Chapter 4

Computed-Torque Control

In this chapter we examine some straightforward control schemes for robot manipulators that fall under the class known as "computed-torque controllers." These generally perform well when the robot arm parameters are known fairly accurately. Some connections are given with classical robot control, and modern design techniques are provided as well. The effects of digital implementation of robot controllers are shown. Trajectory generation is outlined.

4.1 Introduction

A basic problem in controlling robots is to make the manipulator follow a preplanned desired trajectory. Before the robot can do any useful work, we must position it in the right place at the right instances. In this chapter we discuss *computed-torque control,* which yields a family of easy-to-understand control schemes that often work well in practice. These schemes involve the decomposition of the controls design problem into an *inner-loop design* and an *outer-loop design*.

In Section 4.4 we provide connections with classical manipulator control schemes based on independent joint design using PID control. In Section 4.6 we show how to use some modern design techniques in conjunction with computed-torque control. Thus this chapter could be considered as a bridge between classical design techniques of the sort used several years ago in robot control, and the modern design techniques in the remainder of the book which are needed to obtain high performance in uncertain environments.

We assume here the robot is moving in free space, having no contact with its environment. Contact results in the generation of forces. The *force control*

problem is dealt with in Chapter 7. We will also assume in this chapter that the robot is a well-known rigid system, thus designing controllers based on a fairly well-known model. Control in the presence of uncertainties or unknown parameters (e.g., friction, payload mass) requires refined approaches. This problem is dealt with using *robust control* in Chapter 4 and *adaptive control* in Chapter 5.

An actual robot manipulator may have flexibility in its links, or compliance in its gearing (joint flexibility). In Chapter 6 we cover some aspects of control with joint flexibility.

Before we can control a robot arm, it is necessary to know the *desired path* for performing a task. There are many issues associated with the path planning problem, such as avoiding obstacles and making sure that the planned path does not require exceeding the voltage and torque limitations of the actuators. To reduce the control problem to its basic components, in this chapter we assume that the ultimate control objective is to move the robot along a prescribed desired trajectory. We do not concern ourselves with the actual trajectory-planning problem; we do, however, show how to reconstruct a continuous desired path from a given table of desired points the end effector should pass through. This *continuous-path generation* problem is covered in Section 4.2.

In most practical situations robot controllers are implemented on microprocessors, particularly in view of the complex nature of modern control schemes. Therefore, in Section 4.5 we illustrate some notions of the *digital implementation of robot controllers*.

Throughout, we demonstrate how to simulate robot controllers on a computer. This should be done to verify the effectiveness of any proposed control scheme prior to actual implementation on a real robot manipulator.

4.2 Path Generation

Throughout the book we assume that there is given a prescribed path $q_d(t)$ the robot arm should follow. We design control schemes that make the manipulator follow this desired path or trajectory. *Trajectory planning* involves finding the prescribed path and is usually considered a separate design problem involving collision avoidance, concerns about *actuator saturation,* and so on. See [Lee et al. 1983].

We do not cover trajectory planning. However, we do cover two aspects of trajectory generation. First, we show how to convert a given prescribed path from Cartesian space to joint space. Then, given a table of desired points the end effector should pass through, we show how to reconstruct a continuous desired trajectory.

Converting Cartesian Trajectories to Joint Space

In robotic applications, a desired task is usually specified in the workspace or Cartesian space, as this is where the motion of the manipulator is easily described in relation to the external environment and workpiece. However, trajectory-following control is easily performed in the joint space, as this is where the arm dynamics are more easily formulated.

Therefore, it is important to be able to find the desired joint space trajectory $q_d(t)$ given the desired Cartesian trajectory. This is accomplished using the *inverse kinematics*, as shown in the next example. The example illustrates that the mapping of Cartesian to joint space trajectories may not be unique-that is, several joint space trajectories may yield the same Cartesian trajectory for the end-effector.

EXAMPLE 4.2–1: Mapping a Prescribed Cartesian Trajectory to Joint Space

In Example A.3–5 are derived the inverse kinematics for the two-link planar robot arm shown in Figure 4.2.1. Let us use them to convert a path from Cartesian space to joint space.

Suppose that we want the two-link arm to follow a given workspace or Cartesian trajectory

$$p(t)=(x(t), y(t)) \tag{1}$$

in the (x, y) plane which is a function of time t. Since the arm is moved by actuators that control its angles θ_1, θ_2, it is convenient to convert the specified Cartesian trajectory $(x(t), y(t))$ into *a joint space trajectory* $(\theta_1(t), \theta_2(t))$ for control purposes.

This may be achieved by using the inverse kinematics transformations

$$r^2=x^2+y^2 \tag{2}$$

$$C = \cos\theta_2 = \frac{r^2 - a_1^2 + a_2^2}{2a_1a_2} \tag{3}$$

$$D = \pm\sqrt{1 - \cos^2\theta_2} = \pm\sqrt{1 - C^2} \tag{4}$$

$$\theta_2=\text{ATAN2}\ (D, C) \tag{5}$$

$$\theta_1=\text{ATAN2}(y, x)-\text{ATAN2}\,(a_2 \sin\theta_2,\ a_1+a_2\cos\theta_2) \tag{6}$$

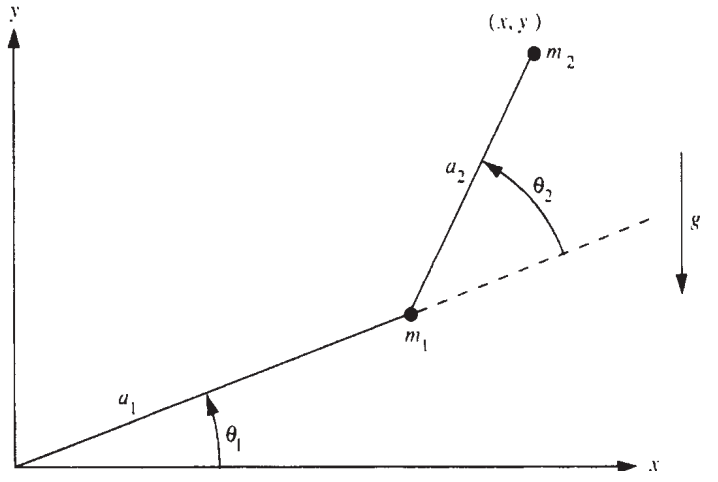


Figure 4.2.1: Two-link planar elbow arm.

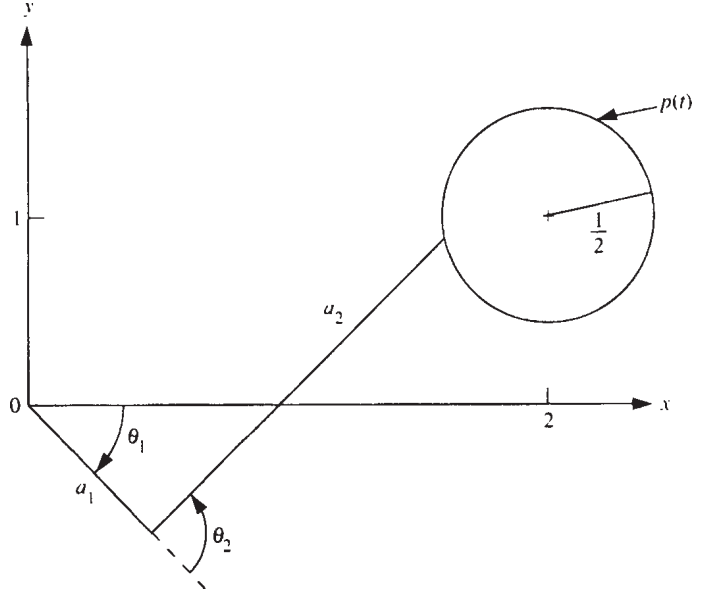


Figure 4.2.2: Desired Cartesian trajectory.

Suppose that the end of the arm should repeatedly trace out the circular workspace path $p(t)$ shown in Figure 4.2.2, which is described by

$$x(t)=2+\tfrac{1}{2}\cos t$$
$$y(t)=1+\tfrac{1}{2}\sin t. \quad (7)$$

By using these expressions for each time t in the inverse kinematics equations, we obtain the required joint-space trajectories $q(t)=(\theta_1(t), \theta_2(t))$ given in Figure 4.2.3 that yield the circular Cartesian motion of the end effector (using $a_1=2$, $a_2=2$).

We have computed the joint variables for the "elbow down" configuration. Selecting the opposite sign in (4) gives the "elbow up" joint space trajectory yielding the same Cartesian trajectory.

■

Polynomial Path Interpolation

Suppose that a desired trajectory for the manipulator motion has been determined, either in Cartesian space or, using the inverse kinematics, in joint space. For convenience, we use the joint space variable $q(t)$ for notation. It is not possible to store the entire trajectory in computer memory, and few practically useful trajectories have a simple closed-form expression. Therefore, it is usual to store in computer memory a sequence of points $q_i(t_k)$ for each joint variable i that represent the desired values of that variable at the discrete times t_k. Thus $q(t_k)$ is a point in R^n that the joint variables should pass through at time t_k.We call these *via points*.

Most robot control schemes require a continuous desired trajectory. To convert the table of via points $q_i(t_k)$ to a continuous desired trajectory $q_d(t)$, we may use many options. Let us discuss here *polynomial interpolation*.

Suppose that the via points are uniformly spaced in time and define the *sampling period* as

$$T=t_{k+1}-t_k. \quad (4.2.1)$$

For smooth motion, on each time interval $[t_k, t_{k+1}]$ we require the desired position $q_d(t)$ and velocity $\dot{q}_d(t)$ to match the tabulated via points. This yields boundary conditions of

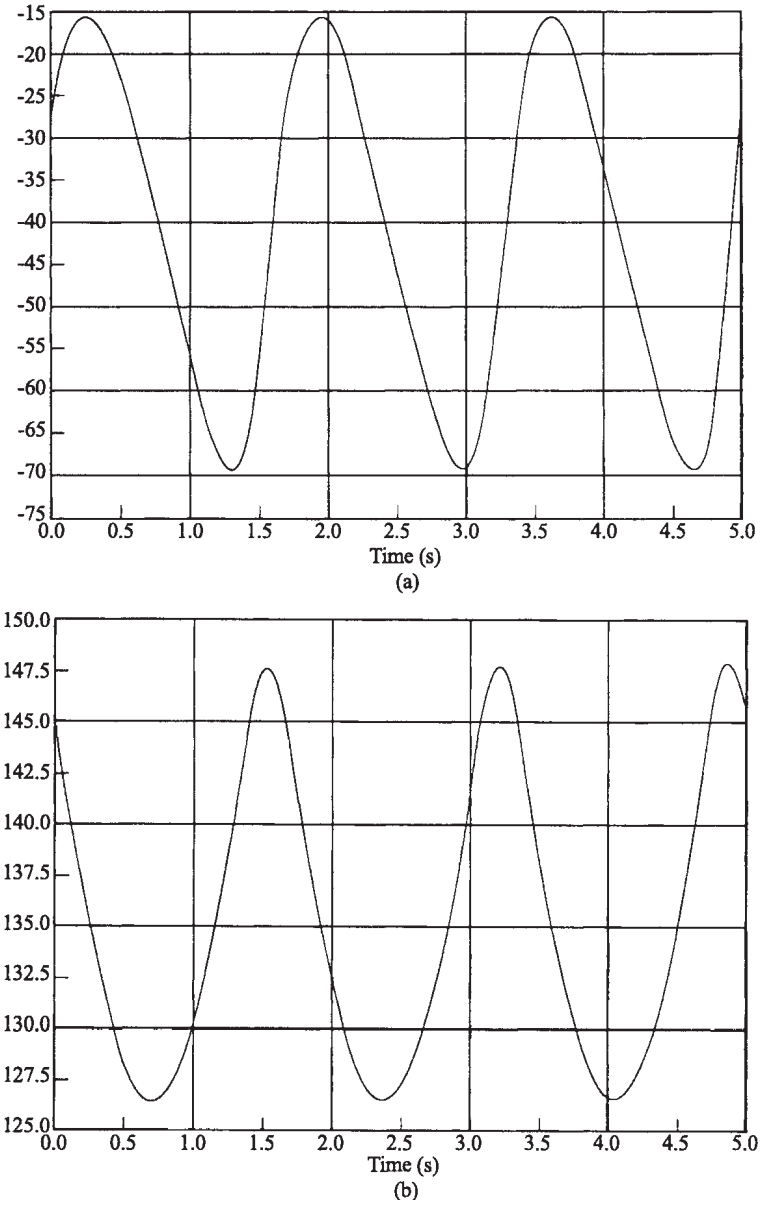


Figure 4.2.3: Required joint-space trajectories: (a) θ_1 (deg); (b) θ_2 (deg).

$$\begin{aligned} q_{d_i}(t_k) &= q_i(t_k) \\ \dot{q}_{d_i}(t_k) &= \dot{q}_i(t_k) \\ q_{d_i}(t_{k+1}) &= q_i(t_{k+1}) \\ \dot{q}_{d_i}(t_{k+1}) &= \dot{q}_i(t_{k+1}). \end{aligned} \tag{4.2.2}$$

To match these boundary conditions, it is necessary to use on $[tk, t_{k+1}]$ the *cubic interpolating polynomial*

$$q_{d_i}(t) = a_i + (t - t_k) b_i + (t - t_k)^2 c_i + (t - t_k)^3 d_i, \tag{4.2.3}$$

which has four free variables. Then

$$\dot{q}_{d_i}(t) = b_i + 2(t - t_k) c_i + 3(t - t_k)^2 d_i \tag{4.2.4}$$

$$\ddot{q}_{d_i}(t) = 2c_i + 6(t - t_k) d_i \tag{4.2.5}$$

so that the acceleration is *linear* on each sample period.

It is easy to solve for the coefficients that guarantee matching of the boundary conditions. In fact, we see that

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & T & T^2 & T^3 \\ 0 & 1 & 2T & 3T^2 \end{bmatrix} \begin{bmatrix} a_i \\ b_i \\ c_i \\ d_i \end{bmatrix} = \begin{bmatrix} q_i(t_k) \\ \dot{q}_i(t_k) \\ q_i(t_{k+1}) \\ \dot{q}_i(t_{k+1}) \end{bmatrix} \tag{4.2.6}$$

This is solved to obtain the required interpolating coefficients on each interval $[t_k, t_{k+1}]$.

$$\begin{aligned} a_i &= q_i(t_k) \\ b_i &= \dot{q}_i(t_k) \\ c_i &= \frac{3[q_i(t_{k+1}) - q_i(t_k)] - T[2\dot{q}_i(t_k) + \dot{q}_i(t_{k+1})]}{T^2} \\ d_i &= \frac{2[q_i(t_k) - q_i(t_{k+1})] - T[\dot{q}_i(t_k) + \dot{q}_i(t_{k+1})]}{T^3}. \end{aligned} \tag{4.2.7}$$

Note that this technique requires storing the desired position *and velocity* at each sampling point in tabular form. A variant uses a higher-order polynomial to ensure continuous position, velocity, *and acceleration* at each sample time t_k.

Although we have used the joint variable notation $q(t)$, it should be emphasized that trajectory interpolation can also be performed in Cartesian space.

Linear Function with Parabolic Blends

Using cubic interpolating polynomials, the acceleration on each sample period is linear. However, in many practical applications there are good reasons for insisting on *constant* accelerations within each sample period. For instance, any real robot has upper limits on the torques that can be supplied by its actuators. For linear systems (think of Newton's law) this translates into constant accelerations. Therefore, constant accelerations are less likely to saturate the actuators. Besides that, most industrial robot controllers are programmed to use constant accelerations on each sample period.

A constant acceleration profile is shown in Figure 4.2.4(a). The associated velocity and position profiles are shown in Figure 4.2.4(b) and 4.2.4(c). The position trajectory has three parts: a quadratic or parabolic initial portion, a linear midsection, and a parabolic final portion. Therefore, let us discuss interpolation of via points using *linear functions with parabolic blends* (LFPB).

The time at which the position trajectory switches from parabolic to linear is known as the *blend time* t_b. A position $q_{di}(t)$ *should be specified for each joint variable* i. The trajectory in Figure 4.2.4(c) can be written for joint i as

$$q_{d_i}(t) = \begin{cases} a_i + (t - t_k)\, b_i + (t - t_k)^2 c_i, & t_k \le t < t_k + t_b \\ d_i + v_i t, & t_k + t_b \le t < t_{k+1} - t_b \\ e_i + (t - t_{k+1})\, f_i + (t - t_{k+1})^2 g_i, & t_{k+1} - t_b \le t \le t_{k+1}. \end{cases} \tag{4.2.8}$$

The coefficient v_i may be interpreted as the maximum velocity allowed for joint variable i. The design parameters are v_i and t_b.

It is straightforward to solve for the coefficients on each time interval $[t_k, t_{k+1}]$ that ensure satisfaction of the boundary conditions (4.2.2). The result is

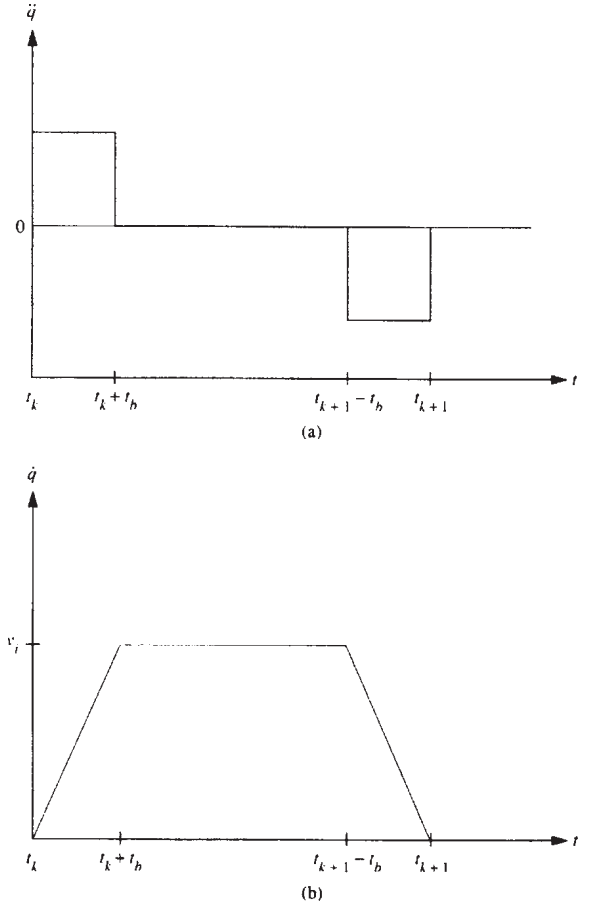


Figure 4.2.4: LFPB trajectory: (a) acceleration; (b) velocity.

$$
\begin{aligned}
a_i &= q_i(t_k), \quad b_i = \dot{q}_i(t_k), \quad c_i = \frac{v_i - \dot{q}_i(t_k)}{2t_b}, \\
d_i &= \frac{q_i(t_k) + q_i(t_{k+1}) - v_i t_{k+1}}{2} \\
e_i &= q_i(t_{k+1}), \quad f_i = \dot{q}_i(t_{k+1}) \\
g_i &= \frac{v_i t_{k+1} + q_i(t_k) - q_i(t_{k+1}) + 2t_b[\dot{q}_i(t_{k+1}) - v_i]}{2t_b^2}.
\end{aligned}
\tag{4.2.9}
$$

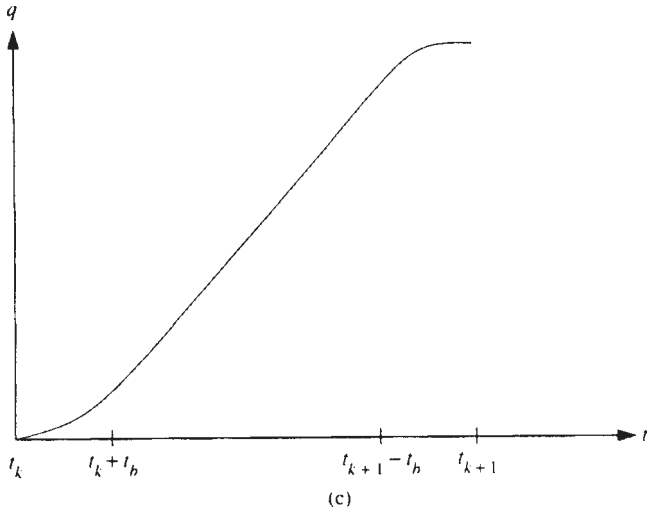


Figure 4.2.4: (*Cont.*)(c) position.

Minimum-Time Trajectories

There is an important special class of LFPB trajectories. Suppose the acceleration is limited by a maximum value of a_M and it is desired for the robot arm to get from one position to another in *minimum time*. For simplicity assume that the initial and final velocities are equal to zero. The general case is covered in [Lewis 1986a] (see the Problems).

A minimum-time trajectory is shown in Figure 4.2.5. To drive joint variable i from a rest position of $q_0=q_i(t_0)$ to a desired final rest position of $q_f=q_i(t_f)$ in a minimum time t_f, the maximum acceleration a_M should be applied until the *switching time* t_s, after which time the maximum deceleration $-a_M$ should be applied until t_f. Note that both t_s and t_f depend on q_0 and q_f. We may write

$$
\begin{aligned}
q_i(t_s) &= q_0 + \tfrac{1}{2}a_M(t_s - t_0)^2 \\
\dot{q}_i(t_s) &= a_M(t_s - t_0) \\
q_i(t_f) &= q_i(t_s) + \dot{q}_i(t_s)(t_f - t_s) - \tfrac{1}{2}a_M(t_f - t_s)^2 \\
\dot{q}_i(t_f) &= \dot{q}_i(t_s) - a_M(t_f - t_s).
\end{aligned}
$$

Then the velocity equations yield

$$\dot{q}_i(t_f) = a_M(t_s - t_0) - a_M(t_f - t_s) = 0$$

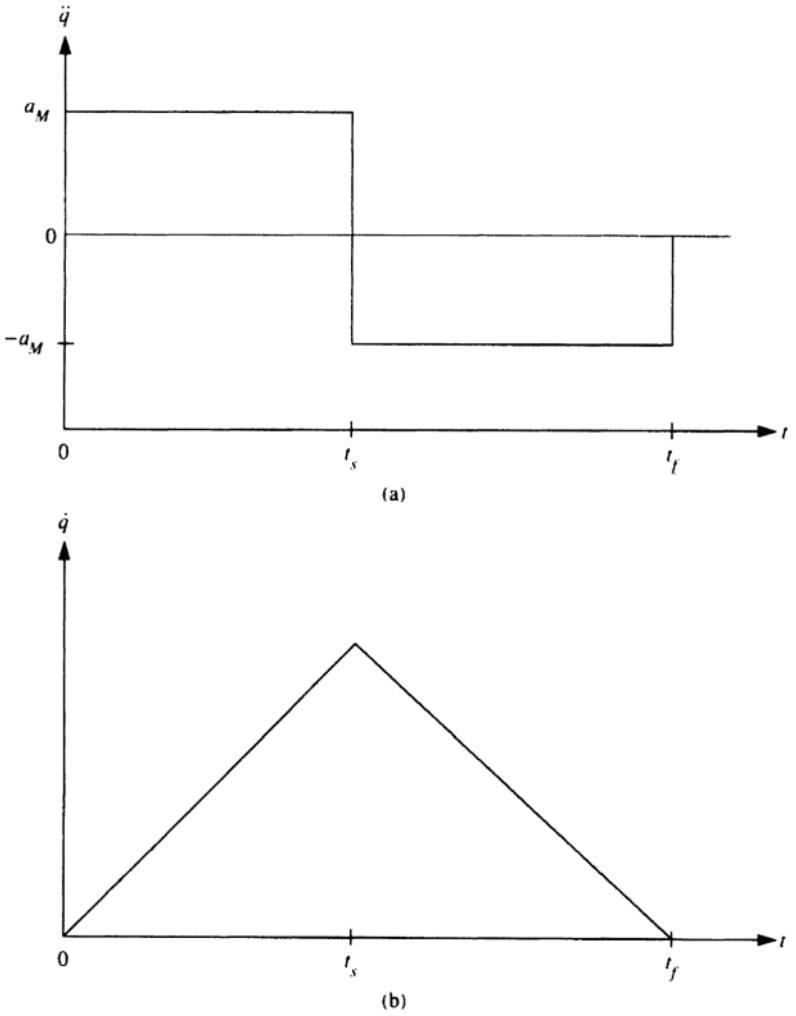


Figure 4.2.5: Minimum-time trajectory: (a) acceleration; (b) velocity.

or

$$t_s=(t_f+t_0)/2. \tag{4.2.10}$$

That is, the switching from maximum acceleration to maximum deceleration occurs at the *half-time point*. Now simple manipulations on the position equations yield

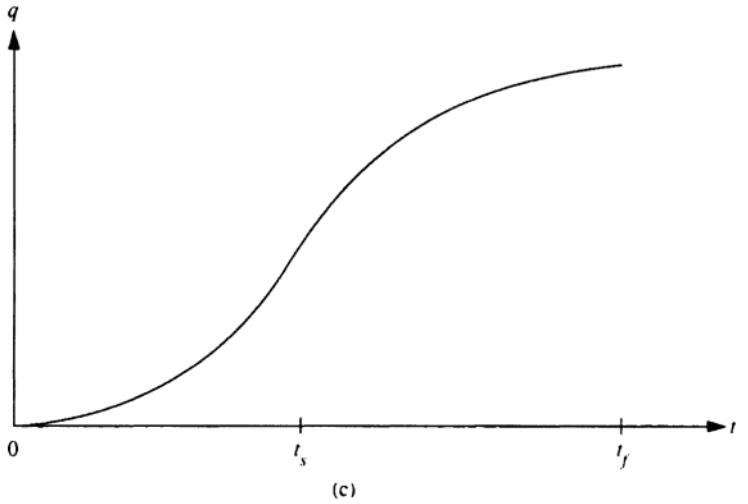


Figure 4.2.5: (*Cont.*)(c) position.

$$q_i(t_f) = q_0 + \tfrac{1}{2}a_M(t_s - t_0)^2 + a_M(t_s - t_0)(t_f - t_s) - \tfrac{1}{2}a_M(t_f - t_s)^2$$
$$= q_f$$
$$\frac{q_f - q_0}{a_M} = \tfrac{1}{2}(t_s - t_0)^2 + (t_s - t_0)(t_f - t_s) - \tfrac{1}{2}(t_f - t_s)^2$$

whence (4.2.10) yields

$$t_s = t_0 + \sqrt{(q_f - q_0)/a_M}. \tag{4.2.11}$$

A closed-loop formulation of minimum-time control appears in [Lewis 1986a].

Unfortunately, minimum-time trajectories computed using a constant maximum acceleration are not directly relevant in robotics. This is due to the fact that an actual manipulator has a *torque limit* of τ_m. Since the robot equation (see Table 3.3.1) is nonlinear, this does not correlate to a constant upper bound on the acceleration. For instance, a robot arm has different maximum accelerations in its fully extended and fully retracted positions. For more discussion see [Kahn and Roth 1971], [Chen 1989], [Geering 1986], [Gourdeau and Schwartz 1989], [Jayasuriya and Suh 1985], [Kahn and Roth 1971], [Kim and Shin 1985], [Shin and McKay 1985].

4.3 Computer Simulation of Robotic Systems

It is very important to *simulate* on a digital computer a proposed manipulator control scheme before actually implementing it on an arm. We show here how to perform such computer simulations for robotic systems. Since most robot controllers are actually implemented in a digital fashion (Section 4.5), we also show how to simulate digital robot arm controllers.

Simulation of Robot Dynamics

There is a variety of software packages for the simulation of nonlinear dynamical systems, including SIMNON [Åström and Wittenmark 1984], MATLAB, and others. For convenience, we include in Appendix B some simulation programs that are quite useful for continuous and digital control.

All simulation programs require the user to write similar subroutines. Time response simulators that use integration routines such as Runge-Kutta all require the computation of the state derivative given the current state. In Section 3.4 we saw how to represent the robot arm equation

$$M(q)\ddot{q} + N(q,\dot{q}) + \tau_d = \tau \tag{4.3.1}$$

in the nonlinear state-space form

$$\dot{x} = f(x,u,t), \tag{4.3.2}$$

with $x(t)$ the state and $u(t)$ the input.

Defining a state as $x=[q^T\ \dot{q}^T]^T$, we may write the *implicit form*

$$\begin{bmatrix} I & 0 \\ 0 & M(q) \end{bmatrix} \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} \dot{q} \\ -N(q,\dot{q}) \end{bmatrix} + \begin{bmatrix} 0 \\ I \end{bmatrix} \tau + \begin{bmatrix} 0 \\ -I \end{bmatrix} \tau_d, \tag{4.3.3}$$

with τ the arm control torque that is provided by the controller and τ_d the disturbance torque. We say that this is an implicit form since the coefficient matrix of the left-hand side means that $\dot{x} = d[q^T \quad \dot{q}^T]^T/dt$ is not given explicitly in terms of the right-hand side.

Given $x(t)$, it is necessary to provide a subroutine for the integration program that computes $\dot{x}(t)$. One approach to solving for $\dot{x}$ is to invert $M(q)$. However, due to potential numerical problems this is not recommended. Let us represent (4.3.3) as

$$E(x)\dot{x} = f(x,u,t). \tag{4.3.4}$$

Note that in this case $u(t)$ is the vector composed of the controls $\tau(t)$ and the disturbances $\tau_d(t)$.

A simple time response program, TRESP, is given in Appendix B. Given a subroutine *F(time, x,* $\dot{\boldsymbol{x}}$) that computes $\dot{\boldsymbol{x}}$ given *x(t)* and *u(t)* using (4.3.4); it uses a Runge-Kutta integrator to compute the state trajectory *x(t)*. To solve for $\dot{\boldsymbol{x}}$ within subroutine *F(t, x,* $\dot{\boldsymbol{x}}$)we recommend computing *M(q)* and *N(q, q̇)* and, then solving

$$M(q)\frac{d\dot{q}}{dt} = -N(q,\dot{q}) + \tau - \tau_d \tag{4.3.5}$$

[i.e., the bottom portion of (4.3.3)] by *least-squares techniques,* which are more stable numerically than the inversion of *M(q)*. Least-squares equation solvers are readily available commercially in, for instance [IMSL], [LINPACK], and elsewhere. For simpler arms, *M(q)* may be inverted analytically.

Throughout the book we illustrate the simulation of the arm dynamics using various control schemes.

Simulation of Digital Robot Controllers

While most robot controllers are designed in continuous time, they are implemented on actual robots digitally. That is, the control signals are only updated at discrete instants of time using a microprocessor. We discuss the implementation of digital robot arm controllers in Section 4.5. To verify that a proposed controller will operate as expected, therefore, it is highly desirable to simulate it in its digitized or discretized form prior to actual implementation.

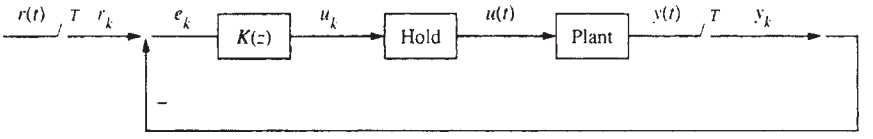


Figure 4.3.1: Digital controller.

A digital control scheme is shown in Figure 4.3.1. The *plant* or system to be controlled is a continuous-time system, and *K(z)* is the dynamic digital controller, where *z* is the Z-transform variable (i.e., z^{-1} represents a unit time delay). The digital controller *K(z)* is implemented using software code in a digital signal processor (DSP). The *reference input r(t)* is the desired trajectory that *y(t)* should follow, and e_k is the (discrete) tracking error.

The sampler with sample period *T* is an analog-to-digital (A/D) converter that takes the samples *yk=y(kT)* of the output *y(t)* that are required by the software controller *K(z)*. The definition of *y(t)* can vary depending on the

control scheme. For instance, in robot control, $y(t)$ might represent the $2n$—vector composed of $q(t)$ and $\dot{q}(t)$.

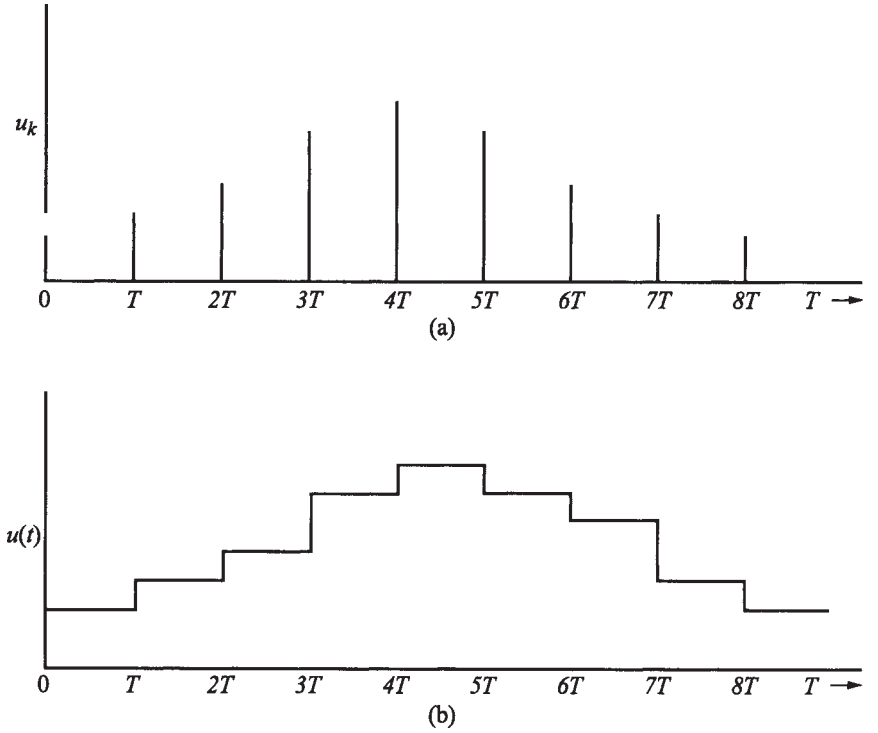


Figure 4.3.2: Data reconstruction using a ZOH: (a) discrete control sequence u_k; (b) reconstructed continuous signal $u(t)$.

The hold device in the figure is a digital-to-analog (D/A) converter that converts the discrete control samples u_k computed by the software controller $K(z)$ into the continuous-time control $u(t)$ required by the plant. It is a *data reconstruction* device. The input u_k and output $u(t)$ for a zero-order hold (ZOH) are shown in Figure 4.3.2. Note that $u(kT)=u_k$, with T the sample period, so that $u(t)$ is continuous from the right. That is, $u(t)$ is updated at times kT. The ZOH is generally used for controls purposes, as opposed to other higher-order devices such as the first-order hold, since most commercially available DSPs have a built-in ZOH.

Once a controller has been designed, it is important to *simulate* it using a digital computer before it is implemented to determine if the closed-loop response is suitable. This is especially true in robotics, since the digital

controller is generally found by designing a continuous-time controller, which is then digitized using approximation techniques such as Euler's method. That is, for nonlinear systems, the controller discretization schemes are generally not exact. This results in degraded performance. To verify that the controller performance will be suitable, the simulation should provide the response at all times, including times *between* the samples.

To simulate a digital controller we may use the scheme shown in Figure 4.3.3. There, the continuous plant dynamics are contained in the subroutine $F(t, x, \dot{x})$; they are integrated using a Runge-Kutta integrator. The figure assumes a ZOH; thus the control input $u(t)$ is updated to u_k at each time kT, and then held constant until time $(k+1)T$ Note that two time intervals are involved; the sampling period T and the *Runge-Kutta integration period* $T_R \ll T$. T_R should be selected as an integral divisor of T.

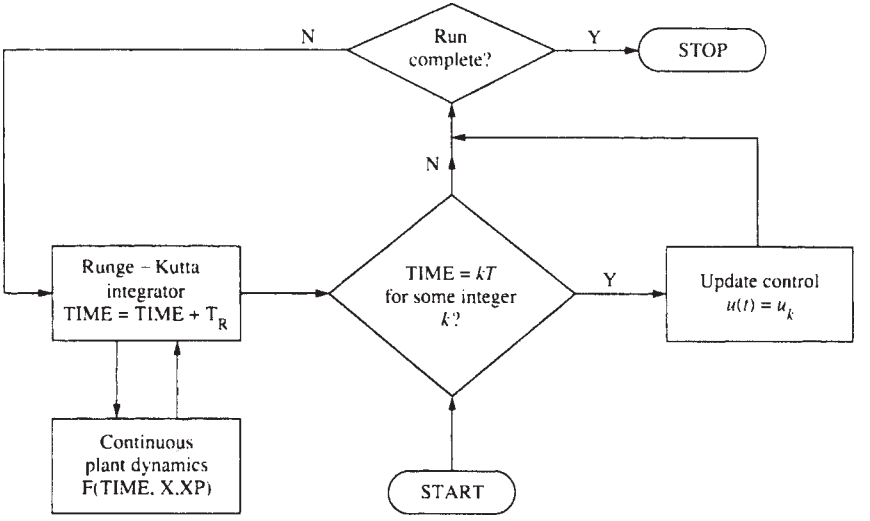


Figure 4.3.3: Digital control simulation scheme.

This simulation technique provides the plant state $x(t)$ as a continuous function of time, even at values *between* the sampling instants [in fact, it provides $x(t)$ at multiples of T_R]. This is essential in verifying acceptable *intersample behavior* of the closed-loop system prior to implementing the digital controller on the actual plant.

Program TRESP in Appendix B can be used to implement Figure 4.3.3. It is written in a modular fashion to apply to a wide variety of situations. We shall illustrate its use for the purpose of digital control in several

subsequent examples. The use of such simulation software as SIMNON is quite similar.

We discuss the implementation of digital robot arm controllers in Section 4.5. Some detailed discussion on digital control, simulation, and DSP implementation of controllers is given in [Lewis 1992].

4.4 Computed-Torque Control

Through the years there have been proposed many sorts of robot control schemes. As it happens, most of them can be considered as special cases of the class of *computed-torque controllers*. Computed torque, at the same time, is a special application of *feedback linearization* of nonlinear systems, which has gained popularity in modern systems theory [Hunt et al. 1983], [Gilbert and Ha 1984]. In fact, one way to classify robot control schemes is to divide them as "computed-torque-like" or "noncomputed-torque-like." Computed-torque-like controls appear in robust control, adaptive control, learning control, and so on.

In the remainder of this chapter we explore this class of robot controllers, which includes such a broad range of designs. Computed-torque control allows us to conveniently derive very effective robot controllers, while providing a framework to bring together classical independent joint control and some modern design techniques, as well as set the stage for the rest of the book. A summary of the different computed-torque-like controllers is given at the end of the section in Table 4.4.1. We shall see that many digital robot controllers are also computed-torque-like controllers (Section 4.5).

Derivation of Inner Feedforward Loop

The robot arm dynamics are

$$M(q)\ddot{q} + V(q,\dot{q}) + F_v\dot{q} + F_d(\dot{q}) + G(q) + \tau_d = \tau \tag{4.4.1}$$

or

$$M(q)\ddot{q} + N(q,\dot{q}) + \tau_d = \tau, \tag{4.4.2}$$

with the joint variable $q(t) \in \mathbf{R}^n$, $\tau(t)$ the control torque, and $\tau_d(t)$ a disturbance. If this equation includes motor actuator dynamics (Section 3.6), then $\tau(t)$ is an input voltage.

Suppose that a desired trajectory $q_d(t)$ has been selected for the arm motion, according to the discussion in Section 4.2. To ensure trajectory tracking by the joint variable, define an output or *tracking error* as

$$e(t) = q_d(t) - q(t). \tag{4.4.3}$$

To demonstrate the influence of the input $\tau(t)$ on the tracking error, differentiate twice to obtain

$$\dot{e} = \dot{q}_d - \dot{q}$$
$$\ddot{e} = \ddot{q}_d - \ddot{q}.$$

Solving now for $\ddot{q}$ in (4.4.2) and substituting into the last equation yields

$$\ddot{e} = \ddot{q}_d + M^{-1}(N + \tau_d - \tau). \tag{4.4.4}$$

Defining the control input function

$$u = \ddot{q}_d + M^{-1}(N - \tau) \tag{4.4.5}$$

and the disturbance function

$$w = M^{-1}\tau_d \tag{4.4.6}$$

we may define a state $x(t) \in \mathbf{R}^{2n}$ by

$$x = \begin{bmatrix} e \\ \dot{e} \end{bmatrix} \tag{4.4.7}$$

and write the *tracking error dynamics* as

$$\frac{d}{dt}\begin{bmatrix} e \\ \dot{e} \end{bmatrix} = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}\begin{bmatrix} e \\ \dot{e} \end{bmatrix} + \begin{bmatrix} 0 \\ I \end{bmatrix} u + \begin{bmatrix} 0 \\ I \end{bmatrix} w. \tag{4.4.8}$$

This is a linear error system in Brunovsky canonical form consisting of n pairs of double integrators $1/s^2$, one per joint. It is driven by the control input $u(t)$ and the disturbance $w(t)$. Note that this derivation is a special case of the general feedback linearization procedure in Section 3.4.

The feedback linearizing transformation (4.4.5) may be inverted to yield

$$\tau = M(\ddot{q}_d - u) + N. \tag{4.4.9}$$

We call this the *computed-torque control law*. The importance of these manipulations is as follows. There has been no state-space transformation in going from (4.4.1) to (4.4.8). Therefore, if we select a control $u(t)$ that stabilizes (4.4.8) so that $e(t)$ goes to zero, then the nonlinear control input given by $\tau(t)$(4.4.9) will cause trajectory following in the robot arm (4.4.1). In fact, substituting (4.4.9) into (4.4.2) yields

$$M\ddot{q} + N + \tau_d = M(\ddot{q}_d - u) + N$$

or

$$\ddot{e} = u + M^{-1}\tau_d, \tag{4.4.10}$$

which is exactly (4.4.8).

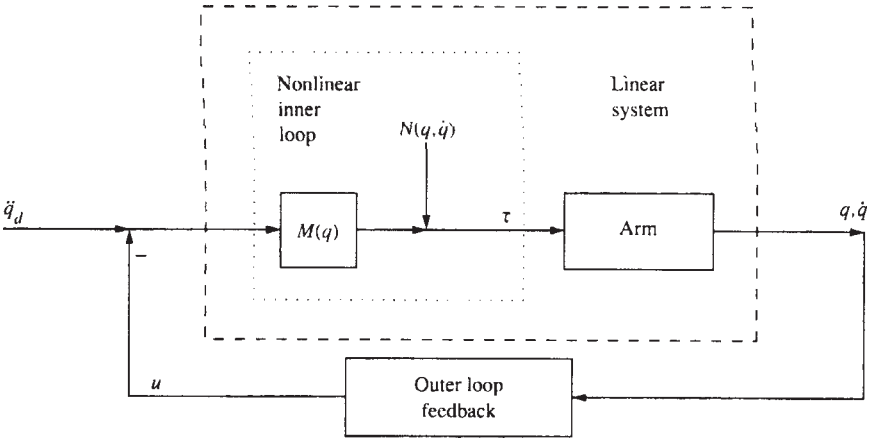


Figure 4.4.1: Computed-torque control scheme, showing inner and outer loops.

The stabilization of (4.4.8) is not difficult. In fact, the nonlinear transformation (4.4.5) has converted a complicated nonlinear controls design problem into a simple design problem for a linear system consisting of n decoupled subsystems, each obeying Newton's laws.

The resulting control scheme appears in Figure 4.4.1. It is important to note that it consists of an *inner nonlinear loop* plus an *outer control signal u(t)*. We shall see several ways for selecting *u(t)*. Since *u(t)* will depend on *q(t)* and $\dot{q}(t)$, the outer loop will be a feedback loop. In general, we may select a dynamic compensator *H(s)* so that

$$U(s)=H(s)E(s). \tag{4.4.11}$$

H(s) can be selected for good closed-loop behavior. According to (4.4.10), the closed-loop error system then has transfer function

$$T(s)=s^2I-H(s). \tag{4.4.12}$$

It is important to realize that computed-torque depends on the inversion of the robot dynamics, and indeed is sometimes called *inverse dynamics control* In fact, (4.4.9) shows that $\tau(t)$ is computed by substituting $\ddot{\boldsymbol{q}}_d$–u for $\ddot{\boldsymbol{q}}$ in (4.4.2); that is, by solving the robot *inverse dynamics problem*. The caveats associated with system inversion, including the problems resulting when the system has non-minimum-phase zeros, all apply here. (Note that in the linear case, the system zeros are the poles of the inverse. Such nonminimum-phase notions generalize to nonlinear systems.) Fortunately for us, the rigid arm dynamics are minimum phase.

There are several ways to compute (4.4.9) for implementation purposes. Formal matrix multiplication at each sample time should be avoided. In some cases the expression may be worked out analytically. A good way to compute the torque $\tau(t)$ is to use the efficient Newton-Euler inverse dynamics formulation [Craig 1985] with $\ddot{\boldsymbol{q}}_d$–u in place of $\ddot{\boldsymbol{q}}(t)$.

The outer-loop signal *u(t)* can be chosen using many approaches, including robust and adaptive control techniques. In the remainder of this chapter we explore some choices for *u(t)* and some variations on computed-torque control.

PD Outer-Loop Design

One way to select the auxiliary control signal *u(t)* is as the proportional-plus-derivative (PD) feedback,

$$u = -K_v\dot{e} - K_p e. \tag{4.4.13}$$

Then the overall robot arm input becomes

$$\tau = M(q)(\ddot{q}_d + K_v\dot{e} + K_p e) + N(q, \dot{q}) \tag{4.4.14}$$

This controller is shown in Figure 4.4.6 with K_i=0. The closed-loop error dynamics are

$$\ddot{e} + K_v\dot{e} + K_p e = w, \tag{4.4.15}$$

or in state-space form,

$$\frac{d}{dt}\begin{bmatrix} e \\ \dot{e} \end{bmatrix} = \begin{bmatrix} 0 & I \\ -K_p & -K_v \end{bmatrix}\begin{bmatrix} e \\ \dot{e} \end{bmatrix} + \begin{bmatrix} 0 \\ I \end{bmatrix} w. \tag{4.4.16}$$

The closed-loop characteristic polynomial is

$$\Delta_c(s) = \left| s^2 I + K_v s + K_p \right|. \tag{4.4.17}$$

Choice of PD Gains. It is usual to take the *n*×*n* gain matrices diagonal so that

$$K_v = diag\{k_{v_i}\}, \quad K_p = diag\{k_{p_i}\}. \tag{4.4.18}$$

Then

$$\Delta_c(s) = \prod_{i=1}^{n} (s^2 + k_{v_i} s + k_{p_i}), \tag{4.4.19}$$

and the error system is asymptotically stable as long as the K_{vi} and K_{pi} are all positive. Therefore, as long as the disturbance $w(t)$ is bounded, so is the error $e(t)$. In connection with this, examine (4.4.6) and recall from Table 3.3.1 that M^{-1} is upper bounded. Thus boundedness of $w(t)$ is equivalent to boundedness of $\tau_d(t)$.

It is important to note that although selecting the PD gain matrices diagonal results in decoupled control at the outer-loop level, it does not result in a decoupled joint-control strategy. This is because multiplication by $M(q)$ and addition of the nonlinear feedforward terms $N(q, \dot{q})$ in the inner loop scrambles the signal $u(t)$ among all the joints. Thus, information on all joint positions $q(t)$ and velocities $\dot{q}(t)$ is generally needed to compute the control $\tau(t)$ for any one given joint.

The standard form for the second-order characteristic polynomial is

$$p(s) = s^2 + 2\xi\omega_n s + \omega_n^2, \tag{4.4.20}$$

with ζ the damping ratio and ω_n the natural frequency. Therefore, desired performance in each component of the error $e(t)$ may be achieved by selecting the PD gains as

$$k_{p_i} = \omega_n^2, \quad k_{v_i} = 2\xi\omega_n, \tag{4.4.21}$$

with ζ, ω_n the desired damping ratio and natural frequency for joint error i. It may be useful to select the desired responses at the end of the arm faster than near the base, where the masses that must be moved are heavier.

It is undesirable for the robot to exhibit overshoot, since this could cause impact if, for instance, a desired trajectory terminates at the surface of a workpiece. Therefore, the PD gains are usually selected for *critical damping* ζ=1. In this case

$$k_{v_i} = 2\sqrt{k_{p_i}}, \quad k_{p_i} = k_{v_i}^2/4. \tag{4.4.22}$$

Selection of the Natural Frequency. The natural frequency ω_n governs the speed of response in each error component. It should be large for fast responses and is selected depending on the performance objectives. Thus the desired trajectories should be taken into account in selecting ω_n. We discuss now some additional factors in this choice.

There are some *upper limits* on the choice for ω_n [Paul 1981]. Although the links of most industrial robots are massive, they may have some flexibility. Suppose that the frequency of the first flexible or resonant mode of link i is

$$\omega_r = \sqrt{k_r/J} \tag{4.4.23}$$

with J the link inertia and k_r the link stiffness. Then, to avoid exciting the resonant mode, we should select $\omega_n < \omega_r/2$. Of course, the link inertia J changes with the arm configuration, so that its maximum value might be used in computing ω_r.

Another upper bound on ω_n is provided by considerations on actuator saturation. If the PD gains are too large, the torque $\tau(t)$ may reach its upper limits.

Some more feeling for the choice of the PD gains is provided from error-boundedness considerations as follows. The transfer function of the closed-loop error system in (4.4.15) is

$$e(s) = (s^2 I + k_v s + k_p)^{-1} w(s), \tag{4.4.24}$$

or if K_v and K_p are diagonal,

$$e_i(s) = \frac{1}{s^2 + k_{v_i} s + k_{p_i}} w(s) \equiv H(s) w(s) \tag{4.4.25}$$

$$\dot{e}_i(s) = \frac{s}{s^2 + k_{v_i} s + k_{p_i}} w(s) \equiv sH(s) w(s). \tag{4.4.26}$$

We assume that the disturbance and M^{-1} are bounded (Table 3.3.1), so that

$$\|w\| \leq \left\|M^{-1}\right\| \|\tau_d\| \leq \bar{m}\bar{d}, \tag{4.4.27}$$

with $\bar{m}$ and $\bar{d}$ known for a given robot arm. Therefore,

$$\|e_i(t)\| \leq \|H(s)\| \, \|w\| \leq \|H(s)\| \, \bar{m}\bar{d} \tag{4.4.28}$$

$$\|\dot{e}_i(t)\| \leq \|sH(s)\| \, \|w\| \leq \|sH(s)\| \, \bar{m}\bar{d}. \tag{4.4.29}$$

Now selecting the L_2—norm, the operator gain $\|H(s)\|_2$ is the maximum value of the Bode magnitude plot of $H(s)$. For a critically damped system,

$$\sup_{\omega} \|H(j\omega)\|_2 = 1/k_{p_i}. \tag{4.4.30}$$

Therefore,

$$\|e_i(t)\|_2 \leq \overline{m}\overline{d}/k_{p_i}. \tag{4.4.31}$$

Moreover (see the Problems),

$$\sup_{\omega} \|j\omega H(j\omega)\|_2 = 1/k_{v_i}, \tag{4.4.32}$$

so that

$$\|\dot{e}_i(t)\|_2 \leq \overline{m}\overline{d}/k_{v_i}. \tag{4.4.33}$$

Thus, in the case of critical damping, the position error decreases with k_{pi} and the velocity error decreases with k_{vi}.

EXAMPLE 4.4–1: Simulation of PD Computed-Torque Control

In this example we intend to show the detailed mechanics of simulating a PD computed-torque controller on a digital computer.

a. Computed-Torque Control Law

In Example 3.2.2 we found the dynamics of the two-link planar elbow arm shown in Figure 4.2.1 to be

$$\begin{bmatrix} (m_1+m_2)a_1^2 + m_2a_2^2 + 2m_2a_1a_2\cos\theta_2 & m2a_2^2 + m_2a_1a_2\cos\theta_2 \\ m_2a_2^2 + m_2a_1a_2\cos\theta_2 & m2a_2^2 \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix}$$
$$+ \begin{bmatrix} -m_2a_1a_2\left(2\dot{\theta}_1\dot{\theta}_2 + \dot{\theta}_2^2\right)\sin\dot{\theta}_2 \\ m_2a_1a_2\dot{\theta}_1^2\sin\theta_2 \end{bmatrix}$$
$$+ \begin{bmatrix} (m_1+m_2)ga_1\cos\theta_1 + m_2ga_2\cos(\theta_1+\theta_2) \\ m_2ga_2\cos(\theta_1+\theta_2) \end{bmatrix}$$
$$= \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}. \tag{1}$$

These are in the standard form

$$M(q)\ddot{q} + V(q,\dot{q}) + G(q) = \tau. \tag{2}$$

Take the link masses as 1 kg and their lengths as 1 m.

The PD computed-torque control law is given as

$$\tau = M(q)(\ddot{q}_d + K_v\dot{e} + K_p e) + V(q,\dot{q}) + G(q), \tag{3}$$

with the tracking error defined as

$$e=q_d-q. \tag{4}$$

b. Desired Trajectory

Let the desired trajectory $q_d(t)$ have the components

$$\begin{aligned}\theta_{1d}&=g_1 \sin(2\pi t/T)\\ \theta_{2d}&=g_2 \cos(2\pi t/T)\end{aligned} \tag{5}$$

with period T=2 s and amplitudes g_i=0.1 rad≈6 deg. For good tracking select the time constant of the closed-loop system as 0.1 s. For critical damping, this means that K_v=diag{k_v}, K_P=diag{k_p}, where

$$\begin{aligned}\omega_n &= 1/0.1 = 10\\ k_p &= \omega_n^2 = 100, \quad k_v = 2\omega_n = 20.\end{aligned} \tag{6}$$

It is important to realize that the selection of controller parameters such as the PD gains depends on the performance objectives-in this case, the period of the desired trajectory.

c. Computer Simulation

Let us simulate the computed-torque controller using program TRESP in Appendix B. Simulation using commercial packages such as MATLAB and SIMNON is quite similar.

The subroutines needed for TRESP are shown in Figure 4.4.2. They are worth examining closely. Subroutine SYSINP (ITx, t) is called once per Runge-Kutta integration period and generates the reference trajectory $q_d(t)$, as well as $q_d(t)$, and $q_d(t)$. Note that the reference signal should be held constant during each integration period.

```
C   FILE 21nkct.FOR
C   IMPLEMENTATION OF COMPUTED-TORQUE CONTROLLER ON 2-LINK PLANAR ARM
C   SUBROUTINES FOR USE WITH TRESP
C
C   SUBROUTINE TO COMPUTE DESIRED TRAJECTORY
C      The trajectory value must be constant within each Runge-Kutta
C         integration interval
C
       SUBROUTINE SYSINP(IT,x,t)
       REAL x(*)
       COMMON/TRAJ/qd(6), qdp(6), qdpp(6)
       DATA gl, g2, per, twopi/0.1, 0.1, 2., 6.283/
C   COMPUTE DESIRED TRAJECTORY qd(t), qdp(t), qdpp(t)
       fact= twopi/per
       qd(1)=    gl*sin(fact*t)
       qd(2)=    g2*cos(fact*t)
       qdp(1)=   gl*fact*cos(fact*t)
       qdp(2)=  -g2*fact*sin(fact*t)
       qdpp(1)= -gl*fact**2*sin(fact*t)
       qdpp(2)= -g2*fact**2*cos(fact*t)
C
       RETURN
       END
C
C
C ***********************************************************
C   MAIN SUBROUTINE CALLED BY RUNGE-KUTTA INTEGRATOR
       SUBROUTINE F(t,x,xp)
       REAL x(*), xp(*)
C
C   COMPUTED-TORQUE CONTROLLER
       CALL CTL(x)
C   ROBOT ARM DYNAMICS
       CALL ARM(x,xp)
C
       RETURN
       END
C
C
C ***********************************************************
C   COMPUTED-TORQUE CONTROLLER SUBROUTINE
       SUBROUTINE CTL(x)
       REAL x(*),m1,m2,M11,M12,M22,N1,N2,kp,kv
       COMMON/CONTROL/tl, t2
C   The next line is to plot the errors and torques
       COMMON/OUTPUT/ e(2), ep(2), tpl, tp2
       COMMON/TRAJ/qd(6), qdp(6), qdpp(6)
       DATA ml,m2,al,a2, g/1.,1.,1.,1., 9.8/
       DATA kp, kv/ 100,20/

C   COMPUTE TRACKING ERRORS
       e(1) = qd(1) - x(1)
       e(2) = qd(2) - x(2)
```

```
      ep(1)= qdp(1) - x(3)
      ep(2)= qdp(2) - x(4)

C   COMPUTATION OF M(q), N(q,qp)
      M11= (m1+m2)*a1**2 + m2*a2**2 + 2*m2*a1*a2*cos(x(2))

      M12= m2*a2**2 + m2*a1*a2*cos(x(2))
      M22= m2*a2**2
      N1= -m2*a1*a2*(2*x(3)*x(4) + x(4)**2) * sin(x(2))
      N1= N1 + (m1+m2)*g*a1*cos(x(1)) + m2*g*a2*cos(x(1)+x(2))
      N2= m2*a1*a2*x(3)**2*sin(x(2))    +m2*g*a2*cos(x(1)+x(2))

C   COMPUTATION OF CONTROL TORQUES
      s1= qdpp(1) + kv*ep(1) + kp*e(1)
      s2= qdpp(2) + kv*ep(2) + kp*e(2)
      t1= M11*s1 + M12*s2 + N1
      t2= M12*s1 + M22*s2 + N2
C   The next lines are to plot the torque
      tp1= t1
      tp2= t2

      RETURN
      END
C
C
C *******************************************************
C   ROBOT ARM DYNAMICS SUBROUTINE
      SUBROUTINE ARM(x,xp)
      REAL x(*),xp(*),m1,m2,M11,M12,M22,MI11,MI12,MI22,N1,N2
      COMMON/CONTROL/t1, t2
C  The next line is to plot the Cartesian position
      COMMON/OUTPUT/ dum(6), y1, y2
      DATA m1,m2,a1,a2, g/1.,1.,1.,1., 9.8/
C   COMPUTATION OF M(q)
      M11= (m1+m2)*a1**2 + m2*a2**2 + 2*m2*a1*a2*cos(x(2))
      M12= m2*a2**2 + m2*a1*a2*cos(x(2))
      M22= m2*a2**2
C   INVERSION OF M(q) (For large n, use least-squares to find qpp)
      det= M11*M22 - M12**2
      MI11= M22/det
      MI12= -M12/det
      MI22= M11/det
C   NONLINEAR TERMS
      N1= -m2*a1*a2*(2*x(3)*x(4) + x(4)**2) * sin(x(2))
      N1= N1 + (m1+m2)*g*a1*cos(x(1))  + m2*g*a2*cos(x(1)+x(2))
      N2= m2*a1*a2*x(3)**2*sin(x(2))   + m2*g*a2*cos(x(1)+x(2))

C   STATE EQUATIONS
      xp(1)= x(3)
      xp(2)= x(4)
      xp(3)= MI11*(-N1 + t1) + MI12*(-N2 + t2)
      xp(4)= MI12*(-N1 + t1) + MI22*(-N2 + t2)
```

```
C  OUTPUT EQUATION - CARTESIAN POSITION

       yl= al*cos(x(1)) + a2*cos(x(1)+x(2))
       y2= al*sin(x(1)) + a2*sin(x(1)+x(2))

       RETURN
       END
```

Figure 4.4.2: Subroutines for simulation using TRESP.

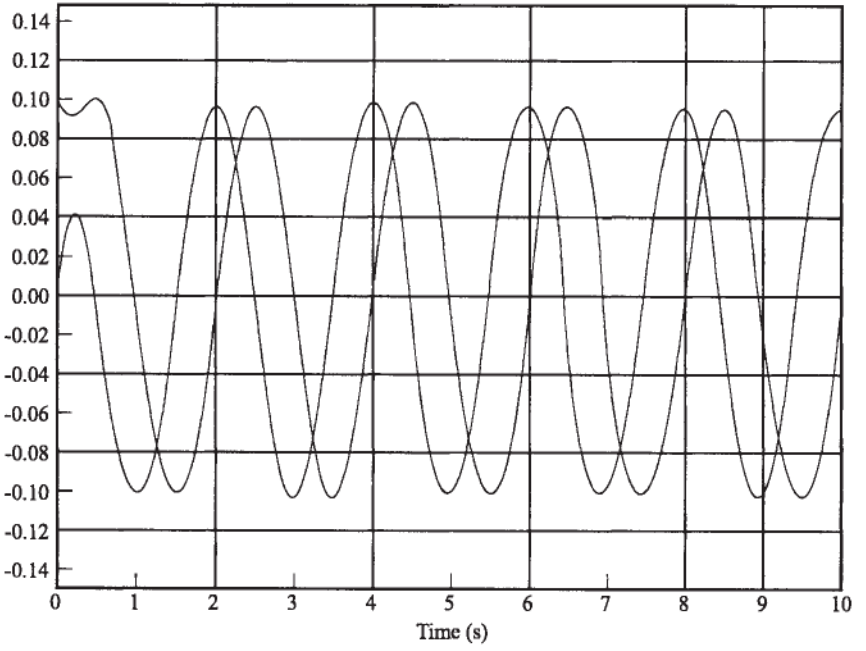


Figure 4.4.3: Joint angles $\theta_1(t)$ and $\theta_2(t)$ (red).

Subroutine F(time, x, xp) is called by Runge-Kutta and contains the continuous dynamics. This includes both the controller and the arm dynamics. The state to be integrated is $x = [\theta_1 \quad \theta_2 \quad \dot{\theta}_1 \quad \dot{\theta}_2]^T$, and the subroutine should compute the state derivative x (i.e., xp, which signifies x—prime).

Subroutine CTL(x) contains the controller (3). Note the structure of this subroutine. First, the tracking error $e(t)$ and its derivative are computed. Then $M(q)$ and $N(q, \dot{q}) = V(q, \dot{q}) + G(q)$ are computed. Finally, (3) is manufactured.

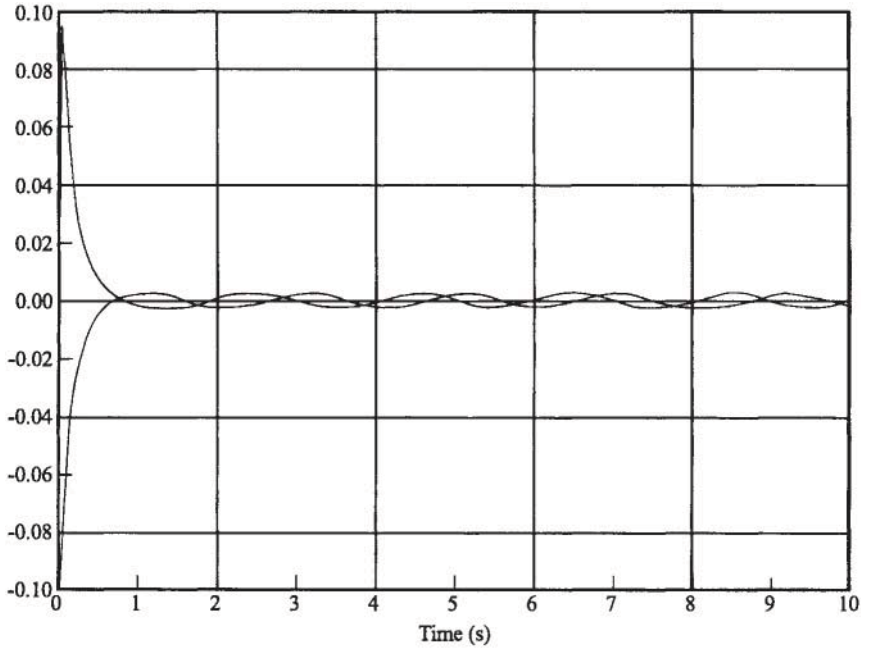


Figure 4.4.4: Tracking error $e_1(t)$, $e_2(t)$ (red).

Subroutine ARM(x, xp) contains the robot dynamics. First, $M(q)$ and $M^{-1}(q)$ are computed, and then $N(q,q)$. The state derivatives are then determined.

The results of the simulation are shown in the figures. Figure 4.4.3 shows the joint angles. Figure 4.4.4 shows the joint errors. The initial conditions result in a large initial error that vanishes within 0.6 s. Figure 4.4.5 shows the control torques; the larger torque corresponds to the inner motor, which must move two links.

It is interesting to note the ripples in $e(t)$ that appear in Figure 4.4.4. These are artifacts of the integrator. The Runge-Kutta integration period was T_R=0.01 s. When the simulation was repeated using T_R=0.001 s, the tracking error was exactly zero after 0.6 s. It should be zero, since computed-torque, or inverse dynamics control, is a scheme for canceling the nonlinearities in the dynamics to yield a second-order linear error system. If all the arm parameters are exactly known, this cancellation is exact. It is a good exercise to repeat this simulation using various values for the PD gains (see the Problems).

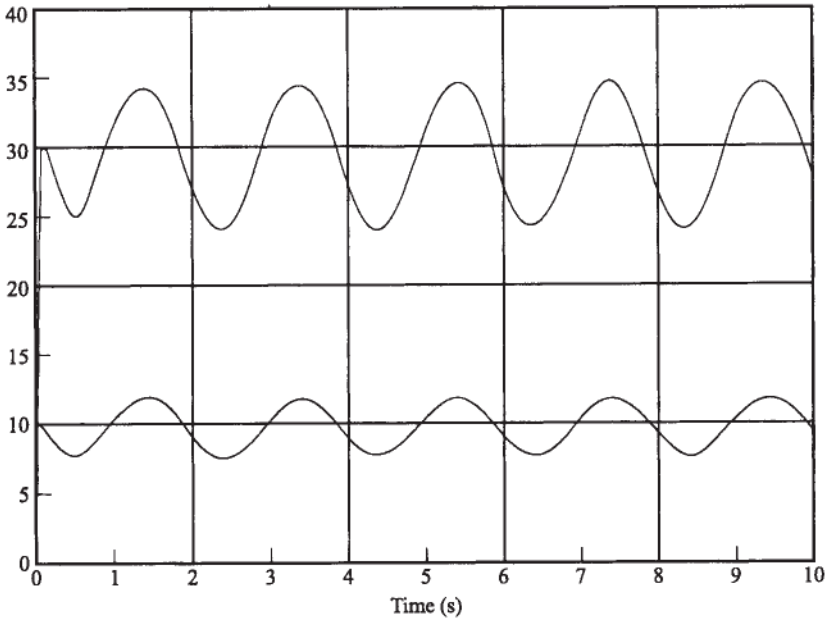


Figure 4.4.5: Torque inputs (N-m).

■

PID Outer-Loop Design

We have just seen that PD computed-torque control is very effective if all the arm parameters are known and there is no disturbance τ_d. However, from classical control theory we know that in the presence of constant disturbances, PD control gives a nonzero steady-state error. Consequently, we are motivated to make the system type I by including an integrator in the feedforward loop-this can be achieved using the *PID computed-torque controller*

$$\dot{\varepsilon} = e \tag{4.4.34}$$

$$u = -K_v\dot{e} - K_p e - K_i\varepsilon, \tag{4.4.35}$$

which yields the arm control input

$$\tau = M(q)\left(\ddot{q}_d + K_v\dot{e} + K_p e + K_i\varepsilon\right) + N(q,\dot{q}), \tag{4.4.36}$$

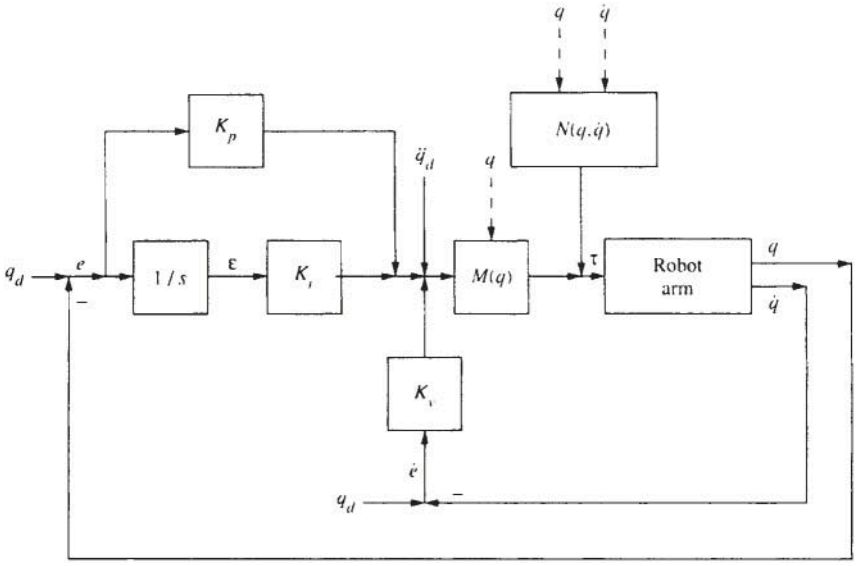


Figure 4.4.6: PID computed-torque controller.

with $e(t)$ the integral of the tracking error $e(t)$. Thus additional dynamics have been added to the linear outer-loop compensator.

This control law is conveniently described by defining the state as $x=[\varepsilon^T \quad e^T \quad \dot{e}^T]^T \in \mathbb{R}^{3n}$ and augmenting the error dynamics (4.4.8) with an integrator, so that

$$\frac{d}{dt}\begin{bmatrix}\varepsilon\\ e\\ \dot{e}\end{bmatrix}=\begin{bmatrix}0 & I & 0\\ 0 & 0 & I\\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\varepsilon\\ e\\ \dot{e}\end{bmatrix}+\begin{bmatrix}0\\ 0\\ I\end{bmatrix}u+\begin{bmatrix}0\\ 0\\ I\end{bmatrix}w. \tag{4.4.37}$$

A block diagram of the PID computed-torque controller appears in Figure 4.4.6. Then the closed-loop system is

$$\frac{d}{dt}\begin{bmatrix}\varepsilon\\ e\\ \dot{e}\end{bmatrix}=\begin{bmatrix}0 & I & 0\\ 0 & 0 & I\\ -K_i & -K_p & -K_v\end{bmatrix}\begin{bmatrix}\varepsilon\\ e\\ \dot{e}\end{bmatrix}+\begin{bmatrix}0\\ 0\\ I\end{bmatrix}w. \tag{4.4.38}$$

The closed-loop characteristic polynomial is

$$\Delta c(s)=|s^3I+K_vs^2+K_ps+K_i|. \tag{4.4.39}$$

Selecting diagonal control' gains

$$K_v = \text{diag}\{k_{v_i}\}, \quad K_p = \text{diag}\{k_{p_i}\}, \quad K_i = \text{diag}\{k_{i_i}\} \quad (4.4.40)$$

gives

$$\Delta_c(s) = \prod_{i=1}^{n} \left(s^3 + k_{v_i} s^2 + k_{p_i} s + k_{i_i}\right). \quad (4.4.41)$$

By using the Routh test it can be found that for closed-loop stability we require that

$$k_{i_i} < k_{v_i} k_{p_i}; \quad (4.4.42)$$

that is, the integral gain should not be too large.

Actuator Saturation and Integrator Windup. It is important to be aware of an effect in implementing PID control on any actual robot manipulator that can cause serious problems if not accounted for. Any real robot arm will have limits on the voltages and torques of its actuators. These limits may or may not cause a problem with PD control, but are virtually guaranteed to cause problems with integral control due to a phenomenon known as *integrator windup* [Lewis 1992].

Consider the simple case where $\tau = k_i \varepsilon$ with $\varepsilon(t)$ the integrator output. The torque input $\tau(t)$ is limited by its maximum and minimum values τ_{max} and τ_{min}. If $k_i \varepsilon(t)$ hits τ_{max}, there may or not may not be a problem. The problem arises if the integrator input remains positive, for then the integrator continues to integrate upwards and $k_i \varepsilon(t)$ may increase well beyond τ_{max}. Then, when the integrator input becomes negative, it may take considerable time for $k_i \varepsilon(t)$ to decrease below τ_{max}. In the meantime τ is held at τ_{max}, giving an incorrect control input to the plant.

Integrator windup is easy to correct using *antiwindup protection* in a digital controller. This is discussed in Section 4.5. The effects of uncorrected windup are demonstrated in Example 4.4.4.

The next example shows the usefulness of an integral term when there are unknown disturbances present.

EXAMPLE 4.4–2: Simulation of PID Computed-Torque Control

In Example 4.4.1 we simulated the PD computed-torque controller for a two-link planar arm. In this example we add a constant unknown

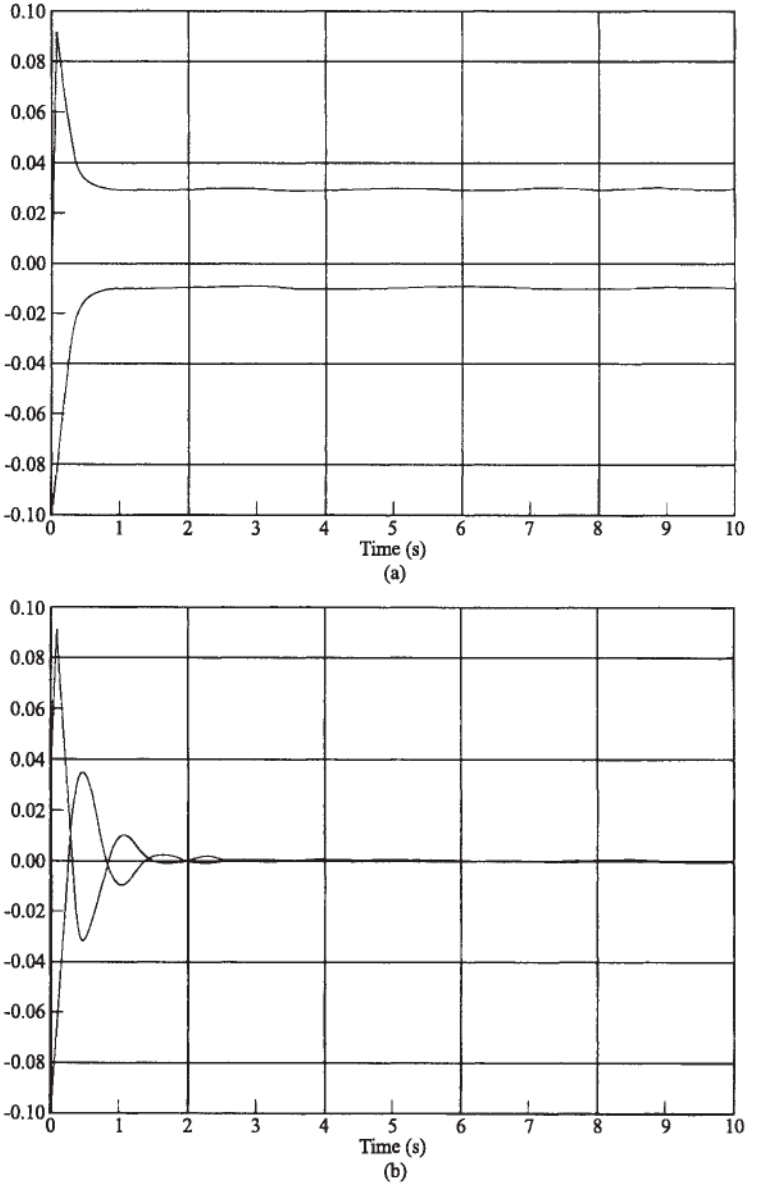


Figure 4.4.7: Computed-torque controller tracking errors $e_1(t)$, $e_2(t)$ (red): (a) PD control; (b) PID control.

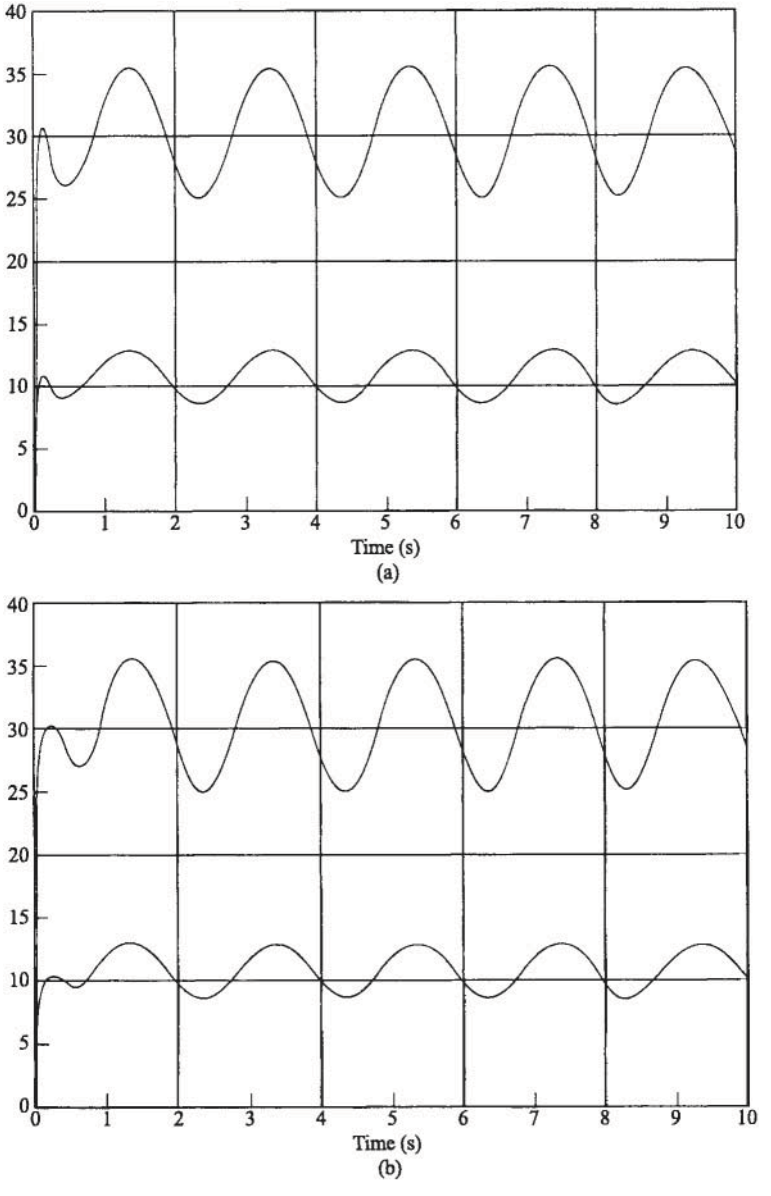


Figure 4.4.8: Computed-torque controller torque inputs (N-m): (a) PD control; (b) PID control.

disturbance to the arm dynamics and compare PD to PID computed torque.
Thus let the arm dynamics be

$$M(q)\ddot{q} + N(q,\dot{q}) + \tau_d = \tau, \tag{1}$$

with τ_d a constant disturbance with 1 N-m in each component. This could model unknown dynamics such as friction, and so on. The value of 1 N-m represents quite a large bias.

Adding 1 to the computation of the nonlinear terms N1 and N2 in subroutine arm(x, xp) in Figure 4.4.2 and using the PD computed-torque controller with k_p=100, k_v=20 yields the error plot in Figure 4.4.7(a). There is a unacceptable residual bias in the tracking error due to the unmodeled constant disturbance, which is not accounted for in the computed-torque law. The largest error is 0.033 rad, somewhat less than 2 deg.

Adding now an integral-error weighting term with k_i=500 yields the results in Figure 4.4.7(b), which show a zero steady-state error and are quite good.

The associated control torques are shown in Figure 4.4.8, which shows that the torque magnitudes are not appreciably increased by using the integral term.

To simulate the PID control law, it is necessary to add two additional states to x in subroutine arm(x, xp). It is convenient to call them x(5) and x(6), so that the lines added to the subroutine are

```
xp(5)=e(1).
xp(6)=e(2).
```
(2)

Thus it is now necessary to compute e(1) and e(2) in subroutine arm(x, xp) for integration purposes.

■

Class of Computed-Torque-Like Controllers

An entire class of *computed-torque-like* controllers can be obtained by modifying the computed-torque control law to read

$$\tau_c = \hat{M}(\ddot{q}_d - u) + \hat{N}. \tag{4.4.43}$$

The carets denote design choices for the weighting and offset matrices. One choice is $\hat{M}=M(q)$, $\hat{N}=N(q,q)$. The calculated control input into the robot arm is $\tau_c(t)$.

In some cases $M(q)$ is not known exactly (e.g., unknown payload mass), or $N(q, \dot{q})$ is not known exactly (e.g., unknown friction terms). Then $\hat{M}$ and $\hat{N}$ could be the best estimate we have for these terms. On the other hand, we might simply wish to avoid computing $M(q)$ and $N(q,q)$ at each sample time, or the sample period might be too short to allow this with the available hardware. From such considerations, we call (4.4.43) an "approximate computed-torque" controller.

In Table 4.4.1 are given some useful computed-torque-like controllers. As it turns out, computed torque is quite a good scheme since it has some important *robustness properties*. In fact, even if $\hat{M}\neq M$ and $\hat{N}\neq N$ the performance of controllers based on (4.4.43) can be quite good if the outer-loop gains are selected large enough. We study robustness formally in Chapter 4.

In the remainder of this chapter we consider various special choices of $\hat{M}$ and $\hat{N}$ that give some special sorts of controllers. We shall present some theorems and simulation examples that illustrate the robustness properties of computed-torque control.

Error Dynamics with Approximate Control Law. Let us now derive the error dynamics if the approximate computed-torque controller (4.4.43) is applied to the robot arm (4.4.2). Substituting $\tau_c(t)$ into the arm equation for $\tau(t)$ yields

$$M\ddot{q} + N + \tau_d = \hat{M}\left(\ddot{q}_d - u\right) + \hat{N}$$
$$\hat{M}\ddot{q}_d - M\ddot{q} = \hat{M}u + \tau_d + \left(N - \hat{N}\right).$$

Adding Mq_d-Mq_d to the left-hand side and Mu-Mu to the right gives

$$M\ddot{e} = Mu - \left(M - \hat{M}\right)u + \tau_d + \left(M - \hat{M}\right)\ddot{q}_d + \left(N - \hat{N}\right)$$

or

$$\ddot{e}=u-\Delta u+d, \tag{4.4.44}$$

where the inertia and nonlinear-term model mismatch terms are

$$\Delta = M^{-1}\left(M - \hat{M}\right) = I - M^{-1}\hat{M} \tag{4.4.45}$$

$$\delta = M^{-1}\left(N - \hat{N}\right) \tag{4.4.46}$$

Table 4.4.1: Computed-Torque-Like Robot Controllers.

Robot Dynamics:

$$M(q)\ddot{q} + V(q,\dot{q}) + F_v\dot{q} + F_d(\dot{q}) + G(q) + \tau_d = \tau$$

or

$$M(q)\ddot{q} + N(q,\dot{q}) + \tau_d = \tau$$

where

$$N(q,\dot{q}) \equiv V(q,\dot{q}) + F_v\dot{q} + F_d(\dot{q}) + G(q)$$

Tracking Error:

$$e(t) = q_d(t) - q(t)$$

PD Computed-Torque:

$$\tau = M(q)(\ddot{q}_d + K_v\dot{e} + K_p e) + N(q,\dot{q})$$

PID Computed-Torque:

$$\dot{\varepsilon} = e$$

$$\tau = M(q)(\ddot{q}_d + K_v\dot{e} + K_p e + K_i\varepsilon) + N(q,\dot{q})$$

Approximate Computed-Torque Control:

$$\tau_c = \hat{M}(\ddot{q}_d - u) + \hat{N}$$

Error System:

$$\ddot{e} = (I - \Delta)u + d$$

$$\text{where } \Delta = I - M^{-1}\hat{M},\ \delta = M^{-1}\left(N - \hat{N}\right)$$

$$d(t) = M^{-1}\tau_d + \Delta\ddot{q}_d(t) + \delta(t)$$

PD-Gravity Control:

$$\tau_c = K_v\dot{e} + K_p e + G(q)$$

Classical PD Control:

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

Classical PID Control:

$$\dot{\varepsilon} = e$$

$$\tau_c = K_v\dot{e} + K_p e + K_i\varepsilon$$

Digital Control (see Section 4.5):

$$\tau_k = M(q_k)\left(\ddot{q}_k^d + K_v\dot{e}_k + K_p e_k\right) + N(q_k,\dot{q}_k)$$

and the disturbance is

$$d(t) = M^{-1}\tau_d + \Delta\ddot{q}_d(t) + \delta(t). \tag{4.4.47}$$

This reduces to the error system (4.4.10) if exact computed-torque control is used so that Δ=0, δ=0. Otherwise, *the error system is driven by the desired acceleration and the nonlinear term mismatch $\delta(t)$*. Thus the tracking error will never go exactly to zero. Moreover, the auxiliary control $u(t)$ is multiplied by $(I - \Delta)$, which can make for a very difficult control problem.

Using outer-loop PD feedback so that $u(t)=-K_v e-K_p e$ yields the error system

$$\ddot{e} + K_v\dot{e} + K_p e = \Delta(K_v\dot{e} + K_p e) + d(t). \tag{4.4.48}$$

The behavior of such systems is not obvious, even if K_v and K_p are selected for good stability of the left-hand side. There are two sorts of problems: first, the disturbance term $d(t)$, and second the function $\Delta(K_v e+K_p e)$ of the error and its derivative.

PD-Plus-Gravity Controller

A useful controller in the computed-torque family is the *PD-plus-gravity controller* that results when $M=I$, $N=G(q)-q_d$, with $G(q)$ the gravity term of the manipulator dynamics. Then, selecting PD feedback for $u(t)$ yields

$$\tau_c = K_v\dot{e} + K_p e + G(q). \tag{4.4.49}$$

This control law was treated in [Arimoto and Miyazaki 1984], [Schilling 1990]. It is much simpler to implement than the exact computedtorque controller.

When the arm is at rest, the only nonzero terms in the dynamics (4.4.1) are the gravity $G(q)$, the disturbance τ_d, and possibly the control torque τ. The PD-gravity controller τ_c, includes $G(q)$, so that we should expect good performance for set-point tracking, that is, when a constant q_d is given so that q_d=0. The next result formalizes this. It relies on a Lyapunov proof (Chapter 1) of the sort that will be of consistent usefulness throughout the book, drawing especially on the skew-symmetry property in Table 3.3.1. Thus it is very important to understand the steps in this proof.

THEOREM 4.4–1: *Suppose that PD-gravity control is used in the arm dynamics (4.4.1) and τ_d=0, q_d=0. Then the steady-state tracking error. e=q_d-q is zero.*

Proof:

1. Closed-Loop System

Ignoring friction, the robot dynamics are given by

$$M(q)\ddot{q} + V_m(q,\dot{q})\dot{q} + G(q) = \tau. \tag{1}$$

When q_d=0, the proposed control law (4.4.49) yields the closed-loop dynamics

$$M\ddot{q} + V_m\dot{q} + K_v\dot{q} - K_p e = 0. \tag{2}$$

2. Lyapunov Function

Select now the Lyapunov function

$$V = \tfrac{1}{2}\left(\dot{q}^T M\dot{q} + e^T K_p e\right) \tag{3}$$

and differentiate to obtain

$$\dot{V} = \dot{q}^T\left(M\ddot{q} + \tfrac{1}{2}\dot{M}\dot{q} - K_p e\right). \tag{4}$$

Substituting the closed-loop dynamics (2) yields

$$\dot{V} = \dot{q}^T\left(\tfrac{1}{2}\dot{M} - V_m\right)\dot{q} = \dot{q}^T K_v\dot{q} \tag{5}$$

Now, the skew symmetry of the first term gives

$$\dot{V} = -\dot{q}^T K_v\dot{q}. \tag{6}$$

The state is $x = [e^T \quad \dot{q}^T]^T$, so that V is positive definite but $\dot{V}$ only negative semidefinite. Therefore, we have demonstrated stability in the sense of Lyapunov, that is, that the error and joint velocity are both bounded.

3. Asymptotic Stability by LaSalle's Extension

The asymptotic stability of the system may be demonstrated using Barbalat's lemma and a variant of LaSalle's extension [Slotine and Li 1991] (Chapter 2). Thus it is necessary to demonstrate that the only invariant set contained in the set V=0 is the origin.

Since V is lower bounded by zero and nonpositive, it follows that V approaches a finite limit, which can be written

$$-\int_0^\infty \dot{V}\,dt = const. \tag{7}$$

We now invoke Barbalat's lemma to show that V goes to zero. For this, it is necessary to demonstrate the uniform continuity of V. We see that

$$\ddot{V} = -2\dot{q}^T K_v \ddot{q}. \tag{8}$$

The demonstrated stability shows that a and q are bounded, whence (2) and the boundedness of $M^{-1}(q)$ (see Table 3.3.1) reveal that q is bounded. Therefore, V is bounded, whence V is uniformly continuous. This guarantees by Barbalat's lemma that V goes to zero.

It is now clear that $\dot{\mathbf{q}}$ goes to zero. It remains to show that the tracking error e(t) vanishes. Note that when q=0, (2) reveals that

$$\ddot{q} = M^{-1} K_p e. \tag{9}$$

Therefore, a nonzero e(t) results in nonzero $\ddot{\mathbf{q}}$ and hence in q≠0, a contradiction. Therefore, the only invariant set contained in {x(t)|V=0} is x(t)=0. This finally demonstrates that both e(t) and vanish and concludes the proof. ■

Some notes on this proof are warranted. First, note that the Lyapunov function chosen is a natural one, as it contains the kinetic energy and the "artificial potential energy" associated with the virtual spring in the control law [Slotine and Li 1991]. K_p appears in V while K_v appears in $\dot{V}$.

The proof of Lyapunov stability is quick and easy. The effort comes about in showing asymptotic stability. For this, it is required to return to the system dynamics and study it more carefully, discussing issues such as boundedness of signals, invariant sets, and so on. The fundamental property of skew symmetry is used in computing $\dot{V}$. The boundedness of $M^{-1}(q)$ is needed in Barbalat's lemma.

The next example illustrates the performance of the PD-gravity controller for trajectory tracking. In general, if $q^d \neq 0$, then the PD-gravity controller guarantees *bounded tracking errors*. The error bound decreases, that is, the tracking performance improves, as the PD gains become larger. This will be made rigorous in Chapter 4.

EXAMPLE 4.4–3: Simulation of PD-Gravity Controller

In Example 4.4.1 we simulated the exact computed-torque control law on a two link planar manipulator. Here, let us simulate the PD-gravity controller. We shall take the same arm parameters and desired trajectory.

Assuming identical PD gains for each link, the PD-gravity control law for the two-link arm is

$$\begin{aligned} \tau_{c1} &= k_v \dot{e}_1 + k_p e_1 + (m_1 + m_2)\, g a_1 \cos\theta_1 + m_2 g a_2 \cos(\theta_1 + \theta_2) \\ \tau_{c2} &= k_v \dot{e}_2 + k_p e_2 + m_2 g a_2 \cos(\theta_1 + \theta_2)\,. \end{aligned} \tag{1}$$

This is easily simulated by commenting out several lines of code and making a few other modifications in subroutine CTL(x) of Figure 4.4.2.

For critical damping, the PD gains are selected as

$$k_p = \omega_n^2, \quad k_v = 2\omega_n. \tag{2}$$

The results of the PD-gravity controller simulation for several values of w_n are shown in Figure 4.4.9. As ω_n, and hence the PD gains, increases, the tracking performance improves. However, no matter how large the PD gains, the tracking error never goes exactly to zero, but is bounded about zero by a ball whose radius decreases as the gains become larger. The performance for ω_n=50, corresponding to k_p=2500, k_v=100, would be quite suitable for many applications.

It quite important to note that the dc value of the errors is equal to zero. One can consider the gravity terms as the "dc portion" of the computed-torque control law. If they are included in computing the torques, there will be no error offset.

The associated control input torques are shown in Figure 4.4.10. It is extremely interesting to note that the torques are *smaller for the higher gains*. This is contrary to popular belief, which assumes that the control torques always increase with increasing PD gains. It is due to the fact that larger gains give "tighter" performance, and hence smaller tracking errors.

In view of the fact that the errors in Figure 4.4.9(a) have different magnitudes, it would probably be more reasonable to take the PD gains larger for the inner link than the outer link.

■

Classical Joint Control

A simple controller that often gives good results in practice is obtained by selecting in (4.4.43)

$$\hat{M} = I, \quad \hat{N} = -\ddot{q}_d. \tag{4.4.50}$$

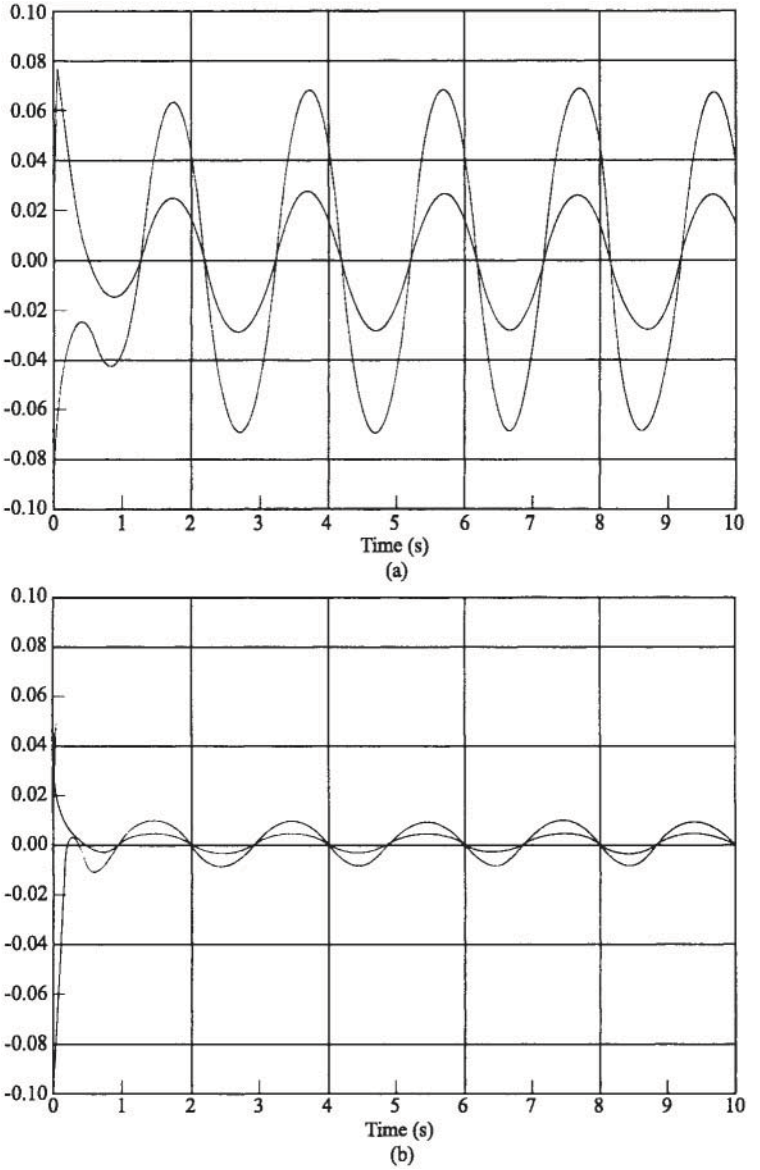


Figure 4.4.9: PD-gravity tracking error $e_1(t)$, $e_2(t)$ (red): (a) w_n=10 rad/s; (b) w_n=25 rad/s.

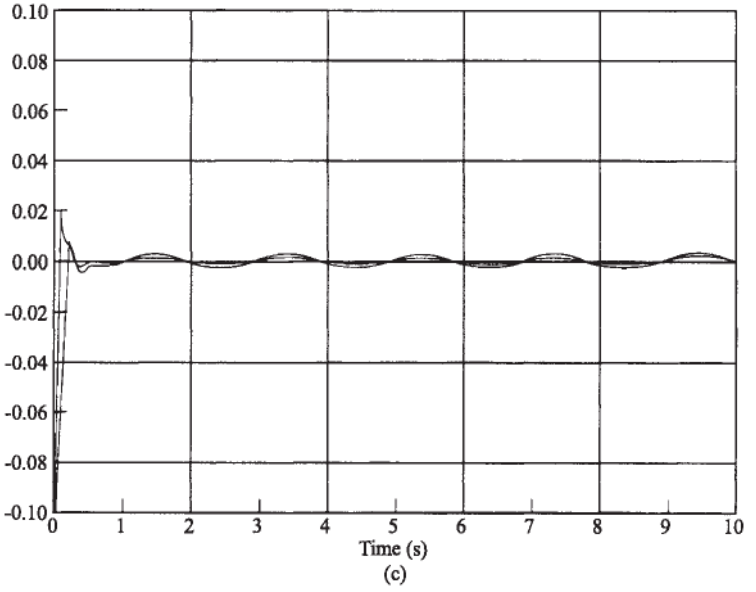


Figure 4.4.9: (*Cont.*) (c)w_n=50 rad/s.

Then there results

$$\tau_c = -u. \tag{4.4.51}$$

By selecting u_i to depend only on joint variable i, this describes n decoupled individual joint controllers and is called *independent joint control*. Implementation on an actual robot arm is easy since the control input for joint i only depends on locally measured variables, not on the variables of the other joints. Moreover, the computations are easy and do not involve solving the complicated nonlinear robot inverse dynamics.

During the early days of robotics, independent joint control was popular [Paul 1981] since it allows a decoupled analysis of the closed-loop system using single-input/single-output (SISO) classical techniques. It is also called *classical joint control*. There are strong arguments even today by several researchers that this control scheme is always suitable in practical implementations, and that modern nonlinear control schemes are too complicated for industrial robotic applications.

A traditional analysis of independent joint control control now follows. It provides a connection with classical control notions and is important for the

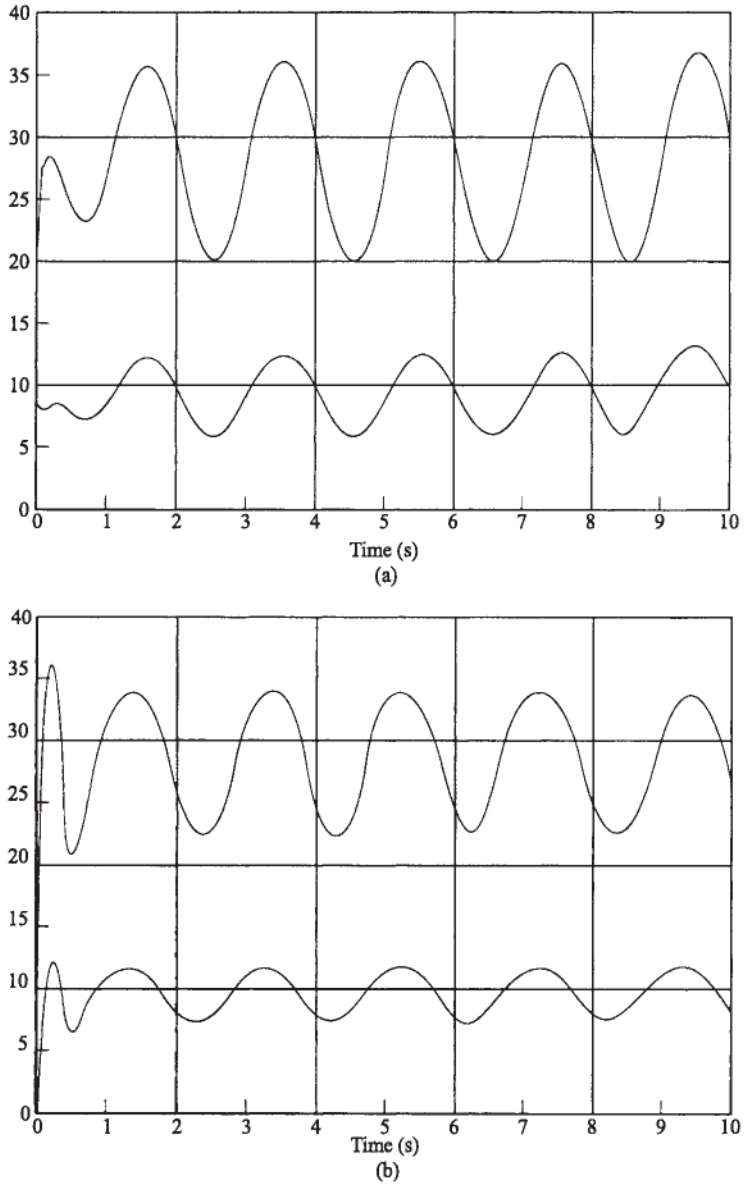


Figure 4.4.10: PD-gravity torque inputs (N-m): (a) w_n=10 rad/s; (b) w_n=25 rad/s.

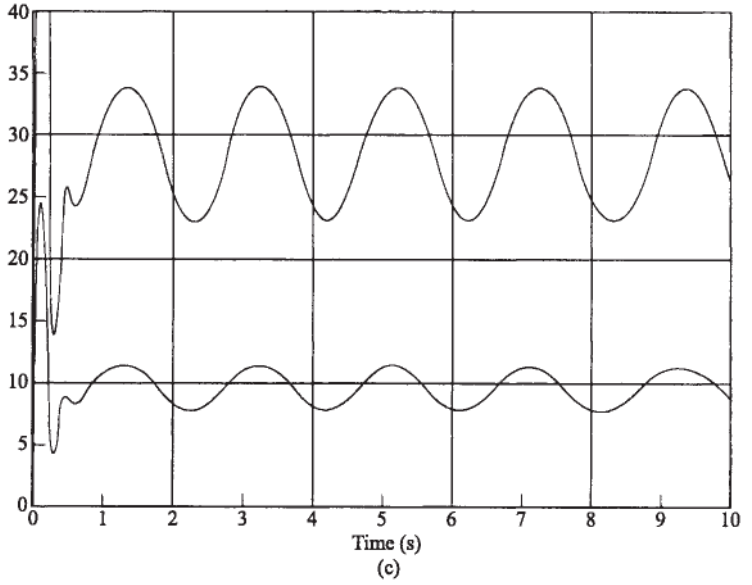


Figure 4.4.10: *(Cont)* (c) w_n=50 rad/s.

robot controls designer to understand. See [Franklin et al. 1986] for a reference on classical control theory.

A simplified dynamical model of a robot arm with electric actuators may be written as (Section 3.6)

$$\left(J_m + r_i^2 m_{ii}\right)\ddot{\theta}_i + \left(B_m + \frac{k_b k_m}{R}\right)\dot{\theta}_i = \frac{k_m}{R} v_i - r_i d_i, \quad i = 1, \cdots, n \tag{4.4.52}$$

with J_m the actuator motor inertia, B_m the rotor damping constant, k_m the torque constant, k_b the back emf constant, R the armature resistance, and r_i the gear ratio for joint i. The motor angle is denoted $\theta_i(t)$. The constant portions of the diagonal elements of $M(q)$ are denoted m_{ii}. The time-varying portions of these elements, as well as the off-diagonal elements of $M(q)$, the nonlinear terms $N(q, \dot{q})$, and any disturbances τ_d are all lumped into the disturbance $d_i(t)$. Thus $d_i(t)$ contains the effects on joint i of all the other joints. The control input is the motor armature voltage $v_i(t)$.

Note that predominantly motor parameters appear in this equation. In fact, if the gear ratio is small, even m_{ii} may be neglected. For this reason, if the gear ratio is small, the robot arm control problem virtually reduces to the problem of controlling the actuator motors.

Let us denote this simplified linear time-invariant model of manipulator joint i as

$$J\ddot{\theta} + B\dot{\theta} = u - rd. \tag{4.4.53}$$

where the constant k_m/R has been incorporated into the definitions of J and B. According to the properties in Table 3.3.1, the disturbance $d(t)$ is bounded, although not generally by a constant. The bound may be a function of $q(t)$, $\dot{q}(t)$, and even $\ddot{q}(t)$. This is generally ignored in a classical analysis. The effects of $d(t)$ are, however, somewhat ameliorated by multiplication by the gear ratio r. The effects of joint compliance (Chapter 6) are also ignored in classical joint control design. [For consistency with classical notation, there is a sign change in this section in the definition of $u(t)$ compared to our previous usage.]

Now, let us consider a few selections for the control input $u(t)$.

PD Control. Selecting the PD control law

$$u = k_v e + k_p e, \tag{4.4.54}$$

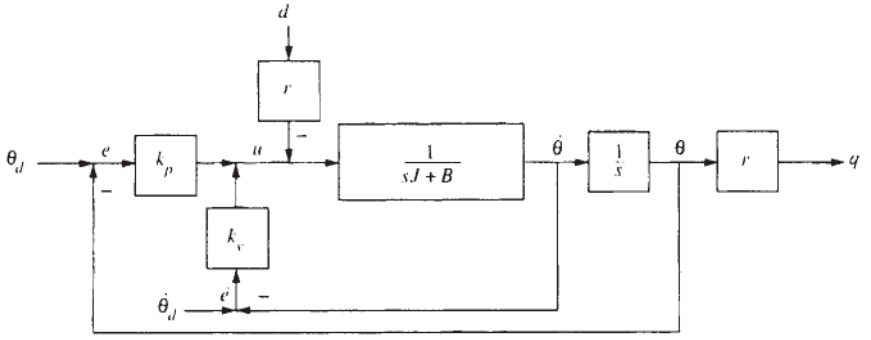


Figure 4.4.11: PD independent joint control.

with $e(t) = \theta_{di}(t) - \theta_i(t)$ the tracking error for motor angle i, yields the closed-loop system shown in Figure 4.4.11. Recall that the motor angle is related to the joint variable by $q_i = r_i\theta_i$. Thus $\theta_d(t)$ may be computed from $q_{di}(t)$. This PD controller is easily implemented on an actual arm with very little computing power.

Using Mason's theorem from classical control, the closed-loop transfer function for set-point tracking (e.g., with $\theta_d = 0$) is found to be (see the problems)

$$\theta = \frac{k_p}{\Delta(s)}\theta_d - \frac{r}{\Delta(s)}d \tag{4.4.55}$$

with the closed-loop characteristic polynomial

$$\Delta(s) = s^2 J + s(B + k_v) + k_p. \tag{4.4.56}$$

The PD gains can be selected to obtain a suitable natural frequency and damping ratio, as we have seen.

At steady state, the only nonzero contribution to the disturbance *d(t)* is the neglected gravity *G(q)*. Table 3.3.1 shows that the gravity vector is bounded by a known value g_b for a given robot arm. Therefore, at steady state,

$$|d| < g_b.$$

Using the final value theorem, the steady-state tracking error for joint *i* using PD control is bounded by

$$e_{ss} < r g_b / k_p. \tag{4.4.57}$$

Therefore, for set-point tracking where a final value for q_d is specified and $\dot{q}_d = 0$ PD control with a large k_p might be suitable.

As a matter of fact, the next results show that PD control is often very suitable even for following a desired trajectory, not only for set-point control. It is proven in [Dawson 1990].

THEOREM 4.4–2: *If the PD control law (4.4.54) is applied to each joint and* $e(0)=0$, $\dot{e}(0)=0$, *the position and velocity tracking errors are bounded within a ball whose radius decreases approximately (for large* k_v*) as* $1/\sqrt{k_v}$.

■

This result gives credence to those who maintain that PD control is often good enough for practical applications.

Of course, the point is that k_v cannot be increased without limit without hitting the actuator torque limits. Other schemes to be discussed in the book allow good trajectory following without such large torques.

In [Paul 1981] are discussed several methods for modifying the PD control law to obtain better performance. These include gravity compensation [which yields exactly controller (4.4.49)], and acceleration feedforward, which amounts to using

$$u = k_v \dot{e} + k_p e + J\ddot{\theta}_{d_i} \tag{4.4.58}$$

for each joint. Also mentioned is joint-coupling control, which amounts to adding back into $u(t)$ some of the neglected terms in $M(q)$ and $N(q, \dot{q})$ that describe the interactions between the joints. Thus such corrections involve using better estimates for $\hat{M}$ and $\hat{N}$ in the approximate computed-torque control (4.4.43).

PID Control. It has been seen that PD independent joint control is often suitable for tracking control. However, at steady state there is a residual error (4.4.57) due to gravity. This can be removed using the *PID independent joint control law*

$$\begin{aligned} \dot{\varepsilon} &= e \\ u &= k_v \dot{e} + k_p e + k_i \varepsilon \end{aligned} \tag{4.4.59}$$

for each joint. See Figure 4.4.12.

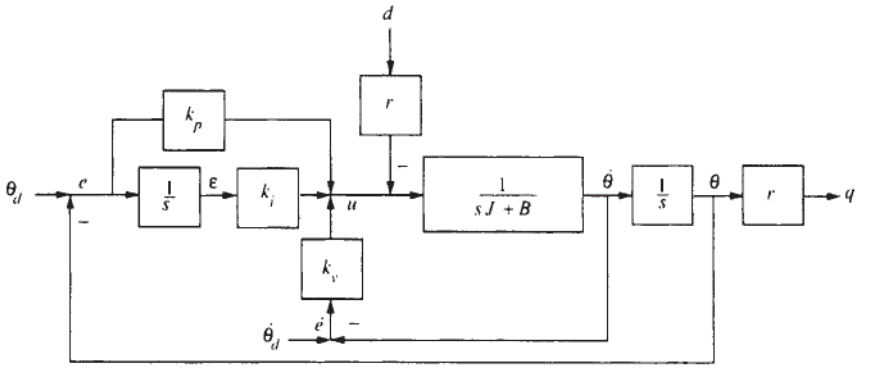


Figure 4.4.12: PID independent joint control.

Now the transfer function for set-point tracking $(\dot{\theta}_d = 0)$ is

$$\theta = \frac{k_p s + k_i}{\Delta_1(s)} \theta_d - \frac{rs}{\Delta_1(s)} d, \tag{4.4.60}$$

with closed-loop characteristic polynomial

$$\Delta_1(s) = s^3 J + s^2 (B + k_v) + s k_p + k_i. \tag{4.4.61}$$

Now the final value theorem shows that the steady-state error for set-point control is zero. The Routh test shows that for stability it is required that

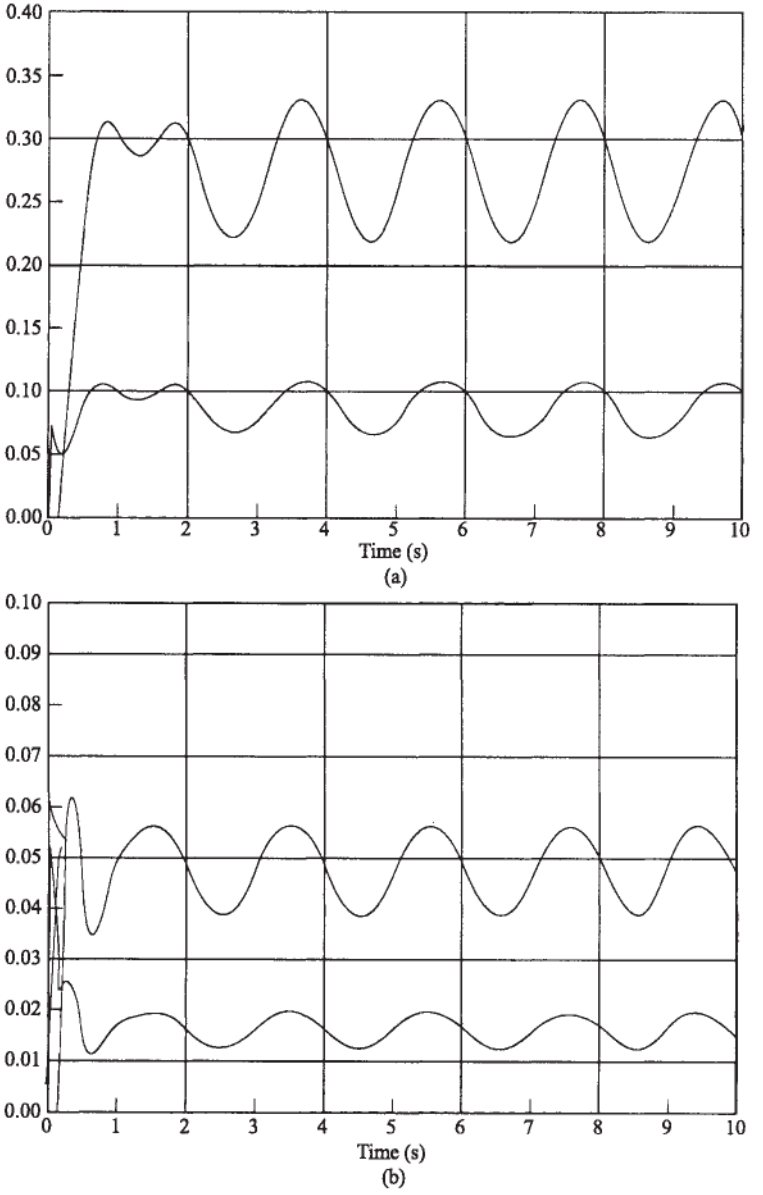


Figure 4.4.13: PD classical joint control tracking error $e_1(t)$, $e_2(t)$ (red): (a) ω_n=10 rad/s; (b) ω_n=25 rad/s.

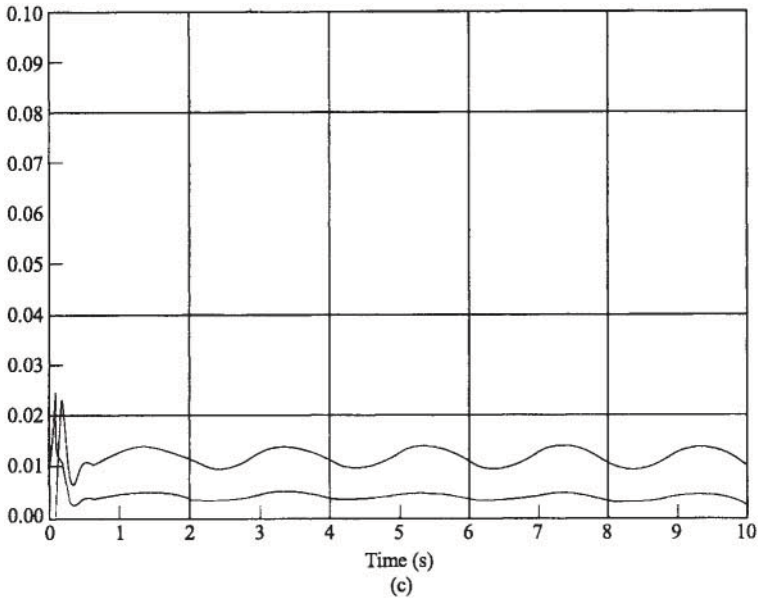


Figure 4.4.13 (*Cont.*) (c) ω_n=50 rad/s.

$$k_i < (B + k_v)\, k_p / J. \tag{4.4.62}$$

In using an integral term in the control law, it is important to be aware of the possibility of *integrator windup* due to actuator saturation limits, which was discussed earlier in the section "PID Outer-Loop Design."

EXAMPLE 4.4–4: Classical Joint Control and Torque Saturation Limits

In Example 4.4.1 we simulated the exact computed-torque control law for a two-link planar elbow arm. Here we want to show the results of using PD and PID classical joint control on the same arm (with the same desired trajectory). We are also interested in demonstrating the effects of torque saturation limits. To highlight the effects without extraneous details, we shall use only the arm dynamics and no actuator dynamics or gear ratio. These may easily be included by making some slight modifications to the subroutine arm(x, xp) in Figure 4.4.2.

a. PD Independent Joint Control

For the two-link arm, PD independent joint control is simply

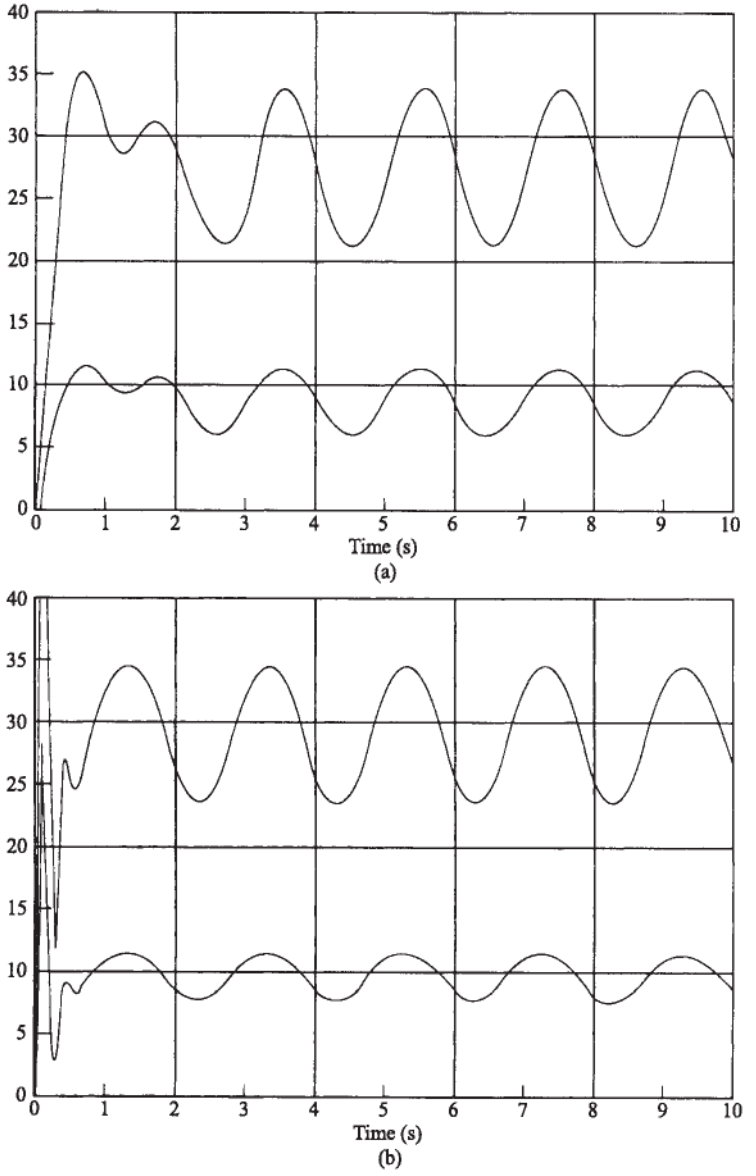


Figure 4.4.14: PD classical joint control torque inputs (N-m): (a) ω_n=10 rad/s; (b) ω_n=25 rad/s.

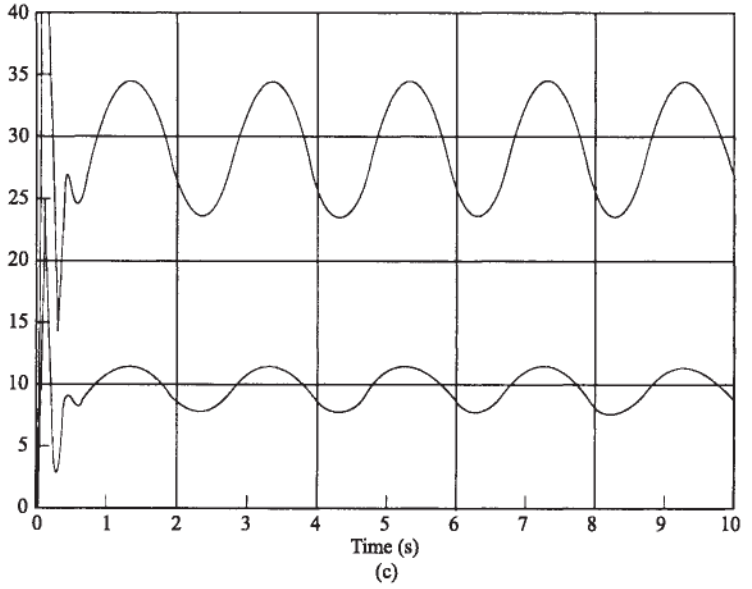


Figure 4.4.14 (*Cont.*) (c) ω_n=50 rad/s.

$$\tau_j = k_v \dot{e}_j + k_p e_j, \quad j = 1, 2, \tag{1}$$

with the tracking error $e(t)=q_d(t)-q(t)$. This control law is very simple and direct to implement on an actual arm without even using a DSP. It is much simpler than the control in Example 4.4.1.

The results of using the PD controller on the two-link arm are shown in Figure 4.4.13 for several values of a closed-loop natural frequency w_n. Recall that the PD gains for critical damping are

$$k_p = \omega_n^2, \quad k_v = 2\omega_n. \tag{2}$$

The low-gain results (k_p=100, k_v=20) in part (a) of the figure are very bad (note the scale). However, the high-gain results in part (c) are much better. Even higher gains would cause a further tracking error decrease.

It is important to notice that there is in all cases a do component of the tracking error. This is due to the fact that the gravity terms are neglected and should be contrasted with the results of Example 4.4.3.

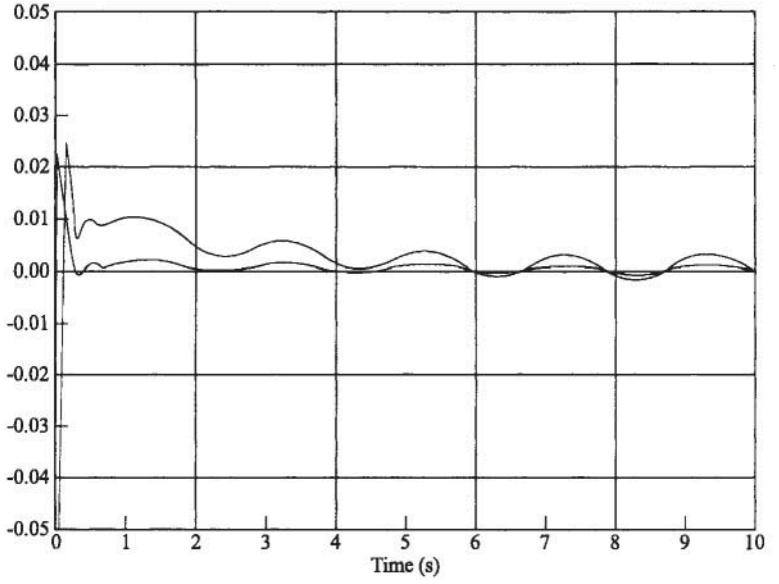


Figure 4.4.15: PD classical joint control: k_v=100, k_p=2500, k_i=1000.

Adding the gravity terms in the control law, therefore, significantly increases the tracking performance.

The associated control torques are shown in Figure 4.4.14.

b. PID Independent Joint Control

PID independent joint control for the two-link arm is

$$\begin{aligned} \dot{\varepsilon}_j &= e_j. \\ \tau_j &= k_v \dot{e}_j + k_p e_j + k_i \varepsilon_j, \quad j = 1, 2, \end{aligned} \tag{3}$$

This simple law is easily implemented by adding two states, x(5) and x(6), to subroutine arm(x, xp) in Figure 4.4.2 to account for the integrators (see Example 4.4.2).

The simulation in Figure 4.4.13(c) was repeated, but now adding an integral gain of k_i=1000. The result is shown in Figure 4.4.15. After several iterations of the sinusoidal motion, the do value of the tracking errors goes to zero, so that the performance is much improved. That is, adding an integral term can compensate for neglecting the gravity terms in the control

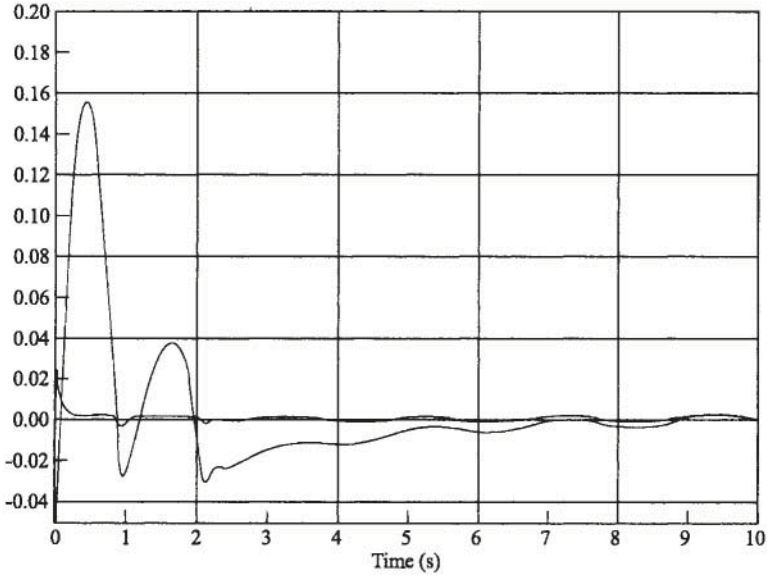


Figure 4.4.16: PID classical joint control with torque limits of ±35 N-m. Tracking error in rads.

law. However, it takes awhile for the error to converge to a zero do value, and the results are still not as good as the PD-gravity controller in Example 4.4.3 for the same PD gains. On the other hand, the integral term can reject other terms besides gravity (e.g., friction of some sorts). The torque control input was virtually the same in form as Figure 4.4.14(c).

c. Actuator Saturation Limits

We discussed integrator windup due to actuator saturation limits earlier in the section "PID Outer-Loop Design." Here we should like to demonstrate its deleterious effects.

The control torque in part (b) of this example is similar to the torque in Figure 4.4.14(c); it is fairly well behaved, except for some frantic action near time zero, where the maximum positive excursion is 78 N-m.

Suppose now that there are torque limits of t_{max}=35 N-m, t_{min}= -35 N-m. This is easily simulated by modifying subroutine CTL(x) in Figure 4.4.2 to include a saturation function by adding the lines

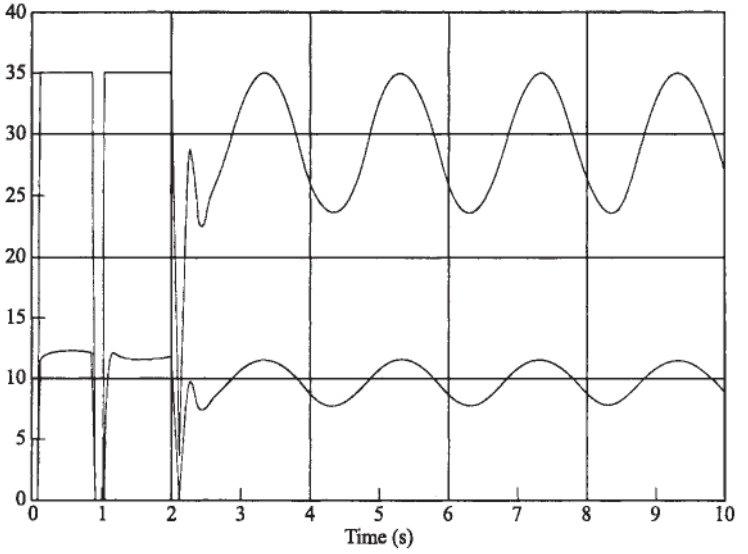


Figure 4.4.17: PID classical joint control with torque limits of ±35 N-m. Torque inputs in N-m.

$$\begin{aligned} &\text{IF (abs (t1) .gt.tmax) t1} = \text{sign (tmax, t1)} \\ &\text{IF (abs (t2) .gt.tmax) t2} = \text{sign (tmax, t2)} \end{aligned} \tag{4}$$

The results of the simulation with these limits are shown in Figure 4.4.16 and are terrible. The torque is shown in Figure 4.4.17.

In Section 4.5 we show how to ameliorate the effects of actuator saturation by using *antiwindup protection* in a digital controller. The simulation results for PD control with actuator limits are not as bad as the ones just shown for PID control, since saturation limits have less effect if no integrator is present.

■

4.5 Digital Robot Control

Many robot control schemes are complicated and involve a great deal of computation for the evaluation of nonlinear terms. Therefore, they are implemented as digital control laws on digital signal processors (DSPs). Certain

sorts of digital controllers for robot arms can be considered as members of the computer-torque-like class (see Table 4.4.1).

One approach to digital robot control is shown in Figure 4.5.1. There, a sampler is placed on $q(t)$ and $\dot{q}(t)$ to define

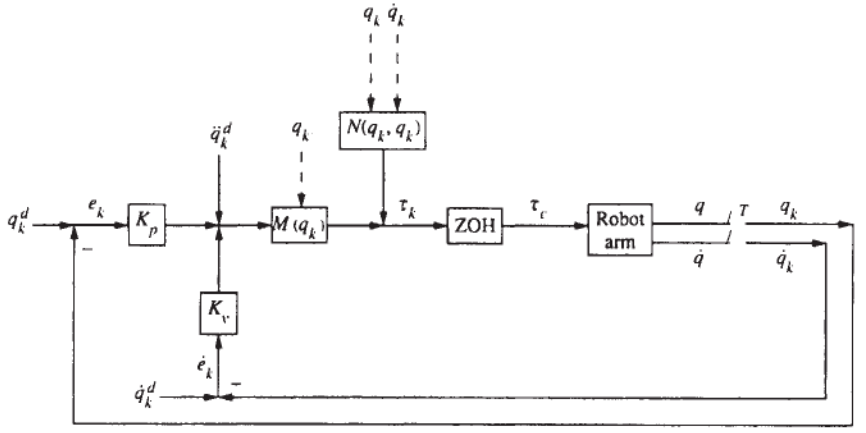


Figure 4.5.1: Digital robot control scheme.

$$\begin{aligned} q_k &\equiv q(kT) \in \mathrm{R}^n \\ \dot{q}_k &\equiv \dot{q}(kT) \in \mathrm{R}^n, \end{aligned} \tag{4.5.1}$$

with T the sample period. Sample periods in robotic applications vary from about 1 to 20 ms. A zero-order hold is used to reconstruct the continuous-time control input $\tau(t)$needed for the actuators from the samples τ_k. In this section we use superscripts *"d"* for the desired trajectory for notational ease.

This digital control law amounts to selecting

$$\hat{M} = M(q(kT)), \quad \hat{N} = N(q(kT), \dot{q}(kT)) \tag{4.5.2}$$

in the approximate computed-torque controller in Table 4.4.1 and a digital outer loop control signal u_k. Then, with PD outer-loop control, the arm control input is

$$\tau_k = M(q_k)\left(\ddot{q}_k^d + k_v \dot{e}_k + k_p e_k\right) + N(q_k, \dot{q}_k), \tag{4.5.3}$$

where the tracking error is $e(t)=q_d(t)-q(t)$. There are many variations of this control scheme. For instance, it is very reasonable to use multirate sampling and update $\hat{M}$ and $\hat{N}$ less frequently than the sampling frequency. That is, *the*

inner nonlinear loop can be sampled more slowly than the outer linear feedback loop. In view of the robustness properties of computed-torque control, this works quite well in practice.

Guaranteed Performance on Sampling

It is usual in robot controls to design the controller in continuous time, providing rigorous proofs of stability and error boundedness. However, when the controller is implemented, a "small" sample period is selected and the stability is left to chance and verified by simulation studies. That this "wishful thinking" approach may not be so unreasonable is suggested by the next theorem. Define $\underline{e} = [\underline{e}^T \quad \dot{e}^T]^T$.

THEOREM 4.5–1: *Suppose that the PD digital control law (4.5.3) is applied to the robot arm. Then for every L>0 there exists a T_M such that for all sampling periods T less than T_M, each error trajectory which at some time to satisfies $\|e(t_0)\| \leq L$ has $\|e(t)\| \leq L+r$ for all $t \geq t_0$ and any r>0.*

Proof:

Using the digital control law yields the error system (Table 4.4.1)

$$\ddot{e} = u - \Delta u + d, \tag{1}$$

where and $d = M^{-1}\tau_d + \Delta\ddot{q}_d + \delta$

$$\Delta = M^{-1}(q)\left[M(q) - M(q_k)\right], \; \delta = M^{-1}(q)\left[N(q,\dot{q}) - N(q_k,\dot{q}_k)\right]. \tag{2}$$

Defining $\underline{e} = [e^T \quad \dot{e}^T]^T$ *this may be written in state-space form as*

$$\dot{\underline{e}} = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix} \underline{e} + \begin{bmatrix} 0 \\ I \end{bmatrix} (u - \Delta u + d) \equiv A\underline{e} + B(u - \Delta u + d). \tag{3}$$

Applying now the outer-loop digital control

$$u = -K\underline{e}_k = -K\underline{e} + K(\underline{e} - \underline{e}_k) \tag{4}$$

yields the closed-loop system

$$\dot{\underline{e}} = (A - BK)\underline{e} + BK(\underline{e} - \underline{e}_k) + B(\Delta K\underline{e}_k + d). \tag{5}$$

The feedback K is selected so that (A-BK) is stable; hence

$$(A - BK)^T P + P(A - BK) = -Q \tag{6}$$

for some positive definite P and Q.

Selecting now the Lyapunov function

$$V(\underline{e}) = \tfrac{1}{2}\underline{e}^T P \underline{e}, \tag{7}$$

(5) and (6) reveal

$$\dot{V}(\underline{e}) = -\tfrac{1}{2}\underline{e}^T Q \underline{e} + \underline{e}^T PB\left[K(\underline{e} - \underline{e}_k) + \mathrm{d} + \Delta K \underline{e}_k\right]. \tag{8}$$

Assuming for simplicity that the disturbance τ_d is zero, note that

$$\dot{V}(\underline{e}_k) = -\tfrac{1}{2}\underline{e}_k^T Q \underline{e}_k \tag{9}$$

is negative definite at each sample time. [If τ_d is nonzero but bounded, then $V(e_k)$ is negative outside some ball, and the discussion may be modified.]

The remainder of the proof follows from a theorem of [LaSalle and Lefschetz 1961] by showing that:

1. $\dot{V}(\underline{e}) \leq 0$ *when* $\|\underline{e}\| > L$.
2. $V(\underline{e}(t_1)) < V(\underline{e}(t_2))$ *for all* $t_2 \geq t_1 \geq 0$ *when* $\|\underline{e}(t_1)\| \leq L$ *and* $\|\underline{e}(t_2)\| > L + r$.

At issue is the fact (9) and the continuity of $\dot{V}(\underline{e})$ ■

This result shows that good tracking is obtained for all sample periods less than some *maximum sample period* T_M which depends on the specific robot arm. A more refined result can show that the errors increase as the maximum desired acceleration $\ddot{\boldsymbol{q}}_{\boldsymbol{d}}$ increases, or equivalently, that smaller sample periods are required for larger desired accelerations.

We now discuss some issues in discretizing the inner nonlinear loop and then the outer linear loop.

Discretization of Inner Nonlinear Loop

There is no convenient exact way to discretize nonlinear dynamics. Given a nonlinear state-space system

$$\dot{x} = f(x, u), \tag{4.5.4}$$

Euler's approximation yields

$$x_{k+1} = x_k + Tf(x_k, u_k). \tag{4.5.5}$$

There do exist "exact" techniques for deriving discrete nonlinear robot dynamics. They rely on discretizing the robot arm dynamics in such a way that energy and momentum are conserved at each sampling instant [Neuman and Tourassis 1985]. See also [Elliott 1990]. Unfortunately, these schemes result in extremely complicated discrete dynamical equations, even for simple robot arms. It is very difficult to derive guaranteed digital control laws for them.

In this section we simply take the discretized inner nonlinear loop as given by the approximation (4.5.2).

Joint Velocity Estimates from Position Measurements

Throughout this chapter in the examples we have simulated continuous-time robot controllers assuming that both the joint positions and velocities are measured exactly. In point of fact, it is usual to measure the joint velocities using optical encoders, and then estimate the joint velocities from these position measurements. Simply computing the joint velocities using the Euler approximation

$$\dot{q}_k = (q_k - q_{k-1})/T \tag{4.5.6}$$

is virtually doomed to failure, since this high-pass filter amplifies the encoder measurement noise.

Denote the joint velocity estimates by v_k. Then a filtered derivative can be used to compute v_k from q_k using

$$v_k = \nu v_{k-1} + (q_k - q_{k-1})/T, \tag{4.5.7}$$

where ν is a design parameter. If ν is small, it corresponds to a fast pole near z=0, which provides some high-pass filtering to reject unwanted sensor noise. Example 4.5.1 will illustrate the use of this joint velocity estimation filter.

The velocity estimation filter design can be optimized for the given encoder noise statistics by using an alpha-beta tracker to reconstruct v_k [Lewis 1986a], [Lewis 1986b], [Lowe and Lewis 1991]. This is a specialized form of Kalman filter. It should be noted that the velocity estimates are not only used in the outer linear loop for computing $\dot{\boldsymbol{e}}_k$; they must be used to compute the inner nonlinear terms in (4.5.3) as well.

Discretization of Outer PD/PID Control Loop

We have seen that a useful computed-torque outer feedback loop is the PID controller. Given a continuous-time PID controller, a digital PID controller for the outer loop may be designed as follows [Lewis 1992].

A continuous PID controller that only uses joint position measurements *q(t)* is given by

$$u = -K^c(s)\, e$$
$$K^c(s) = k\left[1 + \frac{1}{T_I s} + \frac{T_D s}{1 + T_D s/N}\right], \tag{4.5.8}$$

where k is the proportional gain, T_1 is the integration time constant or "reset" time, T_D is the derivative time constant. Rather than use pure differentiation, a "filtered derivative" is used which has a pole far left in the *s*-plane at $s=-N/T_D$. The value for N is often in the range 3 to 10; it is usually fixed by the manufacturer of the controller [Åström and Wittenmark 1984]. A special case of the PID controller, of course, is the PD controller, which is therefore also covered by this discussion.

A common approximate discretization technique for converting continuous-time controllers $K^c(s)$ to digital controllers $K(z)$ is the bilinear transform (BLT), where

$$K(z) = K^c(s') \tag{4.5.9}$$

$$s' = \frac{2}{T}\frac{z-1}{z+1}. \tag{4.5.10}$$

This corresponds to approximating integration by the trapezoidal rule. Under this mapping, stable continuous systems with poles at s are mapped into stable discrete systems with poles at

$$z = \frac{1 + sT/2}{1 - sT/2}. \tag{4.5.11}$$

The finite zeros also map according to this transformation. However, the zeros at infinity in the *s*-plane map into zeros at $z=-1$.

Using the BLT to discretize (4.5.8) yields

$$K(z) = k\left[1 + \frac{T}{T_{Id}}\frac{z+1}{z-1} + \frac{T_{Dd}}{T}\frac{z-1}{z-\nu}\right] \tag{4.5.12}$$

with the discrete integral and derivative time constants

$$T_{Id} = 2T_I \tag{4.5.13}$$

$$T_{Dd} = \frac{NT}{1 + NT/2T_D} \tag{4.5.14}$$

and the derivative-filtering pole at

$$\nu = \frac{1 - NT/2T_D}{1 + NT/2T_D}. \tag{4.5.15}$$

It is easy to implement this digital outer-loop filter in terms of difference equations on a DSP. First, write $K(z)$ in terms of z^{-1}, which is the unit delay in the time domain (i.e., a delay of T seconds), as

$$K\left(z^{-1}\right) = k\left[1 + \frac{T}{T_{Id}}\frac{1 + z^{-1}}{1 - z^{-1}} + \frac{T_{Dd}}{T}\frac{1 - z^{-1}}{1 - \nu z^{-1}}\right]. \tag{4.5.16}$$

[*Note:* There is some abuse in notation in denoting (4.5.16) as $K(z^{-1})$; this we shall accept.]

Now suppose that the control input u_k is related to the tracking error as

$$u_k = -K\left(z^{-1}\right) e_k. \tag{4.5.17}$$

Then u_k may be computed from past and present values of e_k using auxiliary variables as follows:

$$v_k^I = v_{k-1}^I + (T/T_{Id})\left(e_k + e_{k-1}\right) \tag{4.5.18}$$

$$v_k^D = \nu v_{k-1}^D + (T_{Dd}/T)\left(e_k - e_{k-1}\right) \tag{4.5.19}$$

$$-u_k = k\left(e_k + v_k^I + v_k^D\right). \tag{4.5.20}$$

The variables v_k^I and v_k^D represent the integral and derivative portions of the digital PID controller, respectively. These difference equations are easily implemented in software.

Actuator Saturation and Integrator Antiwindup Compensation

Actuator saturation leading to *integrator windup* is a problem that can occur in the outer PID loop of a robot controller. In Example 4.4.4 we saw the deleterious effects of integrator windup. It is easy to implement antiwindup protection digitally [Åström and Wittenmark 1984], [Lewis 1992]. The antiwindup protection circuit would be placed into the outer linear feedback loop of a robot control system, where the integrator (controller memory) is located.

Suppose that the controller is given in transfer function form

$$R\left(z^{-1}\right) v_k = T\left(z^{-1}\right) r_k - S\left(z^{-1}\right) w_k, \tag{4.5.21}$$

where r_k is the reference command and w_k is the controller input, composed generally of the tracking error and the plant measured output. Note that z^{-1} is interpreted in the time domain as a unit delay of T seconds. The controller output v_k is passed through a hold device to generate the continuous plant control input $u(t)$.

We have assumed thus far that the desired plant control input $v_k \in \mathbb{R}^m$ computed by the controller can actually be applied to the plant. However, in practical systems the plant inputs (such as motor voltages, etc.) are limited by *maximum* and *minimum* allowable values. Thus the relation between the *desired plant input* v_k and the *actual plant input* u_k is given by the sort of behavior shown in Figure 4.5.2, where u_H and u_L represent, respectively, the maximum and minimum control effort allowed by the mechanical actuator. If there are no control limits, we may set $u_k=v_k$.

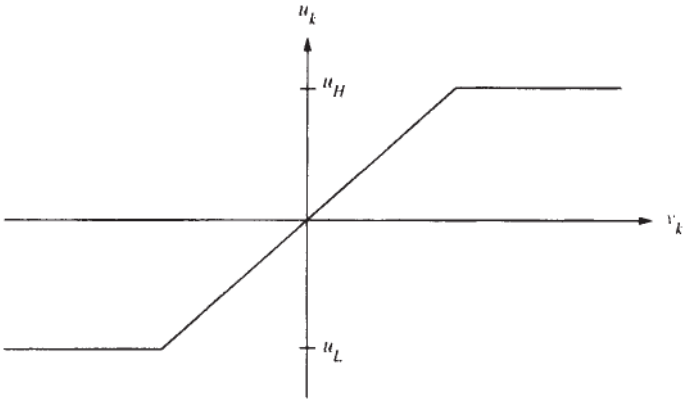


Figure 4.5.2: Actuator saturation function.

Thus, to describe the actual case in a practical control system with actuator saturation, we are forced to include *nonlinear saturation functions* in the control channels as shown in Figure 4.5.3. Consider the simple case where the controller is an integrator with input w_k and output v_k. Then all is well as long as v_k is between u_L and u_H, for in this region the plant input u_k equals v_k. However, if v_k exceeds u_H, then u_k is limited to its maximum value u_H—This in itself may not be a problem. The problem arises if w_k remains positive, for then the integrator continues to integrate and v_k may increase well beyond u_H. Then, when w_k becomes negative, it may take considerable time for v_k to decrease below u_H. In the meantime, u_k is held at u_H, giving an incorrect control input to the plant. This effect of integrator saturation is called windup. It arises because the controllers we design are generally dynamical in nature,

which means that they store information or energy. Windup occurs when $R(z)$ is not a stable polynomial.

To correct integrator windup, it is necessary to *limit the state of the controller* so that it is consistent with the saturation effects being experienced by the plant input u_k. This may be accomplished as follows. Select a desired stable observer polynomial $A_0(z)$ and add $A_0(z^{-1})u_k$ to both sides to obtain

$$A_0 u_k = Tr_k - Sw_k + (A_0 - R)\, u_k. \tag{4.5.22}$$

A regulator with antiwindup compensation is then given by

$$A_0 v_k = Tr_k - Sw_k + (A_0 - R)\, u_k. \tag{4.5.23}$$

$$u_k = \text{sat}\,(v_k)\,. \tag{4.5.24}$$

A special case occurs when $A_0(z)$ has all n of its poles at the origin. Then the regulator displays *deadbeat behavior;* after n time steps its state remains limited to an easily computed value dependent on the values of w_k and u_k (see the Problems).

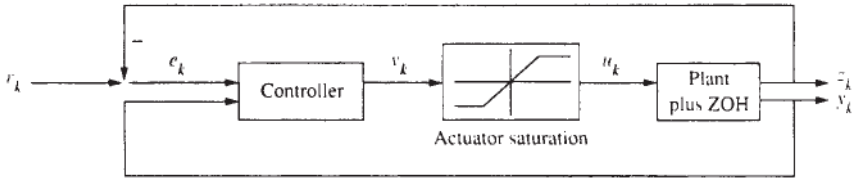


Figure 4.5.3: Control system including actuator saturation.

The next example demonstrates some aspects of digital control in robotics.

EXAMPLE 4.5–1: Simulation of Digital Robot Computed-Torque Controller

In Example 4.4.1 we simulated a continuous-time PD computed-torque (CT) controller for the two-link planar elbow arm. In this example some issues in

```
C   DIGITAL COMPUTED-TORQUE CONTROLLER FOR 2-LINK PLANAR ARM
C   SUBROUTINES FOR USE WITH TRESP

C   DIGITAL COMPUTED-TORQUE CONTROLLER SUBROUTINE
       SUBROUTINE DIG(IK,TD,x)
       REAL x(*),m1,m2,M11,M12,M22,N1,N2,kp,kv
       COMMON/CONTROL/t1, t2
C   The next line is to plot the errors and torques
```

```
      COMMON/OUTPUT/ e(2), ep(2), tp1, tp2
      COMMON/TRAJ/qd(6), qdp(6), qdpp(6)
      DATA m1,m2,a1,a2, g/1.,1.,1.,1., 9.8/
      DATA kp, kv/ 100,20/

C     COMPUTE TRACKING ERRORS
      e(1) = qd(1)  - x(1)
      e(2) = qd(2)  - x(2)
      ep(1)= qdp(1) - x(3)
      ep(2)= qdp(2) - x(4)

C     COMPUTATION OF M(q), N(q,qp)
      M11= (m1+m2)*a1**2 + m2*a2**2 + 2*m2*a1*a2*cos(x(2))
      M12= m2*a2**2 + m2*a1*a2*cos(x(2))
      M22= m2*a2**2
      N1= -m2*a1*a2*(2*x(3)*x(4) + x(4)**2) * sin(x(2))
      N1= N1 + (m1+m2)*g*a1*cos(x(1)) + m2*g*a2*cos(x(1)+x(2))
      N2= m2*a1*a2*x(3)**2*sin(x(2))  + m2*g*a2*cos(x(1)+x(2))

C     COMPUTATION OF CONTROL TORQUES

      s1= qdpp(1) + kv*ep(1) + kp*e(1)
      s2= qdpp(2) + kv*ep(2) + kp*e(2)
      t1= M11*s1 + M12*s2 + N1
      t2= M12*s1 + M22*s2 + N2
C     The next lines are to plot the torque
      tp1= t1
      tp2= t2

      RETURN
      END
C
C
C     CONTINUOUS SUBROUTINE CALLED BY RUNGE-KUTTA INTEGRATOR
      SUBROUTINE F(t,x,xp)
      REAL x(*), xp(*)
C
C     ROBOT ARM DYNAMICS
      CALL ARM(x,xp)
C
      RETURN
      END
```

Figure 4.5–4: Subroutines for use with TRESP for digital CT control.

discretizing that controller are demonstrated. There are two objectives. First, we show the effects of sampling on the CT controller. Then a practical digital controller is designed that uses only joint position measurements from an encoder, reconstructing the velocities needed for computing the control law.

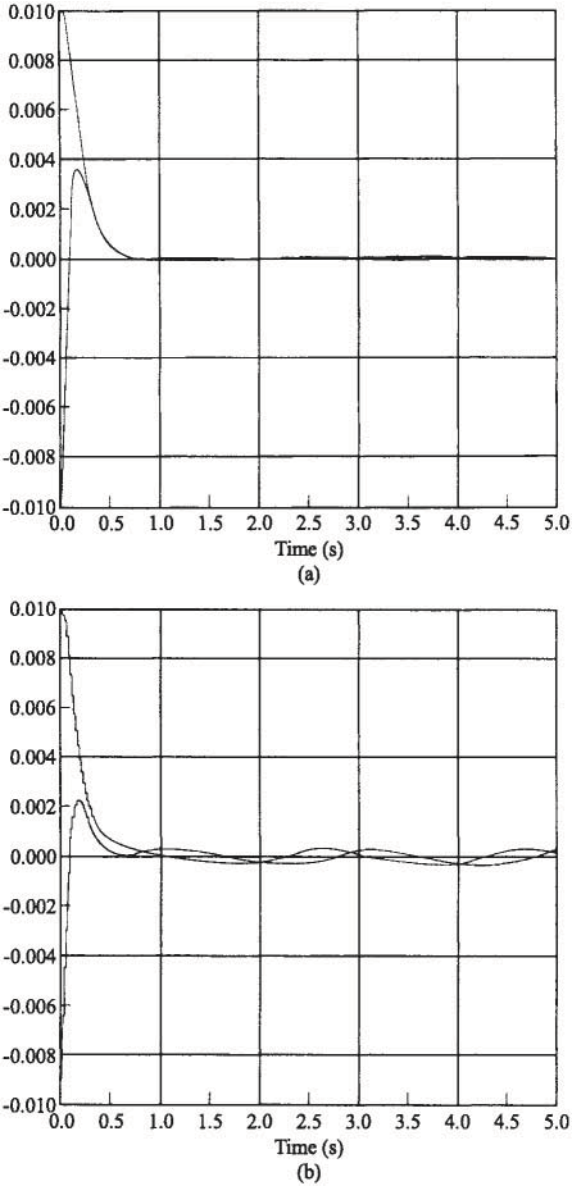
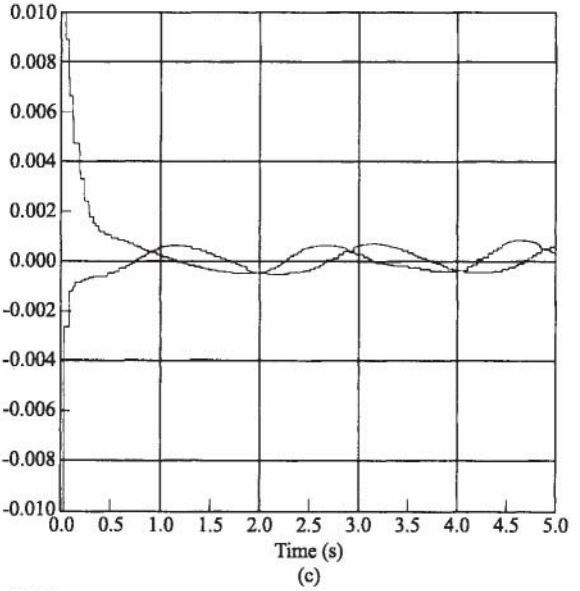


Figure 4.5.5: Joint tracking error on sampling the CT controller: (a) T=5 ms; (b) T=20 ms.



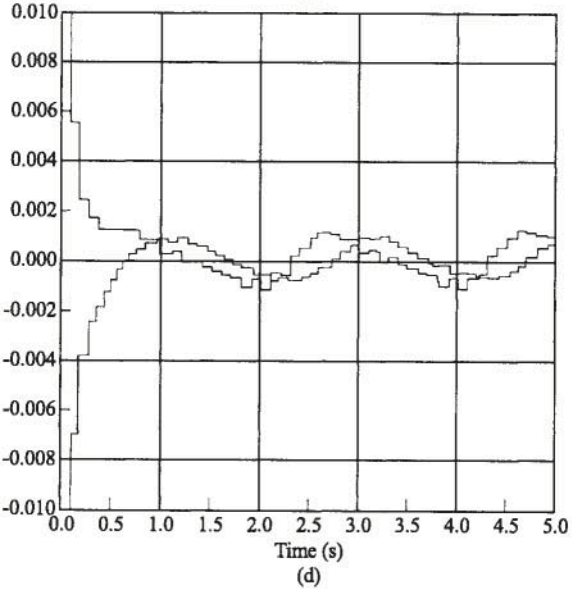


Figure 4.5.5: (*Cont.*) (c) T=50 ms; (d) T=100 ms.

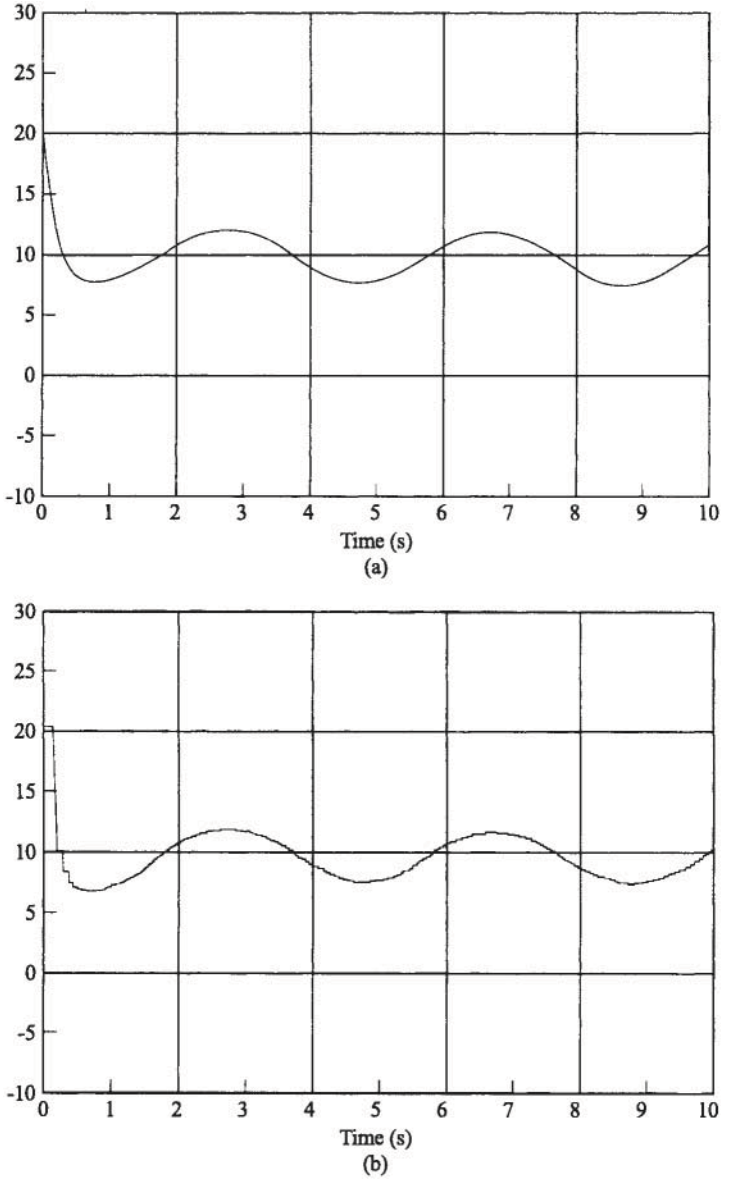


Figure 4.5.6: Link 2 torque input in the digital CT controller: (a) T=5 ms; (b) T=50 ms.

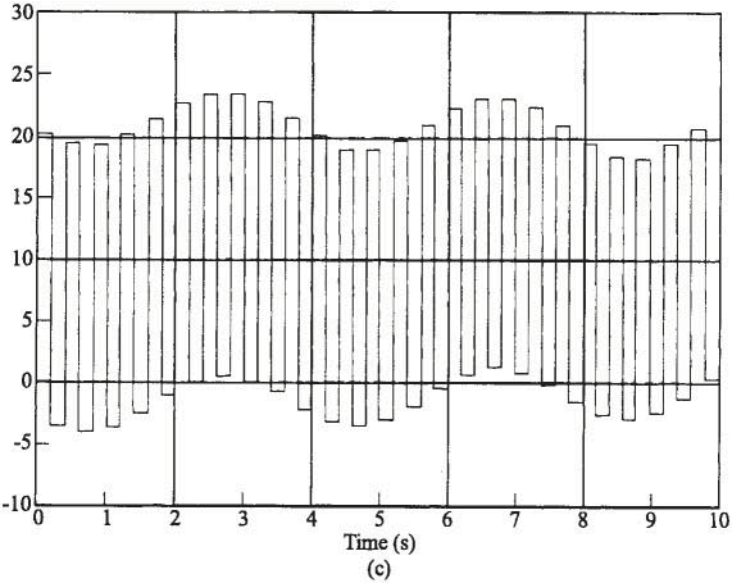


Figure 4.5.6: (*Cont.*) (c) T=100 ms.

a. Effect of Sample Period on Digital Computed-Torque Controller

In Example 4.4.1 were given the subroutines needed for use with TRESP (Appendix B) to simulate the continuous-time PD CT controller. To simulate a digitized version of PD CT control, it is necessary to remove the controller subroutine [called CTL(x) in that example] from the continuous dynamics and place it into the subroutine DIG (IK, T, x) that is called by TRESP. This is in keeping with Figure 4.5.1 and the technique given in Section 4.3 for digital controller simulation.

In Figure 4.5.4 are shown the modified subroutines DIG(IK, TD, x) and F(t, x, xp) needed for digital CT controller simulation with TRESP. Subroutines SYSINP(IT, x, t) for trajectory generation and ARM(x, xp) containing the arm dynamics are the same as in Example 4.4.1.

Running now program TRESP with different sample periods T yields the tracking error plots shown in Figure 4.5.5. The Runge-Kutta integration period was 5 ms for all plots. The graphs show that for T=5 ms the error

was very small, in fact very similar to that in Example 4.4.1. However, as the sampling period T increased the tracking performance deteriorated.

The torque input for joint 2 is depicted in Figure 4.5.6. For small T is is virtually identical to the continuous CT controller in Example 4.4.1. A very interesting phenomenon occurs when T=0.1 s. According to the plot in Figure 4.5.6(c), there is a *limit cycle*, a form of nonlinear oscillation. It is well known that sampling can induce such nonlinear effects in the closed-loop system [Lewis 1992]. According to Figure 4.5.5(d), the limit cycle is reflected in the tracking error as a periodic oscillation about $e(t)$=0. The appearance of limit cycles is closely tied to a *loss of observability* in the sampled system.

```
C     DIGITAL COMPUTED-TORQUE CONTROLLER FOR 2-LINK PLANAR ARM
C       Uses measurements of e(kT) only, no velocity measurements

C     DIGITAL COMPUTED-TORQUE CONTROLLER SUBROUTINE
         SUBROUTINE DIG(IK,T,x)
         REAL x(*),m1,m2,M11,M12,M22,N1,N2,kp,kv
         REAL xml(2), eml(2), v(2), vml(2), ev(2), evml(2), nu
         COMMON/CONTROL/tl, t2
C     The next line is to plot the errors and torques
         COMMON/OUTPUT/ e(2), ep(2), tpl, tp2
         COMMON/TRAJ/qd(6), qdp(6), qdpp(6)
         COMMON/PARAM/ nu
         DATA ml,m2,al,a2, g/1.,1.,1.,1., 9.8/
         DATA kp, kv/ 100,20/

C     COMPUTE JOINT VELOCITIES v(I), TRACKING ERRORS, VELOCITY ERROR
         DO 10 I= 1,2
         v(I)=nu*vml(I) + (x(I) - xml(I)) / T
         IF (IK.EQ.0) v(I)= 0
         e(I) = qd(I) - x(I)
         ev(I)= nu*evml(I) + (e(I) - eml(I))/T
10       IF(IK.EQ.0) ev(I)= 0

C     COMPUTATION OF M(q), N(q,qp)
         M11= (ml+m2)*al**2 + m2*a2**2 + 2*m2*al*a2*cos(x(2))
         M12= m2*a2**2 + m2*al*a2*cos(x(2))
         M22= m2*a2**2
         N1= -m2*al*a2*(2*v(1)*v(2) + v(2)**2) * sin(x(2))
         N1= N1 + (ml+m2)*g*al*cos(x(1)) + m2*g*a2*cos(x(1)+x(2))
         N2=m2*al*a2*v(1)**2*sin(x(2)) + m2*g*a2*cos(x(1)+x(2))

C     COMPUTATION OF CONTROL TORQUES

         sl= qdpp(1) + kp*e(1) + kv*ev(1)
         s2= qdpp(2) + kp*e(2) + kv*ev(2)
         tl= M11*sl + M12*s2 + N1
```

```
          t2= M12*s1 + M22*s2 + N2
C        Store signals for next sample
          DO 20 I= 1,2
          xm1(I)= x(I)
          vm1(I)= v(I)
          em1(I)= e(I)
20        evm1(I)= ev(I)

C     The next lines are to plot the torque
          tp1= t1
          tp2= t2

          RETURN
          END
```

Figure 4.5–7: Digital control subroutine that uses only joint position measurements.

b. Digital Controller with Only Position Measurements

In part a we assumed that joint positions and velocities are both available as measurements. In practical situations, only the joint positions are available, often from optical encoder measurements. Therefore, here we should like to design a realistic digital CT controller that reconstructs the velocities.

The subroutines in Figure 4.5.7 uses only position measurements, estimating the joint velocities using the derivative filter (4.5.7). The resultant tracking error is shown in Figure 4.5.8. The performance is comparable to the controller in part a that used velocity measurements, with the exception of a larger initial error transient. The value of the filter pole ν was taken as 0.1.

Some implementation details are worthy of note. First, it is very important how the digital controller is initialized. Note the code lines that zero the velocity estimates in the first iteration (IK=0). It is a good exercise to repeat this simulation deleting these lines (see the Problems).

Second, it might be thought that a reasonable procedure for finding the velocity error $\dot{\boldsymbol{e}}_k = \dot{\boldsymbol{e}}(kT)$ is to find an estimate v_k of the joint velocity $\dot{\boldsymbol{q}}_k$ and then use

$$\dot{e}_k = \dot{q}_k^d - v_k. \tag{1}$$

There are two disadvantages to this. First, it requires the storage in memory of the desired velocity $\dot{q}_k^d$ as well as the desired trajectory q_k^d Second, it does not work.

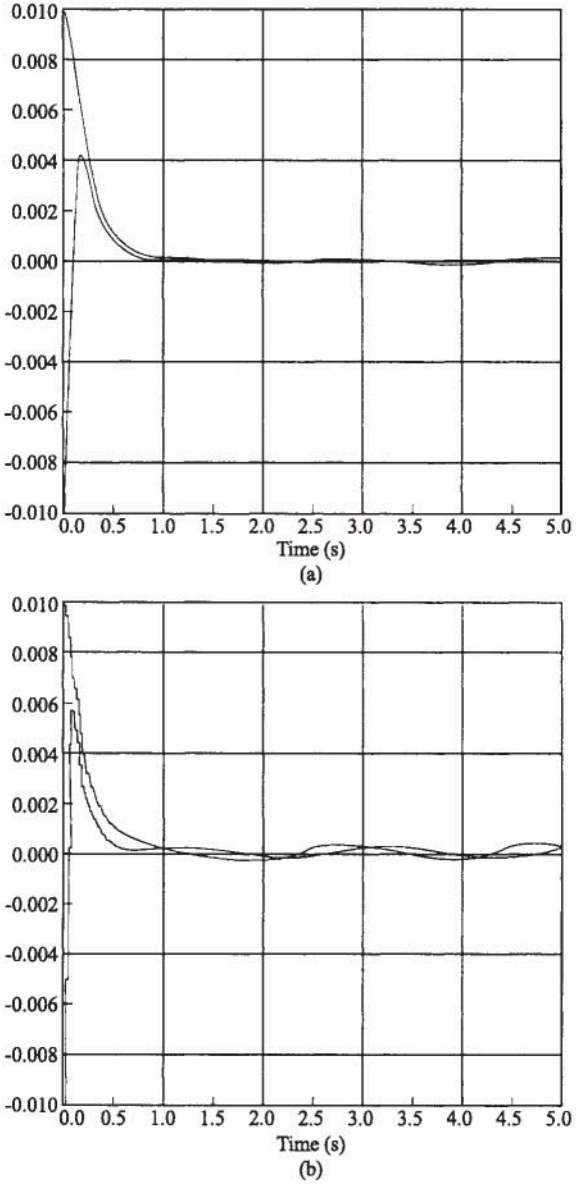


Figure 4.5.8: Tracking error for digital controller using no velocity measurements: (a) T=5 ms; (b) T=20 ms.

Instead, it is necessary to use *two derivative filters*, one for estimating joint velocities $\dot{q}_k$ and one for providing an estimate e_k^v for the velocity error $\dot{e}_k$. A simulation that attempts to use 1 makes the point quite well.

Note that the velocity estimates are also used instead of $\dot{q}_k$ in computing the nonlinear terms $N(q_k, \dot{q}_k)$.

■

EXAMPLE 4.5–2: Digital PI Controller with Anti-windup Compensation

A general digital PI controller with sampling period of T seconds is given by

$$u_k = k\left[1 + \frac{T}{T_I}\frac{1}{z-1}\right] e_k. \tag{1}$$

We have sampled the continuous PI controller by the modified matched-polezero (MPZ) method [Lewis 1992] to obtain a delay of T seconds in the integrator to allow for computation time. The proportional gain is k and the reset time is T_I; both are fixed in the design stage. The tracking error is e_k.

Multiply by $(1\text{-}z^{-1})$ to write

$$\left(1 - z^{-1}\right) u_k = k\left[\left(1 - z^{-1}\right) + Tz^{-1}/T_I\right] e_k, \tag{2}$$

which is in the transfer function form (4.5.21). The corresponding difference equation form for implementation is

$$u_k = u_{k-1} + ke_k + k\left(-1 + T/T_I\right) e_{k-1}. \tag{3}$$

This controller will experience windup problems since the autoregressive polynomial R=1- z^{-1} has a root at z=1, making it marginally stable. Thus, when u_k is limited, the integrator will continue to integrate, "winding up" beyond the saturation level.

a. Antiwindup Compensation

To correct this problem, select an observer polynomial of

$$A_0\left(z^{-1}\right) = 1 - \alpha z^{-1}, \tag{4}$$

which has a pole at some desirable location $|\alpha|<1$. The design parameter α may be selected by simulation studies. Then the controller with antiwindup protection (4.5.23), (4.5.24) is given by

$$\left(1-\alpha z^{-1}\right) v_k = k\left[1+\left(-1+\frac{T}{T_I}\right) z^{-1}\right] e_k + (1-\alpha)\, z^{-1} u_k \tag{5}$$

$$u_k = \text{sat}\,(v_k)\,. \tag{6}$$

The corresponding difference equations for implementation are

$$v_k = k e_k + \alpha v_{k-1} + k\,(-1+T/T_I)\, e_{k+1} + (1-\alpha)\, u_{k+1} \tag{7}$$

$$u_k = \text{sat}\,(v_k)\,. \tag{8}$$

A few lines of FORTRAN code implementing this digital controller are given in Figure 4.5.9. This subroutine may be used as the control update routine DIG for the digital simulation driver program TRESP in Appendix B.

```
DIGITAL PI CONTROLLER WITH ANTIWINDUP COMPENSATION

   SUBROUTINE DIG(IK,T,x)
   REAL x(*), k
   COMMON/CONTROL/ u
   COMMON/OUTPUT/ z
   DATA k,AL,TI,ULOW,UHIGH/ 3.318. 0.9, 1., -1.5 , 1.5/
   DATA r/1./

   v= AL*v + k*(-1 + T/TI)*e + (1-AL)*u
   e= r - z
   v= k*e + v
   u= AMAX1(ULOW,v)
   u= AMIN1(UHIGH,u)

   RETURN
   END
```

Figure 4.5–9: FORTRAN code implementing PI controller with antiwindup compensation.

If α=1 we obtain the special case (2), which is called the *position form* and has no antiwindup compensation. If α=0, we obtain the *deadbeat antiwindup compensation*

$$v_k = k\left[1 + \left(-1 + \frac{T}{T_I}\right) z^{-1}\right] e_k + u_{k-1}, \tag{9}$$

with corresponding difference equation implementation

$$v_k = u_{k-1} + ke_k + k\left(-1 + T/T_I\right) e_{k-1}. \tag{10}$$

If u_k is not in saturation, this amounts to updating the plant control by adding the second and third terms on the right-hand side of (10) to u_{k-1}. These terms are, therefore, nothing but u_k-u_{k-1}. The compensator with α=0 is thus called the *velocity form* of the PI controller.

b. Digital Control of DC Motor

Consider the simplified model for a do motor given by

$$\dot{\omega} = -a\omega + bu, \tag{11}$$

with ω the angular velocity. A motor speed controller has the form

$$u = -\left[k_1 + k_2/s\right] e, \tag{12}$$

where e=r-w is the tracking error, with r the desired command angular velocity. Taking a=1 and b=1, with k_1=k_2=-3.318, we obtain poles at s=-1, -3.318. The slower pole is canceled by a zero, so that the step response is fast, having only a mode like $e^{-3.318t}$.

Writing the PI controller as

$$u = k\left[1 + 1/T_I s\right] e, \tag{13}$$

we see that

$$\begin{aligned} k &= -k_1 = 3.318 \\ T_I &= k_1/k_2 = 1\,\text{s}. \end{aligned} \tag{14}$$

The digital controller obtained using the modified MPZ is given by (1).

The time constant of the closed-loop system is $\tau = 1/3.318 = 0.3$ s, so that a sampling period of T=0.05 s is reasonable. The sampling period should be about one-tenth the time constant.

Program TRESP in Appendix B was used to obtain the response shown in Figure 4.5.10(a). No saturation limits were imposed on u_k.

Next, a saturation limit of u_H=1.5 V was imposed on the control u_k. No antiwindup compensation was used (i.e., α=1 in Figure 4.5.9). The resulting behavior shown in Figure 4.5.10(b) displays an unacceptable overshoot.

Figure 4.5.10(c) shows that the overshoot problem is easily corrected using α=0.9 in the digital PI controller with antiwindup protection in Figure 4.5.9. In this example, as a decreases the step response slows down. The value of α=0.9 was selected after several simulation runs with different values of α.

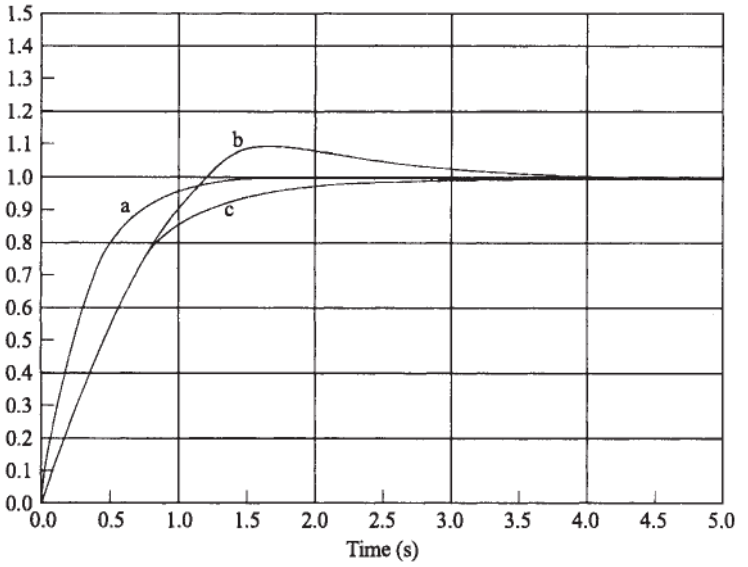


Figure 4.5.10: Angular velocity step responses using digital controller: (a) digital PI controller with no saturation limits; (b) digital PI controller with saturation limit u_H=1.5V and no antiwindup compensation; (c) digital PI controller with saturation limit u_H=1.5V and antiwindup compensation with α=0.9.

c. Antiwindup Compensation in Robotics

To implement the antiwindup compensation, it is necessary to know the maximum and minimum values u_H and u_L of the integrator output. Unfortunately, in a robot arm the motor torques are limited. These torque

limits must be mapped using the feedback linearization input transformation to determine the limits on the integrator outputs see [Bobrow et al. 1983]. Thus the saturation limits needed by the antiwindup compensator are functions of the joint position q, the desired acceleration $\ddot{q}_d$ and so on. This issue is explored in the problems.

■

4.6 Optimal Outer-Loop Design

In Section 4.4 we discussed computed-torque control, showing how to select the inner control loop using exact techniques involving the inverse manipulator dynamics, as well as by a variety of approximate means. We also discussed several schemes for designing the outer linear feedback (tracking) loop. The results of our discussions are summarized in Table 4.4.1. In this section we intend to present a modern control optimal technique for selecting the outer feedback loop. Modern optimal design yields improved robustness in the presence of disturbances and unmodeled dynamics.

Several papers have dealt with "optimal" or "suboptimal" control of robot manipulators [Vukobratovic and Stokic 1983], [Lee et al. 1983], [Luo and Saridis 1985], [Johansson 1990]. Although they are not all based on a computed-torque-like approach, we would like here to present the flavor of this work by using optimal techniques to design the computed-torque outer feedback loop.

Linear Quadratic Optimal Control

First, it is necessary to review modern linear-quadratic (LQ) design. Suppose that we are given the linear time-invariant system in state-space form

$$\dot{x} = Ax + Bu. \tag{4.6.1}$$

with $x(t) \in \mathbf{R}^n$, $u(t) \in \mathbf{R}^m$. It is desired to compute the state-feedback gain in

$$u = -Kx, \tag{4.6.2}$$

so that the closed-loop system

$$\dot{x} = (A - BK)\, x \tag{4.6.3}$$

is asymptotically stable. Moreover, we do not want to use too much control energy to stabilize the system, since in many modern systems (e.g., automobile, spacecraft), fuel or energy is limited.

This is a complex multivariable design problem, for the feedback gain matrix K is of dimension $m \times n$. A classical controls approach might involve, for instance, performing mn root locus designs to close the feedback loops one at a time. On the other hand, a solution that *guarantees stability* can be found using modern controls techniques simply by solving some *standard matrix design equations*. This modern approach *closes all the feedback loops simultaneously and guarantees a good gain and phase margin.*

The feedback matrix is found using modern control theory as follows. First, define a *quadratic performance index* (PI) of the form

$$J = \tfrac{1}{2} \int_0^\infty \left(x^T Q x + u^T R u \right) dt, \tag{4.6.4}$$

where Q is a symmetric positive semidefinite $n \times n$ matrix (denoted $Q \geq 0$) and R is a symmetric positive definite $m \times m$ matrix ($R > 0$). That is, all eigenvalues of R are greater than zero and those of Q are greater than or equal to zero. Q is called the *state-weighting matrix* and R the *control-weighting matrix*. These matrices are *design parameters* that are selected by the engineer depending, for instance, on the desired form of the closed-loop time responses.

The *optimal LQ feedback gain* K is the one that minimizes the PI J. The motivation follows. The quadratic terms $x^T Q x$ and $u^T R u$ in the PI are generalized energy functions (e.g., the energy in a capacitor is $\frac{1}{2}Cv^2$, the kinetic energy of motion is $\frac{1}{2}mv^2$). Suppose, then, that J is minimized in the closed-loop system (4.6.3). This means that the infinite integral of $[x^T(t)Qx(t)+u^T(t)Ru(t)]$ is finite, so that this function of time goes to zero as t becomes large. However,

$$x^T(t)\, Q x(t) = \left\| \sqrt{Q}\, x(t) \right\|^2$$

$$u^T(t)\, R u(t) = \left\| \sqrt{R}\, u(t) \right\|^2$$

with the square root of a matrix defined as $M = \sqrt{M^T}\sqrt{M}$. Since these norms vanish with t and $|R| \neq 0$, the functions $y(t) = \sqrt{Q}x(t)$ and $u(t)$ both go to zero. Under the assumption that $(A, \sqrt{Q})$ is observable [Kailath 1980], $x(t)$ goes to zero if $y(t)$ does.

Therefore, *the optimal gain K guarantees that all signals go to zero with time in the closed-loop system (4.6.3).* That is, K stabilizes $(A\text{-}BK)$.

The determination of the optimal K is easy and is a standard result in modern control theory (see, e.g., [Lewis 1986a], [Lewis 1986b]). The optimal feedback gain is simply found by solving the *matrix design equations*

$$K = R^{-1}B^T P \tag{4.6.5}$$

$$A^T P + PA + Q - PBR^{-1}B^T P = 0, \tag{4.6.6}$$

where P is a symmetric $n \times n$ auxiliary design matrix on which the optimal gain depends. The second of these is a nonlinear matrix quadratic equation known as the *Riccati equation;* it is easy to solve this equation for the auxiliary matrix P using standard routines in, for instance, MATLAB, [MATRIX$_x$ 1989], and other software design packages.

The next result is of prime importance in modern control theory and formalizes the stability discussion just given.

THEOREM 4.6–1: *Let* $(A, \sqrt{Q})$ *be observable and (A, B) be controllable. Then:*

(a) There is a unique positive definite solution P to the Riccati equation.
(b) The closed-loop system (A-BK) is asymptotically stable.
(c) The closed-loop system has an infinite gain margin and 60° *of phase margin.*

■

Controllability was discussed in Chapter 1. Observability means roughly speaking that all the system modes have an independent effect in the PI, so that if J is bounded, all the modes independently go to zero with t. To verify these properties is easy. The system is controllable if the controllability matrix

$$U = \begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix} \tag{4.6.7}$$

has full rank n. The system (A, C) is observable if the observability matrix

$$V = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{4.6.8}$$

has full rank n. MATLAB, for instance, provides routines for these tests. Therefore, the state-weighting matrix Q may be chosen to satisfy the observability requirement.

The theorem makes this modern design approach very powerful. No matter how many inputs and states, a stabilizing feedback gain can always be found under the hypotheses that stabilizes the system. The gain is found by closing

all the *nm* feedback loops simultaneously by solving the matrix design equations.

Linear Quadratic Computed-Torque Design

Now we apply these results to the control of the robot manipulator dynamics

$$M(q)\ddot{q} + N(q,\dot{q}) = \tau. \tag{4.6.9}$$

According to Section 4.4, the computed-torque control law

$$\tau = M(\ddot{q}_d - u) + N \tag{4.6.10}$$

yields the error system

$$\frac{d}{dt}\begin{bmatrix} e \\ \dot{e} \end{bmatrix} = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}\begin{bmatrix} e \\ \dot{e} \end{bmatrix} + \begin{bmatrix} 0 \\ I \end{bmatrix} u, \tag{4.6.11}$$

which we may write as

$$\dot{x} = Ax + Bu, \tag{4.6.12}$$

with the state defined as

$$x = \begin{bmatrix} e \\ \dot{e} \end{bmatrix}. \tag{4.6.13}$$

Now, select the outer-loop PD feedback

$$u = -Kx = -\begin{bmatrix} K_p & K_v \end{bmatrix} x = -K_p e - K_v \dot{e}. \tag{4.6.14}$$

To find a stabilizing gain K, select the design parameter Q in the PI as

$$Q = \text{diag}\{Q_p, Q_v\} \quad \text{with } Q_p, Q_v \in \mathrm{R}^{n\times n}$$

so that the position and velocity errors are independently weighted. Then, due to the simple form of the A and B matrices, which represent n de-coupled Newton's law (i.e., double integrator) systems, the solution of the Riccati equation is easily found (see the Problems). Using this solution in (4.6.5) yields the formula for the optimal stabilizing gains

$$K_p = \sqrt{Q_p R^{-1}}, \quad K_v = \sqrt{2K_p + Q_v R^{-1}}. \tag{4.6.15}$$

This LQ approach reveals the relation between the PD gains and some design parameters Q and R that determine the total energy in the closed-loop system.

Note particularly that the relative magnitudes of $x(t)$ and $u(t)$ in the closed-loop system can be traded off. Indeed, if R is relatively larger than Q_p and Q_v, the control effort in the PI (4.6.4) is weighted more heavily that the state. Then the optimal control will attempt to keep $u(t)$ smaller by selecting smaller control gains; thus the response time will increase. On the other hand, selecting a smaller R will increase the PD gains and make the error vanish more quickly.

If Q_p, Q_v, and R are diagonal, so then are the PD gains K_p, K_v. The LQ approach with nondiagonal Q_p, Q_v, and R affords the possibility of outer feedback loops that are coupled between the joints, which can sometimes improve performance. Another important feature of LQ design is the guaranteed robustness mentioned in the theorem. This can be very useful in approximate computed-torque design where

$$\tau_c = \hat{M}\left(\ddot{q}_d - u\right) + \hat{N} \tag{4.6.16}$$

and $\hat{M}$ and $\hat{N}$ can be simplified versions of $M(q)$ and $N(q, \dot{q})$. The performance of such a controller with an LQ-design outer loop can be expected to surpass that of a controller designed using arbitrary choices for K_p and K_v. This robust aspect of LQ design is explored in the problems.

It is important to note that this LQ design results in minimum closed-loop energy in terms of $e(t)$, $\dot{e}(t)$, and $u(t)$. However, the actual control input into the robot arm is

$$\tau = M\left(\ddot{q}_d + K_v\dot{e} + K_p e\right) + N. \tag{4.6.17}$$

Although the energy in $\tau(t)$ is not minimized using this approach, we can use some norm inequalities to write

$$\|\tau\| \leq \|M(q)\| \cdot \|\ddot{q}_d\| + \|M(q)\| \cdot \|u(t)\| + \|N(q, \dot{q})\|, \tag{4.6.18}$$

so that keeping small $\|u(t)\|$ might be expected to make $\|\tau(t)\|$ smaller. A more formal statement can be made taking into account the bounds on $\|M(q)\|$ and $\|N(q, \dot{q})\|$ given in Table 3.3.1.

Since the energy in is not formally minimized in this approach, it is considered as a *suboptimal approach* with respect to the actual arm dynamics, although with respect to the error system and $u(t)$ it is optimal. An optimal control approach that weights $e(t)$ $\tau(t)$ and in the PI is given in [Johansson 1990].

We have derived an LQ controller using a computed-torque (i.e., feedback linearization) approach. An alternative approach that yields the same Riccati-equation-based design is to employ the full nonlinear arm dynamics and find an approximate (i.e., time-invariant) solution to the Hamilton-Jacobi-Bellman equation [Luo and Saridis 1985]; [Luo et al. 1986].

4.7 Cartesian Control

We have seen how to make a robot manipulator track a desired joint space trajectory $q_d(t)$. However, in any practical application the desired trajectories of a robot arm are given in the workspace or Cartesian coordinates. An important series of papers dealt with *resolved motion manipulator control* [Whitney 1969]; [Luh et al. 1980], [Wu and Paul 1982]. There the joint motions were resolved into the Cartesian coordinates, where the control objectives are specified. The result is that an operator can use a joystick to specify Cartesian motion (e.g., for a prosthetic device), with the arm following the specified motion. Older teleoperator devices used joysticks that directly controlled the motion of the actuators, resulting in long training times and very awkward manipulability.

There are several approaches to Cartesian robot control. For instance, one might:

1. Use the Cartesian dynamics in Section 3.5 for controls design (see the Problems).
2. Convert the desired Cartesian trajectory $y_d(t)$ to a joint-space trajectory $q_d(t)$ using the inverse kinematics. Then use the joint-space computed-torque control schemes in Table 4.4.1.
3. Use Cartesian computed-torque control.

Let us discuss the last of these.

Cartesian Computed-Torque Control

This approach begins with the joint space dynamics

$$M(q)\ddot{q} + N(q,\dot{q}) + \tau_d = \tau. \tag{4.7.1}$$

In Section 3.4 we discussed a general feedback-linearization approach for linearizing the arm dynamics with respect to a general output. In this section the output we are interested in is the *Cartesian error*

$$e_y(t) = y_d(t) - y(t), \tag{4.7.2}$$

with $y_d(t)$ the desired Cartesian trajectory and $y(t)$ the end-effector Cartesian position.

The problems associated with specifying the Cartesian position of the end effector are covered in Appendix A. There we see that $y(t)$ is not necessarily a 6-vector, but could in fact be the 4×4 arm T matrix. Then $y_d(t)$ is a 4×4 matrix given by

$$y_d(t) = T_d(t) = \begin{bmatrix} n_d(t) & o_d(t) & a_d(t) & p_d(t) \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.7.3}$$

containing the desired orientation *($n_d(t)$, $o_d(t)$, $a_d(t)$)* and position $p_d(t)$ of the end effector with respect to base coordinates. On the other hand, $y(t)$ could be specified (nonuniquely) using Euler angles as a 6-vector, or using quaternions as a 7-vector, or using the encoded tool configuration vector which gives $y(t) \in R^6$.

Although there are problems with specifying $y(t)$ as a 6-vector, the Cartesian error is easily specified (see the next subsection) as the 6-vector

$$e_y = \begin{bmatrix} e_p \\ e_o \end{bmatrix} \tag{4.7.4}$$

with $e_p(t)$ the position error and $e_o(t)$ the orientation error. Thus equation (4.7.2) is generally valid only as a loose notational convenience.

Let us assume that

$$y = h(q) \tag{4.7.5}$$

with $h(q)$ the transformation from $q(t)$ to $y(t)$, which is a modification of the kinematics transformation, depending on the form decided on for $y(t)$. Then the associated Jacobian is $J = \P h / \P q$ and

$$\dot{y} = J\dot{q}. \tag{4.7.6}$$

Now, the approach of Section 3.4, or a small modification of the derivation in Section 4.4, shows that the computed-torque control relative to $e_y(t)$ is given by

$$\tau = MJ^{-1}\left(\ddot{y}_d - \dot{J}\dot{q} - u\right) + N, \tag{4.7.7}$$

which results in the error system

$$\ddot{e}_y = u + w \tag{4.7.8}$$

with the disturbance

$$w = JM^{-1}\tau_d. \tag{4.7.9}$$

We call (4.7.7) the *Cartesian computed-torque control law*.

The outer-loop control $u(t)$ may be selected using any of the techniques already mentioned for joint-space computed-torque control (see Table 4.4.1). For PD control, for instance, the complete control law is

$$\tau = MJ^{-1}\left(\ddot{y}_d - \dot{J}\dot{q} + K_v\dot{e}_y + K_p e_y\right) + N. \tag{4.7.10}$$

A disadvantage with Cartesian computed-torque control is the necessity to compute the inverse Jacobian. To avoid inverting the Jacobian at each sample period, we might propose the *approximate Cartesian computedtorque controller*

$$\tau_c = \hat{C}\left(\ddot{y}_d - \dot{J}\dot{q} - u\right) + \hat{N}, \tag{4.7.11}$$

where $\hat{C}$ and $\hat{N}$ are approximations to MJ^{-1} and N, respectively. The error system for this control law is not difficult to compute (cf. the joint space approximation in Table 4.4.1 and see the Problems).

A PD outer feedback loop yields

$$e_o(t) = k(t)\sin\varphi(t), \quad -\pi/2 \le \varphi(t) \le \pi/2, \tag{4.7.12}$$

A special case of this control law is obtained by setting $\hat{C}=I$, $\hat{N}=-\ddot{y}_d + \dot{J}\dot{q} + G(q)$ which yields the *Cartesian PD-gravity* controller

$$\tau_c = K_v\dot{e}_y + K_p e_y + G(q). \tag{4.7.13}$$

The robustness properties of computed-torque control make this a successful control law for many applications.

Simulations like those presented in this section could be carried out for Cartesian computed-torque control. The basic principles would be the same as for joint space computed-torque control (see the Problems).

Cartesian Error Computation

The actual Cartesian position may be computed from the measured joint variables using the arm kinematics in terms of the arm T matrix

$$y(t) = T(t) = \begin{bmatrix} n(t) & o(t) & a(t) & p(t) \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.7.14}$$

and the desired Cartesian position may likewise be expressed as (4.7.3). Then a Cartesian position error and velocity error suitable for computedtorque control may be computed as follows [Luh et al. 1980], [Wu and Paul 1982]. Define

$$\dot{y} = \begin{bmatrix} v \\ \omega \end{bmatrix}, \quad \dot{y}_d = \begin{bmatrix} v_d \\ \omega_d \end{bmatrix} \tag{4.7.15}$$

with $v(t)$, $v_d(t)$ the actual and desired linear velocity, and $\omega(t)$, $\omega_d(t)$ the actual and desired angular velocities. Then

$$\dot{e}_y = \dot{y}_d - \dot{y} \tag{4.7.16}$$

is easy to compute, since $\dot{\boldsymbol{y}}(t)$ may be determined from the measured joint variables using (4.7.6).

The linear position error is simply given by

$$e_p = p_d - p. \tag{4.7.17}$$

An orientation error $e_o(t)$ suitable for feedback purposes is more difficult to obtain, but may be defined as follows.

Denote the rotation transformation portions of (4.7.14) and (4.7.3), respectively, as $R(t)$, $R_d(t)$. The orientation error can be expressed in terms of a rotation of $\phi(t)$ rads about an Euler axis of $k(t)$ that takes $R(t)$ into $R_d(t)$ (Appendix A). In fact, one may define the 3-vector

$$e_o(t) = k(t)\sin\varphi(t), \quad -\pi/2 \le \varphi(t) \le \pi/2, \tag{4.7.18}$$

where $e_o(t)$ may be assumed small. With this definition, it can be shown that $e_o(t)$ is found from $T(t)$ and $T_d(t)$ using

$$e_o = \tfrac{1}{2}(n \times n_d + o \times o_d + a \times a_d). \tag{4.7.19}$$

The overall Cartesian error is now given by (4.7.4). Unfortunately, with this definition of e_o, it happens that e_y is not the derivative of e_y; however, the control law (4.7.10) still yields suitable results. Alternative definitions of $e_y(t)$ and $\dot{\boldsymbol{e}}_y(t)$ are given in [Wu and Paul 1982]; they are closely tied to the cross-product matrix O, in Appendix A and require the selection of a sampling period T.

4.8 Summary

In this chapter we showed how to generate smooth trajectories defining robot end-effector motion that passes through a set of specified points. Then we covered the important class of computed-torque controllers, which subsumes many types of robot control algorithms. Both classical and modern control algorithms are described by this class, so that computed torque provides a bridge between older and more modern algorithms for motion control.

As special types of computed-torque algorithms, we mentioned PD control, PID control, PD-plus-gravity, classical joint control, and digital control. Most robot control algorithms are implemented digitally, and computedtorque provides a rigorous framework for analyzing the effects of digitization and the size of the sampling period. This is approached by considering digital

control as an approximate computed-torque law and studying the error system (cf. subsequent chapters).

We showed some aspects of modern linear quadratic outer-loop design, and concluded with a discussion of Cartesian control.

REFERENCES

[Arimoto and Miyazaki 1984] Arimoto, S., and F Miyazaki, "Stability and robustness of PID feedback control for robot manipulators of sensory capability," *Proc. First Int. Symp.,* pp. 783–799, MIT, 1984.

[Åström and Wittenmark 1984] Åström, K.J., and B.Wittenmark, *Computer Controlled Systems.* Englewood Cliffs, NJ: Prentice Hall, 1984.

[Bobrow et al. 1983] Bobrow, J.E., S.Dubowsky, and J.S.Gibson, "On the optimal control of robotic manipulators with actuator constraints," *Proc. Am. Control Conf,* pp. 782–787, June 1983.

[Chen 1989] Chen, Y.-C., "On the structure of the time-optimal controls for robotic manipulators," *IEEE Trans. Autom. Control,* vol. 34, no. 1, pp. 115–116, Jan. 1989.

[Dawson 1990] Dawson, D.M., "Uncertainties in the control of robot manipulators," Ph.D. thesis, School of Electrical Engineering, Georgia Institute of Technology, Mar. 1990.

[Elliott 1990] Elliott, D.L., "Discrete-time systems on manifolds," *Proc. IEEE Conf Decision Control,* pp. 1908–1909, Dec. 1990.

[Franklin et al. 1986] Franklin, G.F:, J.D.Powell, and A.Emami-Naeini, *Feedback Control of Dynamic Systems.* Reading, MA: Addison-Wesley, 1986.

[Geering 1986] Geering, H.P., L.Guzzella, S.A.R.Hepner, and C.H. Onder, "Time-optimal motions of robots in assembly tasks," *IEEE Trans. Autom. Control,* vol. AC-31, no. 6, pp. 512–518, June 1986.

[Gilbert and Ha 1984] Gilbert, E.G., and I.J.Ha, "An approach to nonlinear feedback control with applications to robotics," *IEEE Trans. Syst. Man Cybern.,* vol. SMC-14, no. 6, pp. s 879–884, Nov./Dec. 1984.

[Gourdeau and Schwartz 1989] Gourdeau, R., and H.M.Schwartz, "Optimal control of a robot manipulator using a weighted time-energy cost function," *Proc. IEEE Conf. Decision Control,* pp. 1628–1631, Dec. 1989.

[Hunt et al. 1983] Hunt, L.R., R.Su, and G.Meyer, "Global transformations of nonlinear systems," *IEEE Trans. Autom. Control,* vol. AC-28, no. 1, pp. 24–31, Jan. 1983.

[IMSL] IMSL, *Library Contents Document,* 8th ed. Houston, TX: International Mathematical and Statistical Libraries.

[Jayasuriya and Suh 1985] Jayasuriya, S., and M.-S.Suh, "Sub-optimal control strategies for manipulators with actuator constraints: the near minimum-time problem," *Proc. Am. Control Conf.,* pp. 61–62, June 1985.

[Johansson 1990] Johansson, R., "Quadratic optimization of motion coordination and control," *IEEE Trans. Autom. Control,* vol. 35, no. 11, pp. 1197–1208, Nov. 1990.

[Kahn and Roth 1971] Kahn, M.E., and B.Roth, "The near-minimum-time control of open-loop articulated kinematic chains," *Trans. ASMEJ. Dyn. Syst. Meas. Control,* pp. 164–172, Sept. 1971.

[Kailath 1980] Kailath, T. *Linear Systems.* Englewood Cliffs, NJ: Prentice Hall, 1980.

[Kim and Shin 1985] Kim, B.K., and K.G.Shin, "Suboptimal control of industrial manipulators with a weighted minimum time-fuel criterion," *IEEE Trans. Autom. Control,* vol. AC30, no. 1, pp. 1–10, Jan. 1985.

[LaSalle and Lefschetz 1961] LaSalle, J., and S.Lefschetz, *Stability by Liapunov's Direct Method.* New York: Academic Press, 1961.

[Lee et al. 1983] Lee, C.S.G., and M.H.Chen, "A suboptimal control design for mechanical manipulators," *Proc. Am. Control Conf.,* pp. 1056–1061, June 1983.

[Lewis 1986a] Lewis, F.L., *Optimal Control.* New York: Wiley, 1986 (a).

[Lewis 1986b] Lewis, F.L., *Optimal Estimation.* New York: Wiley, 1986 (b).

[Lewis 1992] Lewis, F.L., *Applied Optimal Control and Estimation.* Englewood Cliffs, NJ: Prentice Hall, 1992.

[LINPACK] LINPACK, *User's Guide,* J.J.Dongarra, C.B.Moler, J.R. Bunch, and G.W.Stewart. Philadelphia: SIAM Press, 1979.

[Lowe and Lewis 1991] Lowe, J.A., and F.L.Lewis, "Digital signal processor implementation of a Kalman filter for disk drive head-positioning mechanism," in *Microprocessors in Robotic and Manufacturing Systems,* S.Tzafestas ed., pp. 369–383, 1991.

[Luh et al. 1980] Luh, J.Y.S., M.W.Walker, and R.P.C.Paul, "Resolved-acceleration control of mechanical manipulators," *IEEE Trans. Autom. Control,* vol. AC-25, no. 3, pp. 195–200, June 1980.

[Luo and Saridis 1985] Luo, G.L., and G.N.Saridis, "L-Q design of PID controllers for robot arms," *IEEE J. Robot. Autom.,* vol. RA-1, no. 3, pp. 152–159, Sept. 1985.

[Luo et al. 1986] Luo, G.L., G.N.Saridis, and C.Z.Wang, "A dual-mode control for robotic manipulators," *Proc. IEEE Conf. Decision Control,* pp. 409–414, Dec. 1986.

[MATRIXx 1989] MATRIX_X, Santa Clara, CA: Integrated Systems, Inc., 1989.

[Moler 1987] Moler, C., J.Little, and S.Bangert, *PC-Matlab.* Sherborn, MA: The Mathworks, 1987.

[Neuman and Tourassis 1985] Neuman, C.P., and V.D.Tourassis, "Discrete dynamic robot models," *IEEE Trans. Syst. Man and Cybern.,* vol. SMC-15, no. 2, pp. 193–204, Mar./Apr. 1985.

[Paul 1981] Paul, R.P., *Robot Manipulators.* Cambridge, MA: MIT Press, 1981.

[Schilling 1990] Schilling, R.J., Fundamentals of Robotics. Englewood Cliffs, NJ: Prentice Hall, 1990.

[Shin and McKay 1985] Shin, K.G., and N.D.McKay, "Minimum-time control of robotic manipulators with geometric path constraints," *IEEE Trans. Autom. Control,* vol. AC-30, no. 6, pp. 531–541, June 1985.

[Slotine and Li 1991] Slotine, J.-J.E, and W.Li, *Applied Nonlinear Control.* Englewood Cliffs, NJ: Prentice Hall, 1991.

[Vukobratovic and Stokic 1983] Vukobratovic and Stokic, "Contribution to the suboptimal control of manipulation robots," *IEEE Trans. Autom. Control,* vol. AC-28, no. 10, pp. 981–985, Oct. 1983.

[Whitney 1969] Whitney, D.E., "Resolved motion rate control of manipulators and human prostheses," *IEEE Trans. Man Machine Syst.,* vol. MMS-10, no. 2, pp. 47–53, June 1969.

[Wu and Paul 1982] Wu, C.-H., and R.P.Paul, "Resolved motion force control of robot manipulator," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-12, no. 3, pp. 266–275, June 1982.

PROBLEMS

Section 4.2

4.2–1 **Minimum-Time Control.** Derive the minimum-time control switching time t_s [cf. (4.2.11)] when the initial and final velocities are not zero.

4.2–2 **Polynomial Path Interpolation.** It is desired to move a single joint from $q(0)=0$, $\dot{q}(0)=0$ through the point $q(1)=5$, $\dot{q}(1)=40$ to a final position/velocity of $\dot{q}(2)=10$, Determine the cubic interpolating polynomials required in this two-interval path. Plot the path generated and verify that it meets the specified requirements on $q(t)$ and $\dot{q}(t)$ Plot $\dot{q}(t)$ versus $q(t)$.

4.2–3 **LFPB.** Repeat Problem 4.2–2 using LFPB.

4.2–4 **Polynomial Path for Acceleration Matching.** Derive the interpolating polynomial required to match positions, velocities, and accelerations at the via points.

Section 4.3

4.3–1 **Simulation of Flexible Coupling System.** Use computer simulation to reproduce the results for the motor with flexible coupling shaft in Example 3.6.1.

4.3–2 **Simulation of Nonlinear System.** The Van der Pol oscillator is a nonlinear system with some interesting properties. The state equation is

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = -\alpha(x_1^2 - 1)x_2 - x_1.$$

Simulate the dynamics for initial conditions of $x_1(0)=0.1$, $x_2(0)= 0.1$. Use values for the parameter of $\alpha=0.1$ and then $\alpha=0.8$. Plot $x_1(t)$ and $x_2(t)$, as well as $x_2(t)$ vs. $x_1(t)$ in the phase plane. For each simulation you should clearly see the limit cycle that is characteristic of the Van der Pol oscillator.

Section 4.4

4.4–1 Prove (4.4.32).

4.4–2 **PD Computed-Torque Simulation.** Repeat Example 4.4.1 using various values for the PD gains. Try both critical damping and underdamping to examine the effects of overshoot on the joint trajectories.

4.4–3 **Classical Joint Control.** Prove (4.4.55), (4.4.57), (4.4.60), and (4.4.62). See [Franklin et al. 1986].

4.4–4 **PD Computed Torque with Payload Uncertainty. The CT** controller is inherently robust. In Example 4.4.1, suppose that m_2 changes from 1 kg to 2 kg at t=5 s, corresponding to a payload mass being picked up. The CT controller, however, still uses a value of m_2=1. Use simulation to plot the error time history. Does the performance improve with larger PD gains?

4.4–5 **PID Computed Torque with Payload Uncertainty.** Repeat Problem 4.4–4 using a PID outer loop. Does the integral term help in rejecting the mass uncertainty?

4.4–6 **PD Computed Torque with Friction Uncertainty.** Repeat Problem 4.4–4 assuming now that m_2=1 kg stays constant and is known to the controller. However, add friction of the form $F(q, \dot{q}) = F_v\dot{q}+K_d\text{sgn}(\dot{q})$(see Table 3.3.1) to the arm dynamics, but not to the CT controller. Use v_i=0.1, k_i=0.1. Simulate the performance for different PD gains.

4.4–7 **PID Computed Torque with Friction Uncertainty.** Repeat Problem 4.4–6 using a PID outer loop.

4.4–8 **PD Computed Torque with Actuator Dynamics**

(a) Design a CT control law for the two-link planar elbow arm with actuator dynamics (Section 3.6) of the form

$$J_M\ddot{q}_M + B\dot{q}_M + F_M + R\tau = K_M v.$$

Take the link masses and lengths as 1 kg, 1 m. Take motor parameters of J_m=0.1 kg-m^2, b_m=0.2 N-m/rad/s, and R=5Ω, Set the gear ratio

r=0.1.

(b) Simulate the controller for various values of PD gains.

4.4–9 PD Computed Torque with Neglected Actuator Dynamics. Consider the arm-plus-dynamics in Problem 4.4–8. Suppose, however, that the CT controller was designed using only the arm dynamics and neglecting the actuator dynamics. Use the PD gains in Example 4.4.1. Simulate the control law on the arm-plus-dynamics for various values of gear ratio r. As r decreases, the performance should deteriorate.

4.4–10 PD Computed Torque Using Only Actuator Dynamics. Consider the arm-plus-dynamics in Problem 4.4–8. Design a CT controller using only the actuator dynamics and no arm dynamics. Simulate the control law on the arm-plus-dynamics for various values of gear ratio r. As r decreases, the performance should improve. For fixed r, it is also instructive to try different PD gains.

4.4–11 Classical Joint Control with Actuator Dynamics. Repeat Example 4.4.4 including actuator dynamics like those in Problem 4.4–8. Try different values of gear ratio r. Compare to Problem 4.4–10.

4.4–12 PD Computed Torque with Flexible Coupling. Combine the flexible shaft in Example 3.6.1 with the two-link arm in Example 4.4.1 to study the effects of using CT control on a robot with compliant motor coupling. Try different values of the coupling shaft parameters.

4.4–13 Error Dynamics with Approximate CT Control. Considzer the two-link polar arm in Example 3.2.1 with friction of the form Find the error dynamics (4.4.44) for the cases:
(**a**) Friction is not included in the CT control law.
(**b**) Payload mass m_2 is not exactly known in the CT control law.
(**c**) PD-gravity CT is used.
(**d**) PD classical joint control is used with no nonlinear terms.

Section 4.5

4.5–1 Digital Control Simulation

(**a**) Repeat Example 4.5.1 using a desired trajectory with period of 1

s instead of 2 *s*. Plot as well the Cartesian position $(x_2(t), y_2(t))$ of the end effector in base coordinates.

(**b**) Redo the simulation deleting the lines in Figure 4.5.7 that zero the initial velocity estimates.

(**c**) Try to simulate the digital CT controller using the alternative technique to compute $\dot{\boldsymbol{e}}_k$ from $\dot{\boldsymbol{q}}$ and v_k, as given in equation (1) in the example.

4.5–2 **Digital Control Simulation.** Convert the PD-gravity CT controller in Example 4.4.3 to a digital controller. Try several sample periods.

4.5–3 **Error Dynamics for Digital Control.** Find the error system in Table 4.4.1 using digital control of the form (4.5.3).

4.5–4 **Antiwindup Protection.** In Example 4.4.4 we saw the deleterious effects in robot control of integrator windup due to actuator saturation. In Example 4.5.2 we showed how to implement antiwindup protection on a simple PI controller. Implement antiwindup protection on the robot controller in Example 4.4.4. The issue is determining the limits on the integrator outputs given the motor torque limits. Successful and thorough completion of this problem might lead to a nice conference paper.

Section 4.6

4.6–1 **Optimal LQ Outer-Loop PD Gains.** Verify (4.6.15). To do this, select the Riccati solution matrix as

$$P = \begin{bmatrix} P_1 & P_2 \\ P_2^T & P_4 \end{bmatrix}$$

with $P_i \in R^n$ Substitute P, A, B, Q, R into the Riccati equation (4.6.6). You will obtain three $n \times n$ equations that can be solved for P_i. Now use (4.6.5).

4.6–2 **Robust Control Using LQ Outer-Loop Design.** Redo the PDgravity simulation in Example 4.4.3 using PD gains found from LQ design. Does the LQ robustness property improve the responses found in Example 4.4.3 using nonoptimal gains? Try various choices for Q_p, Q_v, and R, both diagonal and nondiagonal.

Section 4.7

4.7–1 **Direct Cartesian Computed-Torque Design.** Begin with the Cartesian dynamics in Section 3.5 and design a computed-torque controller. Compare it to (4.7.10).

4.7–2 **Approximate Cartesian Computed-Torque.** Derive the error system dynamics associated with the approximate control law (4.7.11).

4.7–3 **Cartesian PD-Plus-Gravity Control.** Repeat Example 4.4.3 using Cartesian computed-torque control, where the trajectory is given in workspace coordinates.