.4 Link Estimation

The inertial parameters, i.e., the mass, the location of center of mass, and the moments of inertia of each rigid body link of a robot are usually not known even to the manufacturers of the robots. Robots are usually designed to satisfy kinematic specifications, but the inertial parameters are incidental attributes. Since commercial robots are invariably controlled by simple independent-joint PID control, there is no impetus by the manufacturer to determine the inertial parameters accurately since a dynamic model is not used.

As mentioned above, link inertial parameters can be determined experimentally by disassembling the robot, and then weighing the pieces for mass, counterbalancing for center of mass, and swinging for moments of inertia (Armstrong, Khatib, and Burdick, 1986). Besides requiring intensive human involvement, this procedure introduces considerable measurement difficulties. Counterbalanced points have to be referred somehow to the joint axes, while some components of inertia are difficult to determine by pendular motion.

Another approach is CAD modeling of the parts, where computerized geometric information can be combined with specific gravities of materials to estimate the inertial parameters. Again, this approach requires intensive human involvement, and is also subject to modeling errors. The approach we present again emphasizes the robot calibrating itself, and will be shown to compare favorably to the other alternatives.

Link inertial parameter estimation is similar to load estimation, in that each link can be viewed as a load to the proximal joints. The inertial parameters again appear linearly in the dynamic equations, even though the multi-link rigid body dynamics are nonlinear. A major difference from load estimation is that there is not a six-axis force/torque sensor at each joint, and only the component of torque about the joint axis can be measured. The inverse dynamics equation can be written as:

$$\tau = \hat{R}^{-1}(\theta, \dot{\theta}, \ddot{\theta}) \tag{1.18}$$

Since the inertial parameters appear linearly in $\hat{R}^{-1}$, this can be rewritten as:

$$\tau = Q(\theta, \dot{\theta}, \ddot{\theta})[m\ m\mathbf{c}\ \mathbf{I}] \tag{1.19}$$

where the inertial parameters for all the links have been separated out from $\hat{R}^{-1}$, leaving a function Q of the kinematic parameters only.

Unfortunately, it is not possible to invert (1.19) as before to obtain the link inertial parameters. Aside from the limited sensing at each joint, the proximal links do not undergo general motion because of restricted degrees of freedom. The manifestation of limited sensing and movement in estimation is twofold.

1. Not all parameters can be identified, since they do not influence the joint torques. For example, consider link 1 attached to the robot base by joint 1. Only the link 1 inertia about the first joint is reflected in the joint 1 torque for an arm on a stationary base; all other link 1 inertial parameters cannot be identified.

2. Other inertial parameters can only be determined in linear combinations.

In Chapter 5, we discuss various ways of solving this rank-deficient least squares problem, and present experimental results for our DDArm. One practical aspect is that the parameters that cannot be estimated are unimportant for control, because they do not affect the torques necessary to drive the robot.

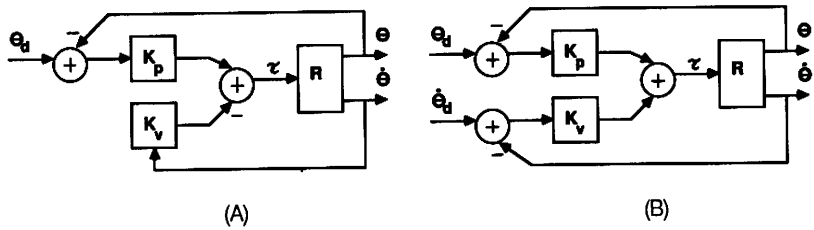

Figure 1.4: Independent-joint PD control with (B) and without (A) velocity reference.

1.3 Position Control

An example of a model-based position/force controller was presented in the beginning of this chapter. This section goes into greater detail about position control, examining the alternatives of how a robot model can be incorporated to improve position control. In a later section we discuss how force control is also improved by using a robot model. We will discuss three basic types of position controllers proposed for robotics:

- independent-joint proportional-derivative (PD) control,
- feedforward control (Liégeois, Fournier, and Aldon, 1980), and
- computed torque control (Paul, 1972).

Independent-joint PD control (Paul, 1981) is by far the most popular feedback controller for robots. As mentioned earlier, independent-joint PD feedback control is distinguished from feedforward control in that the latter uses a model of the robot. Two particular kinds of feedforward control are the *feedforward controller* and the *computed torque controller*, which differ in how the dynamic model is used in conjunction with a feedback loop. Unfortunately, there is a potential confusion in the term *feedforward controller*, which has come to mean in robotics a particular kind of feedforward control. We emphasize again that while much has been written about these and other kinds of robot control, there are too few experimental evaluations of them.

1.3.1 Independent-Joint PD Control

Feedback control can be defined as any control action based on the actual state history of the controlled system. We will restrict the focus to

independent-joint PD control. The basic structure of this controller is shown in Figure 1.4A. A reference position $\boldsymbol{\theta}_d$ is compared to an actual position $\boldsymbol{\theta}$, and the difference is multiplied by a position gain $\mathbf{K}_p$ to produce an output to the actuator or plant. To provide stability, a damping term is added to the output based on the actual velocity $\dot{\boldsymbol{\theta}}$ multiplied by a velocity gain $\mathbf{K}_v$:

$$\boldsymbol{\tau} = \mathbf{K}_p(\boldsymbol{\theta}_d - \boldsymbol{\theta}) - \mathbf{K}_v\dot{\boldsymbol{\theta}} \tag{1.20}$$

The output torque $\boldsymbol{\tau}$ is applied to the robot, represented by the direct dynamics transformation R, which undergoes a motion $\boldsymbol{\theta}(t)$. Note that feedback control does not include a model of the robot, and is purely error driven. This controller is duplicated for each joint θ of the robot, and the control action at each joint is totally independent of control actions at other joints. Hence the name *independent-joint PD control* is derived.

One feature of this form of PD control is that there is heavy damping during the fastest parts of movement, where it is not particularly needed. To remedy this situation, *PD control with velocity reference* is often proposed (Figure 1.4B), which now requires the trajectory planner to specify the desired velocity $\dot{\boldsymbol{\theta}}_d$ as well as the desired position. Now the damping during the fast parts of movement is reduced to $\mathbf{K}_v(\dot{\boldsymbol{\theta}}_d - \dot{\boldsymbol{\theta}})$.

$$\boldsymbol{\tau} = \mathbf{K}_p(\boldsymbol{\theta}_d - \boldsymbol{\theta}) - \mathbf{K}_v(\dot{\boldsymbol{\theta}}_d - \dot{\boldsymbol{\theta}}) \tag{1.21}$$

Much effort has gone into robot feedback controller design, but there are limits on feedback control. Many proofs of stability for various robot feedback controllers amount to infinite actuator arguments, since it is presumed that actuators do not limit the ability to increase gains to the point where disturbances can be overcome and errors reduced to a desired level. In reality, actuator saturation prevents this easy solution. Moreover, gains cannot be increased to high levels to reduce errors, because of potential instabilities that may arise from modeling error, parameter variations, and measurement or command noise (Åström and Wittenmark, 1984). In general, non-minimum phase elements such as delays and right half plane zeros set limits on maximum feedback gains.

Control of terminal compliance or, more generally, impedance has been proposed as a goal for robotic control. Force control may require limiting feedback gains, if the desired compliance is implemented as a low gain position servo. For non-redundant robots with no terminal force/torque sensing, choosing an impedance specifies the feedback controller completely. The use of force sensors at the interface between the robot and its load or environment may allow differential rejection of modeling errors and external forces, but has not yet been shown to work well.

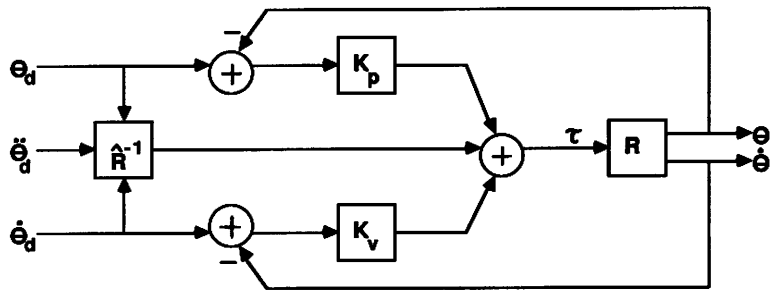

Figure 1.5: Feedforward controller.

1.3.2 Feedforward Controller

The *feedforward controller* (Figure 1.5) predicts the output to the actuators from a dynamic model $\hat{R}^{-1}$ of the robot, based on desired position, velocity, and acceleration. It represents just the position control part of Figure 1.2. Once again, to alleviate a potential source of confusion, it should be repeated that the feedforward controller is a particular kind of the more generic feedforward control.

The trajectory planner must specify not only the desired velocity, but the desired acceleration $\ddot{\theta}_d$ as well. This should be no problem, because the position commands, being internal to the controller, are assumed to have negligible noise and derivative operators can accurately be applied to the command. In parallel with the feedforward computation, there is an independent-joint PD controller with velocity reference. The sum of the feedforward output and the feedback controller output then drives the robot:

$$\tau = \hat{R}^{-1}(\theta_d, \dot{\theta}_d, \ddot{\theta}_d) + \mathbf{K}_p(\theta_d - \theta) + \mathbf{K}_v(\dot{\theta}_d - \dot{\theta}) \qquad (1.22)$$

Presumably the feedforward computation has compensated for the dynamics of the robot fairly well, and only small perturbations or unmodeled dynamics remain for the feedback controller to compensate. Hence the gains of the PD controller can be kept low to avoid stability problems.

One issue is the fidelity of the dynamic model $\hat{R}$ of the robot. If the model is not very good, then the feedforward computation can degrade system performance. Our experiments in identification and control with the direct drive arm, however, indicate that this is not a problem.

That the dynamics computation $\hat{R}^{-1}(\theta_d, \dot{\theta}_d, \ddot{\theta}_d)$ in the feedforward controller is done on the basis of the planned trajectory, and hence can

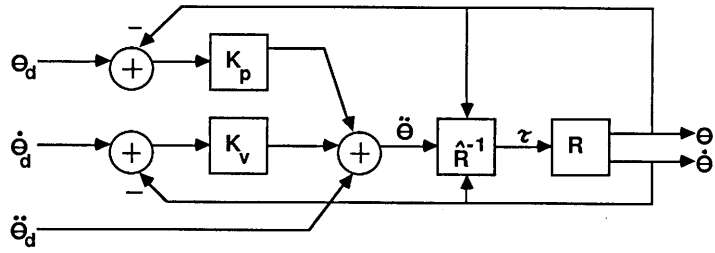

Figure 1.6: Computed torque control.

be done off-line, is an advantage over the computed torque controller discussed next. This may have been an important issue in the past, but it is much less of one today due to increases in computational power of real-time control systems (Narasimhan et al., 1986), and in the efficiency of dynamics computation (Hollerbach and Sahar, 1983). A disadvantage of the feedforward controller is that the PD portion of the controller acts independently of the dynamics and produces perturbations at neighboring joints. That is to say, a corrective torque at one joint perturbs the other joints, whereas ideally the corrective torques would decouple joint interactions. It is to this latter problem that computed torque control is addressed.

1.3.3 Computed Torque Control

In *computed torque control*, the feedback controller sends its output through the dynamic model (Figure 1.6). The feedback control law comprises an independent-joint PD controller with velocity reference, plus the desired acceleration. This yields a corrected acceleration which is then input to the inverse dynamics model:

$$\ddot{\boldsymbol{\theta}}^* = \ddot{\boldsymbol{\theta}}_d + \mathbf{K}_p(\boldsymbol{\theta}_d - \boldsymbol{\theta}) + \mathbf{K}_v(\dot{\boldsymbol{\theta}}_d - \dot{\boldsymbol{\theta}}) \tag{1.23}$$

$$\boldsymbol{\tau} = \hat{R}^{-1}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}, \ddot{\boldsymbol{\theta}}^*) \tag{1.24}$$

Note that $\ddot{\boldsymbol{\theta}}^*$, derived from the feedback law, is the nominal rather than the actual acceleration. The feedforward computation $\hat{R}^{-1}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}, \ddot{\boldsymbol{\theta}}^*)$ is done on the basis of the actual trajectory, so that the dynamics computation must be on-line.

Computed torque control is a form of control called *non-linearity cancellation*, because if the dynamic model is exact ($\hat{R} = R$), the nonlinear

dynamic perturbations are exactly canceled ($\hat{R}^{-1}(R(\tau)) = I(\tau)$, where $I(\tau) = \tau$ is the identity transformation). What is left is a decoupled linear system that can be controlled according to standard techniques. In principle, computed torque control should be more accurate than the feedforward controller, because the action of the feedback controller is decoupled through the dynamics.

As will be seen in Chapter 6, surprisingly the experimental results do not bear out this expectation. Computed torque control and the feedforward controller were about equally accurate in following a trajectory. These results cast into doubt the utility of the ever more sophisticated, or at least complicated, nonlinear feedforward controllers that are being proposed. Once again the importance of experimental testing should be emphasized.

1.4 Trajectory Learning

The implementation of the model building procedures on the DDArm reveals that good models can be identified quickly and are useful for control. Nevertheless, the models used to represent the arm and load dynamics have limited degrees of freedom, and cannot represent the full complexity of the true dynamics. The models do not represent well deviations from the assumed model structure, which are always present to some degree. Thus, on any particular trajectory execution, a rigid body dynamics model will have small errors. Changing the model parameters to fit this trajectory more exactly will degrade performance on other trajectories. What is required is an additional level of modeling that allows representation of fine details of the dynamics.

A solution to this problem, *trajectory learning*, has recently become an important topic in robotics, and arises from the recognition that robots often repeat the same motion over and over again. Hence the possibility arises to tune the output for this single trajectory through repetition to reduce the errors to a very low level. The key issues here are the stability and convergence rate of the iterative process, and how to design the learning operator. One drawback of this approach is that it only produces the appropriate command for a single trajectory. There is little guidance at present as to how to modify that command for similar trajectories.

In trajectory learning, torques are initially generated based on some form of feedforward or feedback controller. The torque profiles are remembered, and refined on a point-by-point basis after each iteration of

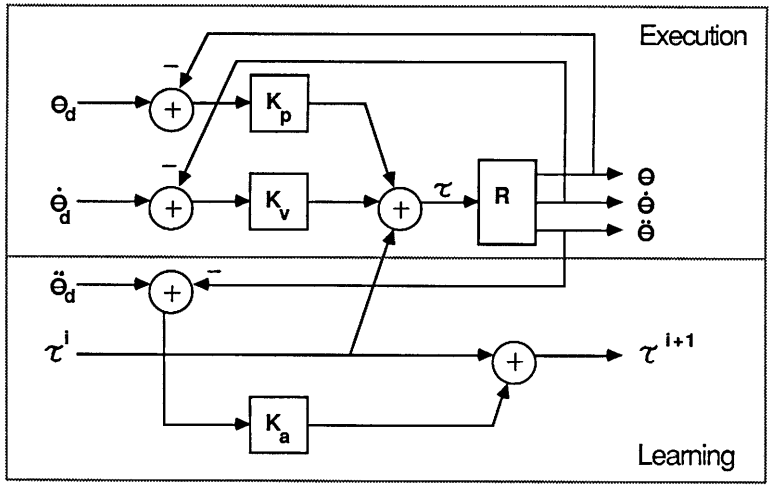

Figure 1.7: Trajectory learning scheme.

the trajectory. The errors during the trajectory are also remembered, and are converted off-line into corrective torques to the torque profiles. One such learning scheme is presented in Figure 1.7, and is similar to one presented in (Arimoto et al., 1984a-c, 1985):

$$\tau = \mathbf{K}_p(\theta_d - \theta) + \mathbf{K}_v(\dot{\theta}_d - \dot{\theta}) + \tau^i \quad (1.25)$$

$$\tau^{i+1} = \tau^i + \mathbf{K}_a(\ddot{\theta}_d - \ddot{\theta}) \quad (1.26)$$

In (1.25) there is a PD position controller with velocity reference, and in addition there is a feedforward torque $\tau^i(t)$ derived from a learning operator after the ith repetition of a movement. At the initial repetition the feedforward torque $\tau^0(t) = 0$, so that the controller is a pure PD position controller the first time. In Figure 1.7, this phase is labeled **Execution**. After the ith repetition, the acceleration error is multiplied by a gain term, and added to the current feedforward torques τ^i in (1.26) to yield the feedforward torques τ^{i+1} for the next movement repetition. This phase is labeled **Learning** in the figure.

This scheme has been shown to converge eventually to a torque profile τ^N that drives the manipulator along a trajectory with small errors. One aspect of this scheme is that a dynamic model $\hat{R}$ of the manipulator is not used. This can be viewed as an advantage in those cases in which a model is not easily obtained. A disadvantage is that the convergence can be very slow.

The relation (1.26) derives a corrective torque by constant scaling of acceleration, but we know from the inverse dynamics R^{-1} that torque and acceleration are not this simply related:

$$\tau = R^{-1}(\theta, \dot{\theta}, \ddot{\theta}) = \mathbf{H}(\theta)\ddot{\theta} + \dot{\theta} \cdot \mathbf{C}(\theta) \cdot \dot{\theta} + \mathbf{g}(\theta) \qquad (1.27)$$

where $\mathbf{H}$ is the inertia matrix, $\dot{\theta} \cdot \mathbf{C} \cdot \dot{\theta}$ represents the centripetal and Coriolis torques, and $\mathbf{g}$ represents the gravity torques. This suggests that a better learning scheme would take the manipulator dynamics into account, because one must use an accurate model of the controlled system to make sense of trajectory errors, i.e., convert the errors into corrections to feedforward commands.

Without an accurate model, attempts to improve trajectory performance can actually degrade performance. We have mathematically analyzed the effect of various proposed trajectory learning algorithms such as in Figure 1.7, and can explain why one learning operator works better than another. The most important result is that the convergence rates of the algorithms are determined by the quality of the learning operators used. We can put mathematical bounds on acceptable modeling error for the linear case.

The model used in the trajectory learning work of Chapter 7 is the identified arm model presented in Chapter 5. Thus, starting with only knowledge of system structure, we have demonstrated a system that can build a general model of itself after only three or four movements, and then can learn to execute any particular trajectory to almost the limits of the system repeatability in an additional three or four movements. This work demonstrates the role of knowledge in analyzing past behavior and correcting previous mistakes, and should be compared to other trajectory learning schemes, to table-based schemes to learn arm dynamics, and to much of traditional adaptive control.

1.5 Force Control

Force control is the most general form of trajectory control, because the manipulator is allowed to contact the environment as it executes a trajectory. Instead of just position variables to plan and control, there are now additionally force variables to plan and control. When we use the term *force control,* we mean the simultaneous control of both force and position.

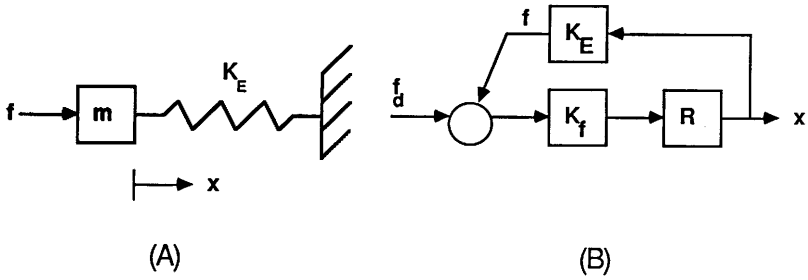

Figure 1.8: (A) A simple model of the robot in contact with the environment. (B) A simple proportional force controller.

Force control is much less well understood than position control, and there have been correspondingly fewer real implementations. As mentioned before, conventional robots are not well suited for implementing force control algorithms since they are essentially positioning devices. Therefore, previous implementations seldom produced satisfactory results (Caine, 1985), and researchers in the past have experienced significant instability problems associated with force controllers (Whitney, 1987). Another goal of this book is to understand some of the stability and performance problems associated with force control, and suggest and demonstrate some remedies to those problems using the DDArm.

In this section, two different aspects of stability in force control are discussed. The first aspect we call *dynamic instability,* which arises when manipulators are in contact with stiff environments. This instability arises whether the robot has single or multiple joints, and is the source of instability in force control most commonly described. The second aspect we call *kinematic instability,* and arises in some Cartesian-based force control schemes only for certain multiple-joint manipulators. We also introduce various types of force controllers, beginning with single-joint controllers and then moving to multiple-joint controllers.

1.5.1 Dynamic Instability in Force Control

Many robot force controllers go unstable during hard contact, such as against metal. The robot chatters uncontrollably, bouncing back and forth against the surface. To illustrate the problem, Figure 1.8A shows a simple model of the robot and its environment. The robot is modeled as a pure mass m, and the environment is modeled as a pure stiffness

K_E. We presume there is a stiff and massless force sensor between the robot and environment that measures the contact force $f = K_E x$, where x is the displacement. The force controller in Figure 1.8B is a simple proportional controller, multiplying the force error $f_d - f$ by a gain K_f, where f_d is the desired force. Hence the applied force f from the force control law is:

$$f = K_f(f_d - K_E x) \tag{1.28}$$

The pertinent feature of this equation is that *force control is essentially high-gain position control.* The stiffness of the environment multiplies the force control gain, yielding a large effective position gain of $K_f K_E$. Systems with such large feedback gains in general exhibit unstable behavior. Sources of instability include unmodeled dynamics, such as flexibility in the manipulator joints or link. Since flexibility is present in all manipulators, the chattering behavior mentioned earlier is manifested in virtually all force controllers.

There are a number of ways in which this dynamic instability can be overcome. One way is to dominate the stiffness of the environment with a soft skin or covering or with a soft spring attaching the force sensor to the robot. Disadvantages with this approach include loss of position resolution and a reduction in the speed of response. The damping in the controller can also be elevated to match the high position gain, but again the response speed would be slowed.

In Chapter 8 we propose a two-part force controller that is dynamically stable but that is still fast and accurate. The fast part of the controller is based on open-loop joint torque control. The stiffness of the environment does not enter into this feedback loop because the external force sensor is not being employed there, and hence the response is always stable. A slower force control loop based on an external force sensor is also used to maintain steady-state accuracy, but the force sensing is low-pass filtered to prevent the environmental stiffness from destabilizing the system. Experimental results are presented with the DDArm.

1.5.2 Cartesian-Based Position Control

As a way of introducing the Cartesian-based force controllers of the next section, it will be helpful to discuss Cartesian-based position controllers at this point. The position controllers discussed earlier (independent-joint PD control, the feedforward controller, and computed torque control) are based on joint coordinates. If the trajectory is initially specified in

terms of Cartesian coordinates of the endpoint, then inverse kinematic transformations are required to convert to joint angles.

It is also possible to specify the control law in terms of Cartesian coordinates rather than in terms of joint coordinates. The reason that Cartesian-based position control has been proposed is that the control law is often best cast into the task variables according to which the trajectory is planned. This is particularly true of force control, where variables are partitioned into those that can be controlled for position versus those that are controlled for force. For example, a *Cartesian-based PD position controller* could be defined by:

$$\mathbf{f} = \mathbf{K}_p(\mathbf{x}_d - \mathbf{x}) + \mathbf{K}_v(\dot{\mathbf{x}}_d - \dot{\mathbf{x}}) \tag{1.29}$$

where $\mathbf{f}$ is the endpoint force that the manipulator generates in response to a perturbation. That is to say, the endpoint of the manipulator acts like a spring plus damper.

To evaluate this control law, joint positions and velocities must be converted to endpoint positions and velocities. The endpoint force can be converted into joint torques in several ways. One way is to directly convert to joint torques by the relation $\tau = \mathbf{J}^T\mathbf{f}$ (Figure 1.9):

$$\tau = \mathbf{J}^T(\mathbf{K}_p(\mathbf{x}_d - \mathbf{x}) + \mathbf{K}_v(\dot{\mathbf{x}}_d - \dot{\mathbf{x}})) \tag{1.30}$$

This equation is closely related to Salisbury's stiffness controller, discussed in Chapter 9.

Another way (Figure 1.9B) is to note that $\delta\mathbf{x} = \mathbf{x}_d - \mathbf{x}$ is usually small and can be approximated by the incremental relation $\delta\mathbf{x} = \mathbf{J}\delta\boldsymbol{\theta}$, where $\delta\boldsymbol{\theta} = \boldsymbol{\theta}_d - \boldsymbol{\theta}$. Similarly, $\delta\dot{\mathbf{x}} = \mathbf{J}\delta\dot{\boldsymbol{\theta}}$, where $\delta\dot{\mathbf{x}} = \dot{\mathbf{x}}_d - \dot{\mathbf{x}}$ and $\delta\dot{\boldsymbol{\theta}} = \dot{\boldsymbol{\theta}}_d - \dot{\boldsymbol{\theta}}$. Inverting the Jacobian yields $\delta\boldsymbol{\theta} = \mathbf{J}^{-1}\delta\mathbf{x}$ and $\delta\dot{\boldsymbol{\theta}} = \mathbf{J}^{-1}\delta\dot{\mathbf{x}}$. The gains $\mathbf{K}_p$ and $\mathbf{K}_v$ are now interpreted as joint position and velocity gains, so that

$$\tau = \mathbf{K}_p\mathbf{J}^{-1}(\mathbf{x}_d - \mathbf{x}) + \mathbf{K}_v\mathbf{J}^{-1}(\dot{\mathbf{x}}_d - \dot{\mathbf{x}}) \tag{1.31}$$

This equation is closely related to the hybrid position/force controller of Raibert and Craig, discussed in the next section.

These different implementations of a Cartesian-based PD position controller are pure feedback controllers and do not incorporate a dynamic model of the robot. One could define Cartesian-based position controllers analogous to both the feedforward controller and computed torque control. The position control part of Figure 1.2 would be a hypothetical implementation of a *Cartesian-based feedforward controller.* In actual

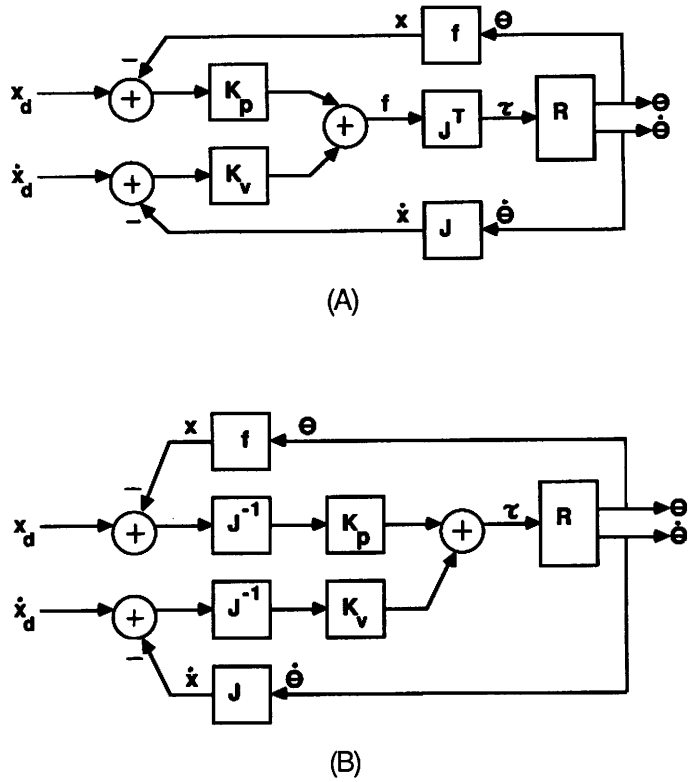

Figure 1.9: (A) Cartesian position control based on the transpose Jacobian matrix. (B) Cartesian position control based on the inverse Jacobian matrix.

implementations and the literature, only the Cartesian-based computed torque controller has been proposed, and is called *resolved acceleration position control* (Luh, Walker, and Paul, 1980b).

Resolved acceleration position control is quite similar to computed torque control, except that the desired trajectory and the feedback law are expressed in terms of task coordinates $\mathbf{x}$ (Figure 1.10):

$$\ddot{\mathbf{x}}^* = \ddot{\mathbf{x}}_d + \mathbf{K}_p(\mathbf{x}_d - \mathbf{x}) + \mathbf{K}_v(\dot{\mathbf{x}}_d - \dot{\mathbf{x}}) \tag{1.32}$$

Direct kinematic transformations are required to compute the actual endpoint positions and velocities from the joint positions and velocities, and an inverse kinematic transformation is required to convert the nominal

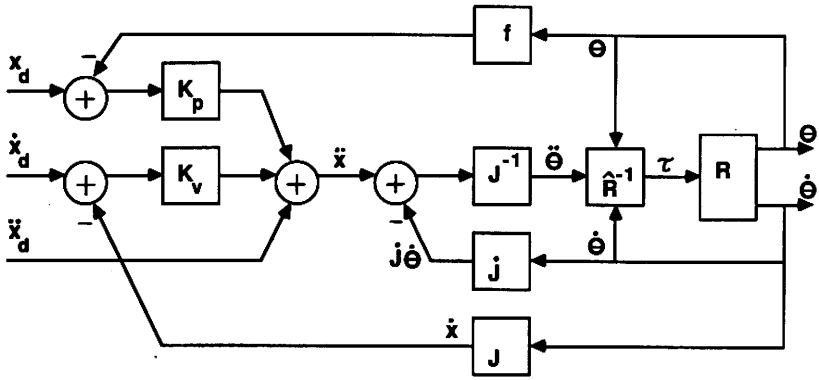

Figure 1.10: Resolved acceleration position control.

endpoint acceleration to a nominal joint acceleration. The nominal joint acceleration is then substituted into the inverse dynamics to yield the joint torques, according to:

$$\ddot{\theta}^* = \mathbf{J}^{-1}(\ddot{\mathbf{x}}^* - \dot{\mathbf{J}}\dot{\theta}) \tag{1.33}$$

$$\tau = \hat{R}^{-1}(\theta, \dot{\theta}, \ddot{\theta}^*) \tag{1.34}$$

Resolved acceleration position control is the basis for several Cartesian-based force control schemes discussed next. We have experimentally compared resolved acceleration position control to Cartesian-based PD position control, and found that the former did indeed track the trajectory more accurately.

1.5.3 Cartesian-Based Force Control

When the tip of the manipulator contacts the environment, it will be able to generate positions in certain directions and forces in other directions. Thus the geometry of the environment provides the best coordinate system to partition variables into position-controlled versus force-controlled and to plan the movement (Mason, 1981).

Just as in position control, there are issues of feedback versus feedforward control, and of the role of a model in accurate trajectory tracking. In addition, it turns out there is an issue of instability as well; that is to say, not using a dynamic model can make force control unstable as well as inaccurate. Figures 1.11 and 1.12 illustrate two alternatives of using or not using a model in force control.

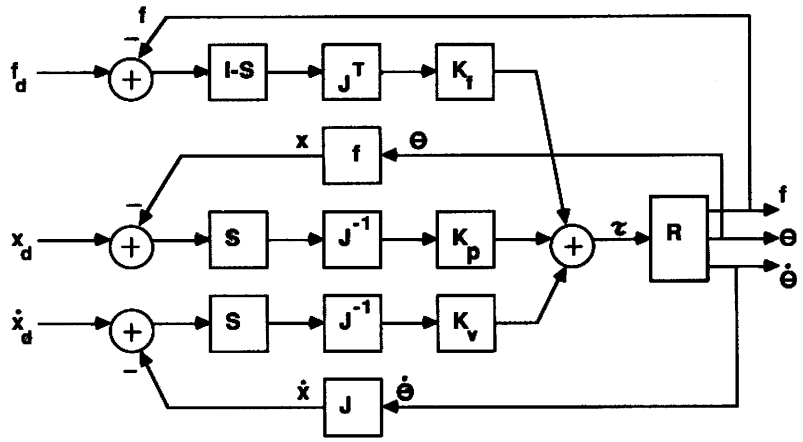

Figure 1.11: Hybrid position/force control.

The controller in Figure 1.11 is known as *hybrid position/force control* (Raibert and Craig, 1981), and does not use a model of the robot. It differs from the Cartesian-based PD position controller based on the inverse Jacobian matrix (Figure 1.9B) only by the inclusion of a force control loop.

$$\tau = \mathbf{K}_p\mathbf{J}^{-1}\mathbf{S}(\mathbf{x}_d - \mathbf{x}) + \mathbf{K}_v\mathbf{J}^{-1}\mathbf{S}(\dot{\mathbf{x}}_d - \dot{\mathbf{x}}) + \mathbf{K}_f\mathbf{J}^T(\mathbf{I} - \mathbf{S})(\mathbf{f}_d - \mathbf{f}) \quad (1.35)$$

The force error $\mathbf{f}_d - \mathbf{f}$ is transformed to joint coordinates by the Jacobian matrix $\mathbf{J}^T$ and then multiplied by the force gain $\mathbf{K}_f$. The force and position feedbacks are summed to provide torques to the robot's joints.

The external variables are presumed to have been partitioned into position-controlled $\mathbf{x}$ versus force-controlled $\mathbf{f}$, and desired trajectories $\mathbf{x}_d$ and $\mathbf{f}_d$ have been specified for each. In the figure, this partitioning is indicated by projection matrices $\mathbf{S}$ and $\mathbf{I} - \mathbf{S}$, where $\mathbf{S}$ selects the variables to be position-controlled (by diagonal elements of 0 and 1), and $\mathbf{I} - \mathbf{S}$ selects the complementary force variables ($\mathbf{I}$ is the identity matrix).

Resolved acceleration force control (Shin and Lee, 1985) is a simple extension to resolved acceleration position control (Figure 1.10), by adding a force loop (Figure 1.12):

$$\tau = \hat{R}^{-1}(\theta, \dot{\theta}, \ddot{\theta}^*) + \mathbf{J}^T(\mathbf{I} - S)\mathbf{K}_f(\mathbf{f}_d - \mathbf{f}) \quad (1.36)$$

A force error $\mathbf{f}_d - \mathbf{f}$ is multiplied first by a force gain $\mathbf{K}_f$ and then by a selection matrix $\mathbf{I} - \mathbf{S}$ to ensure the correct partitioning between position

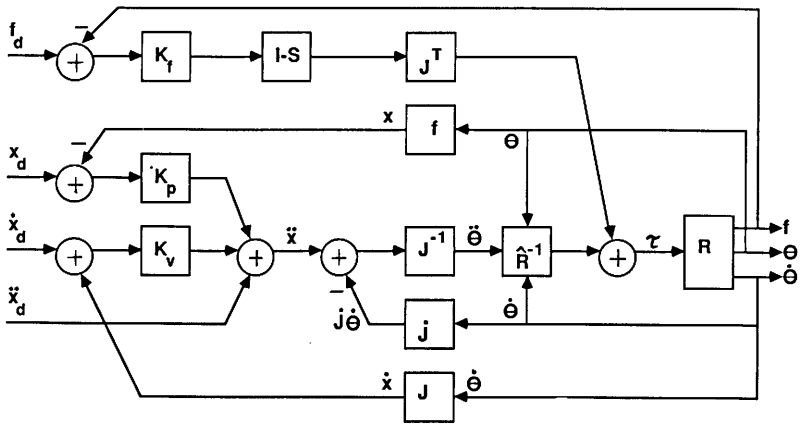

Figure 1.12: Resolved acceleration force control.

variables and force variables. The transformation to joint torques is then accomplished with the transpose Jacobian matrix $\mathbf{J}^T$; note the different order of transformations in force for hybrid position/force control (Figure 1.11). Finally, the outputs from the position and force controllers are summed to produce the joint torques.

Thus the dynamic model is applied to the position control loop, entirely analogous to computed torque control or resolved acceleration position control, and the force control loop adds in separately. Because the transpose Jacobian matrix $\mathbf{J}^T$ directly maps endpoint forces and torques into joint torques, there is no need to map the force control loop through the dynamic model. Resolved acceleration force control is virtually identical to other Cartesian-based force controllers, such as impedance control (Hogan, 1985a-c) and the operational space method (Khatib, 1987), with an advantage perhaps that it is more transparent.

Although hybrid position/force control (Figure 1.11) is quite well known, Chapter 9 presents the surprising result that this control is fundamentally unstable for revolute manipulators. This is a new form of instability that has not been recognized before, and which we call kinematic instability. This instability evidently arises from the interaction of the inverse Jacobian matrix $\mathbf{J}^{-1}$ with the selection matrix $\mathbf{S}$ and the inertia matrix $\mathbf{H}$. Kinematic instability depends on the geometric structure of the manipulator, since polar manipulators do not exhibit this form of instability. The original implementation of Raibert and Craig (1981) was on the Stanford manipulator, which is a polar manipulator, and hence

they did not observe any stability problems.

Resolved acceleration force control solves the problem, and is always stable, as long as the dynamic model is reasonably accurate. Note that if the inverse dynamics transformation is modeled as the identity matrix in Figure 1.12, then resolved acceleration force control essentially reduces to hybrid position/force control (Figure 1.11). This poignantly illustrates the importance of a dynamic model in force control as well as in position control, since a substantial modeling error (the identity matrix) makes the controller unstable.

Chapter 9 also presents experiments on resolved acceleration force control with the direct drive arm. The estimated dynamic model from Chapter 5 is used, as well as the two-part dynamically stable force control loop from Chapter 8. The results indicate fast, stable, and accurate force control.