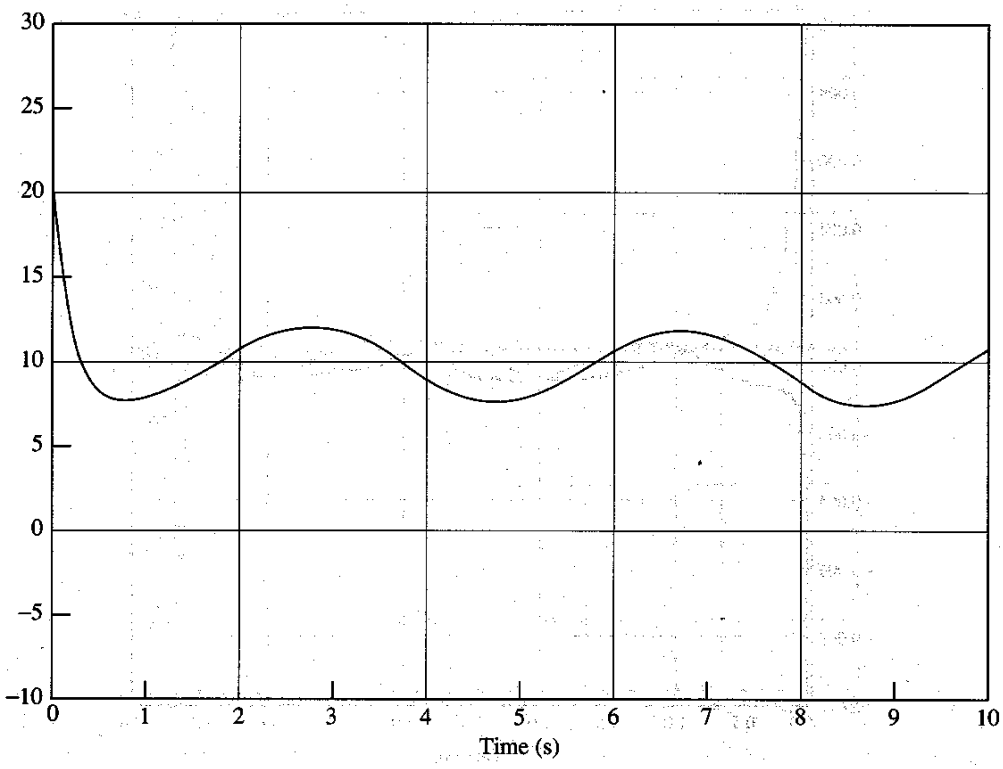
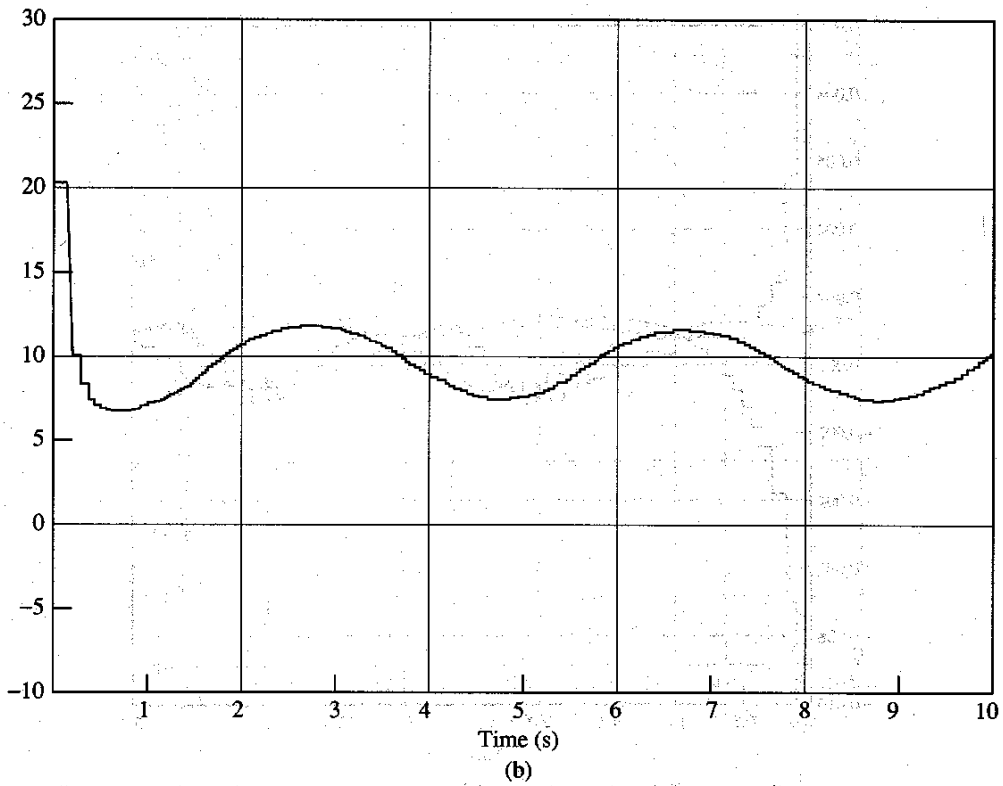


(a)



(b)

FIGURE 3.5-6 Link 2 torque input in the digital CT controller: (a) $T = 5$ ms; (b) $T = 50$ ms.

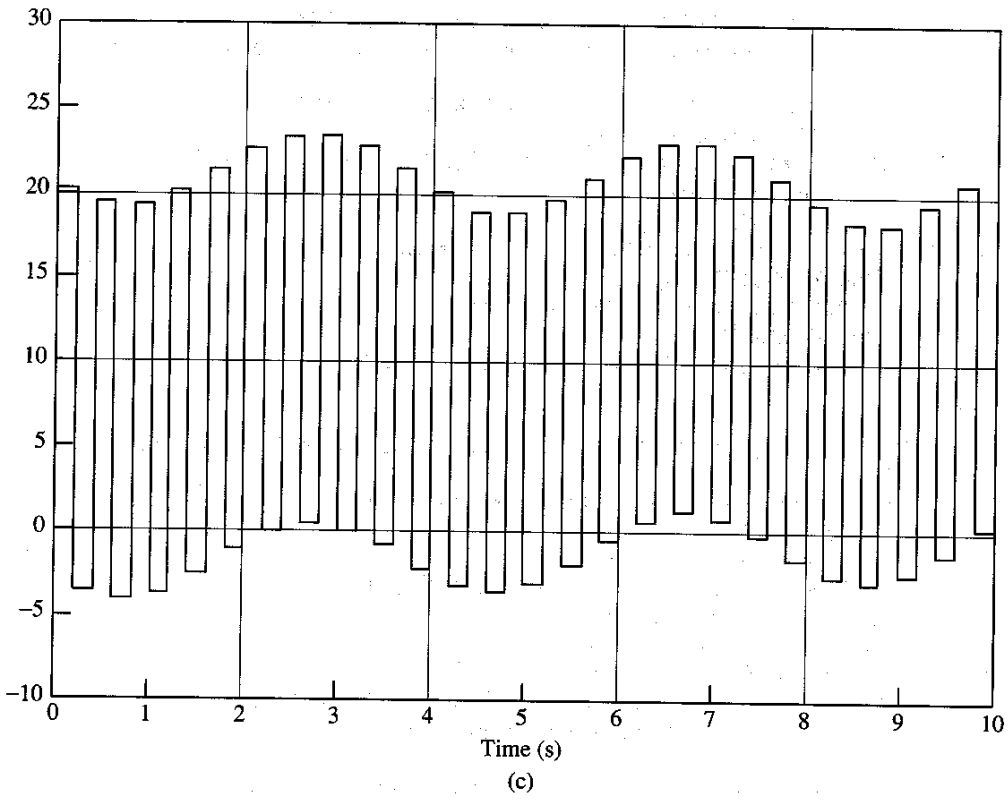


FIGURE 3.5-6 (*Cont.*) (c) $T = 100$ ms.

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.

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

In Example 3.4-1 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 Fig. 3.5-1 and the technique given in Section 3.3 for digital controller simulation.

In Fig. 3.5-4 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 3.4-1.

Running now program TRESP with different sample periods T yields the tracking error plots shown in Fig. 3.5-5. 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 3.4-1. However, as the sampling period T increased the tracking performance deteriorated.

The torque input for joint 2 is depicted in Fig. 3.5-6. For small T is is virtually identical to the continuous CT controller in Example 3.4-1. A very interesting

```
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 xm1(2), em1(2), v(2), vm1(2), ev(2), evm1(2), nu
       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)
       COMMON/PARAM/ nu
       DATA m1,m2,a1,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*vm1(I) + (x(I) - xm1(I)) / T
       IF (IK.EQ.0) v(I)= 0
       e(I) = qd(I)  - x(I)
       ev(I)= nu*evm1(I) + (e(I) - em1(I))/T
10     IF(IK.EQ.0) ev(I)= 0

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*v(1)*v(2) + v(2)**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*v(1)**2*sin(x(2))  + m2*g*a2*cos(x(1)+x(2))

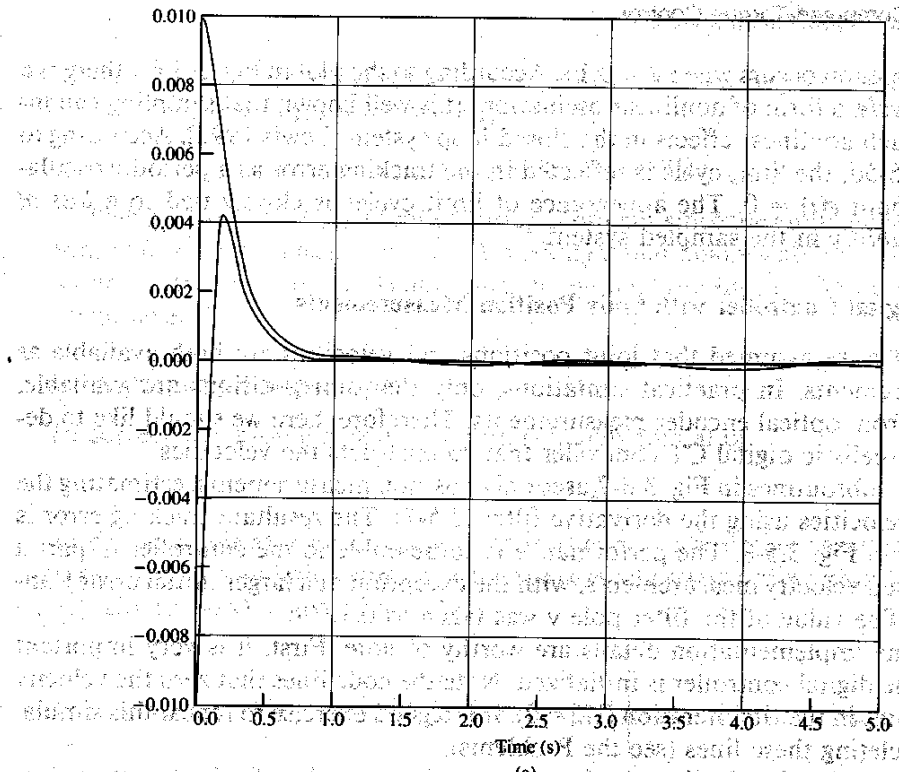
C   COMPUTATION OF CONTROL TORQUES

       s1= qdpp(1) + kp*e(1) + kv*ev(1)
       s2= qdpp(2) + kp*e(2) + kv*ev(2)
       t1= M11*s1 + 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 3.5-7 Digital control subroutine that uses only joint position measurements.



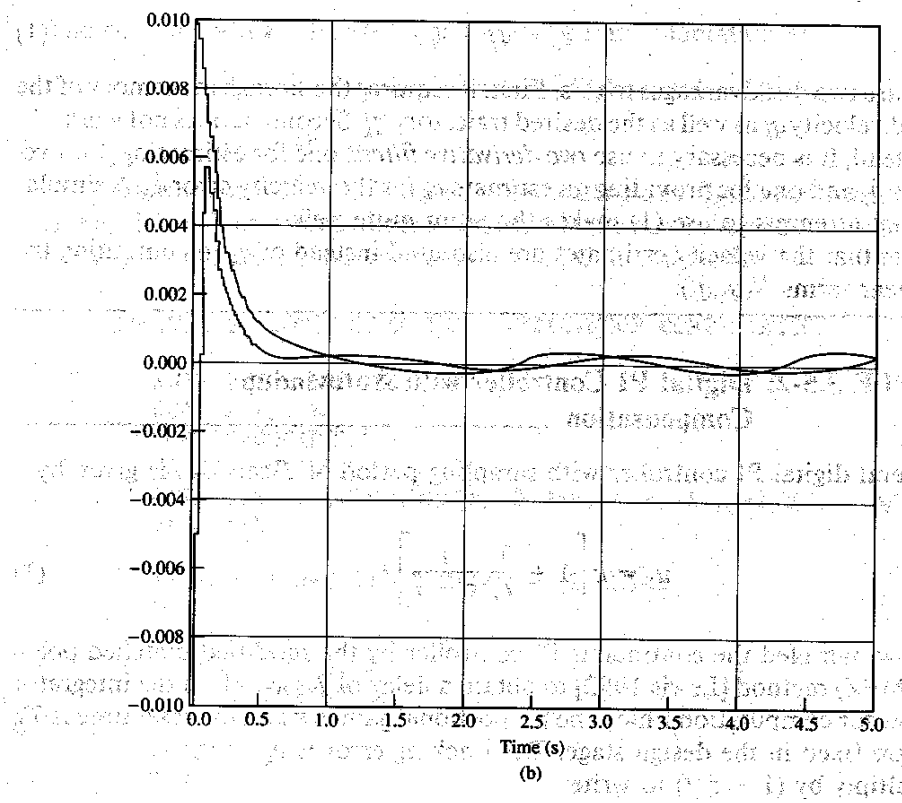


FIGURE 3.5-8 Tracking error for digital controller using no velocity measurements: (a) $T = 5$ ms; (b) $T = 20$ ms.

phenomenon occurs when $T = 0.1$ s. According to the plot in Fig. 3.5-6c, 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 Fig. 3.5-5d, 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.

b. Digital Controller with Only Position Measurements

In part a we assumed that joint positions and velocities are both available as measurements. In practical simtations, 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 Fig. 3.5-7 uses only position measurements, estimating the joint velocities using the derivative filter (3.5-7). The resultant tracking error is shown in Fig. 3.5-8. 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 $\dot{e}_k = \dot{e}(kT)$ is to find an estimate v_k of the joint velocity $\dot{q}_k$ 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.

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 3.5-2: Digital PI Controller with Antiwindup 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-pole–zero (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 $(1 - z^{-1})$ to write

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

which is in the transfer function form (3.5-21). The corresponding difference equation form for implementation is

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

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

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

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 (3.5-23), (3.5-24) is given by

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

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

The corresponding difference equations for implementation are

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

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

A few lines of FORTRAN code implementing this digital controller are given in Fig. 3.5-9. 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 3.5-9 FORTRAN code implementing PI controller with antiwindup compensation.

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

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

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 $\alpha = 0$ is thus called the *velocity form* of the PI controller.

b. Digital Control of DC Motor

Consider the simplified model for a dc motor given by

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

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

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

where $e = r - \omega$ 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

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

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 Fig. 3.5-10a. 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., $\alpha = 1$ in Fig. 3.5-9). The resulting behavior shown in Fig. 3.5-10b displays an unacceptable overshoot.

Figure 3.5-10c shows that the overshoot problem is easily corrected using $\alpha = 0.9$ in the digital PI controller with antiwindup protection in Fig. 3.5-9.

In this example, as α decreases the step response slows down. The value of $\alpha = 0.9$ was selected after several simulation runs with different values of α.

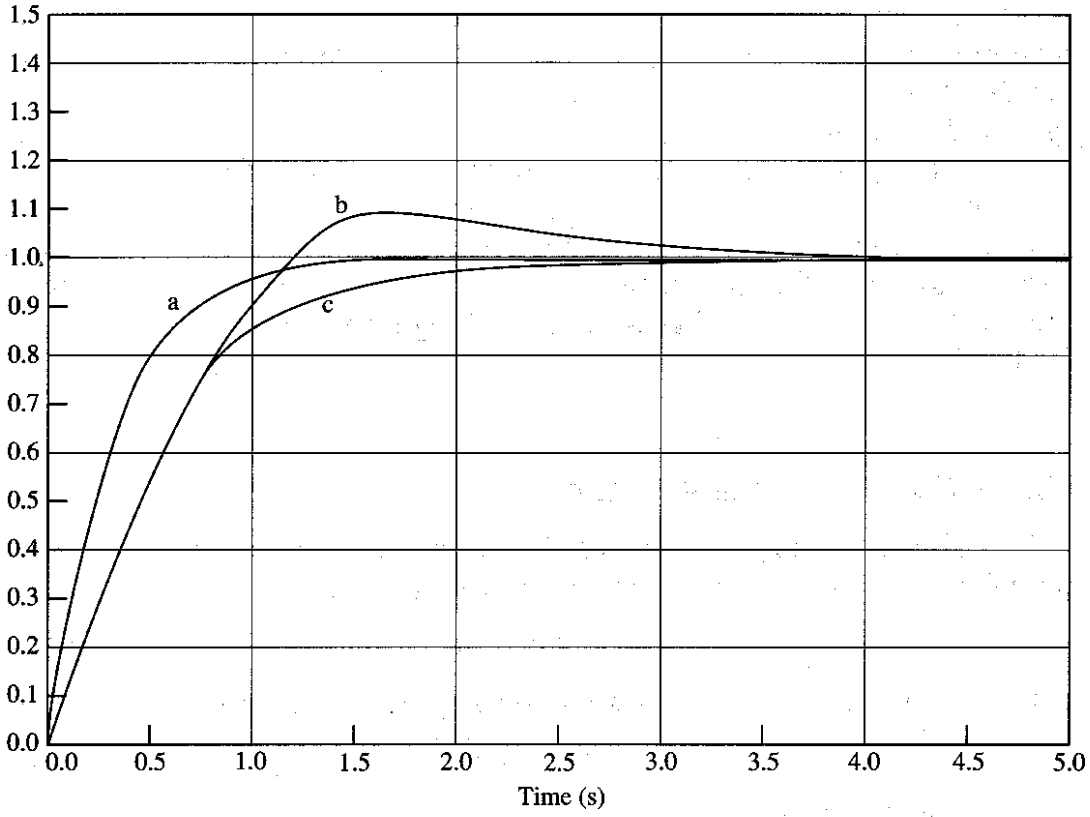


FIGURE 3.5-10 Angular velocity step responses using digital controller: (a) digital PI controller with no saturation limits; (b) digital PI controller with saturation limit $u_H = 1.5$ V and no antiwindup compensation; (c) digital PI controller with saturation limit $u_H = 1.5$ V and antiwindup compensation with $\alpha = 0.9$.

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.