EXAMPLE A.2–5: Inverse Kinematics for Two-Link Planar Elbow Arm

For the planar RR arm of Example A.2.2, the inverse kinematics problem amounts to finding the joint variable θ_1 and θ_2 given a desired Cartesian position (x, y) of the end of the arm. See Figure A.2.7.

The first thing that is evident is that, as long as $a_1^2+a_2^2 < r \equiv x^2+y^2$, there are two solutions. The one shown in Figure A.2.7 is the "elbow down" solution. Another solution may be determined for the "elbow up"configuration, where both links are above the vector $[x\ y]^T$. Thus *the inverse kinematics problem generally has a nonunique solution*. This may often be taken advantage of to obtain end-effector positioning with *collision avoidance*.

Given the T matrix from Example A.2.2, we see that θ_1 and θ_2 may be found by solving

$$a_1 c_1 + a_2 c_{12} = x \tag{1}$$

$$a_1 s_1 + a_2 s_{12} = y. \tag{2}$$

Referring to Figure A.2.7, define

$$r^2 = x^2 + y^2 \tag{3}$$

and use the law of cosines to obtain

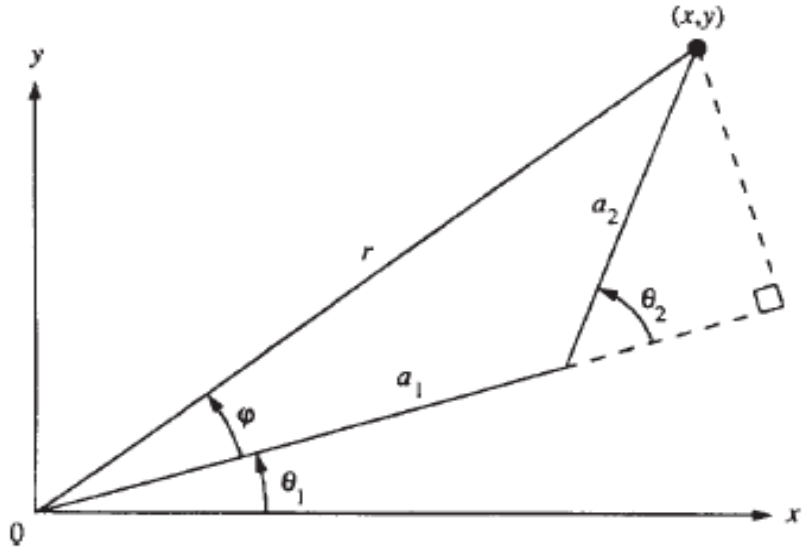


Figure A.2.7: Inverse kinematics for two-link planar RR arm.

$$\begin{aligned} r^2 &= a_1^2 + a_2^2 - 2a_1a_2\cos(\pi - \theta_2) \\ &= a_1^2 + a_2^2 + 2a_1a_2\cos\theta_2. \end{aligned} \tag{4}$$

Therefore, we proceed by computing

$$\cos\theta_2 = \frac{r^2 - a_1^2 - a_2^2}{2a_1a_2} \equiv C \tag{5}$$

$$\sin\theta_2 = \pm\sqrt{1 - \cos^2\theta_2} = \pm\sqrt{1 - C^2} \equiv D \tag{6}$$

$$\theta_2 = \text{ATAN2}(D, C)\,. \tag{7}$$

To determine θ_1 define the auxiliary angle φ in the figure. By inspection of the right triangle shown,

$$\tan\varphi = \frac{a_2 \sin\theta_2}{a_1 + a_2\cos\theta_2}. \tag{8}$$

Moreover,

$$\tan(\varphi + \theta_1) = \frac{y}{x}, \tag{9}$$

so that

$$\theta_1 = \mathrm{ATAN2}(y, x) - \mathrm{ATAN2}(a_2 \sin\theta_2,\ a_1 + a_2\cos\theta_2). \tag{10}$$

Note that θ_1 depends on θ_2.

■

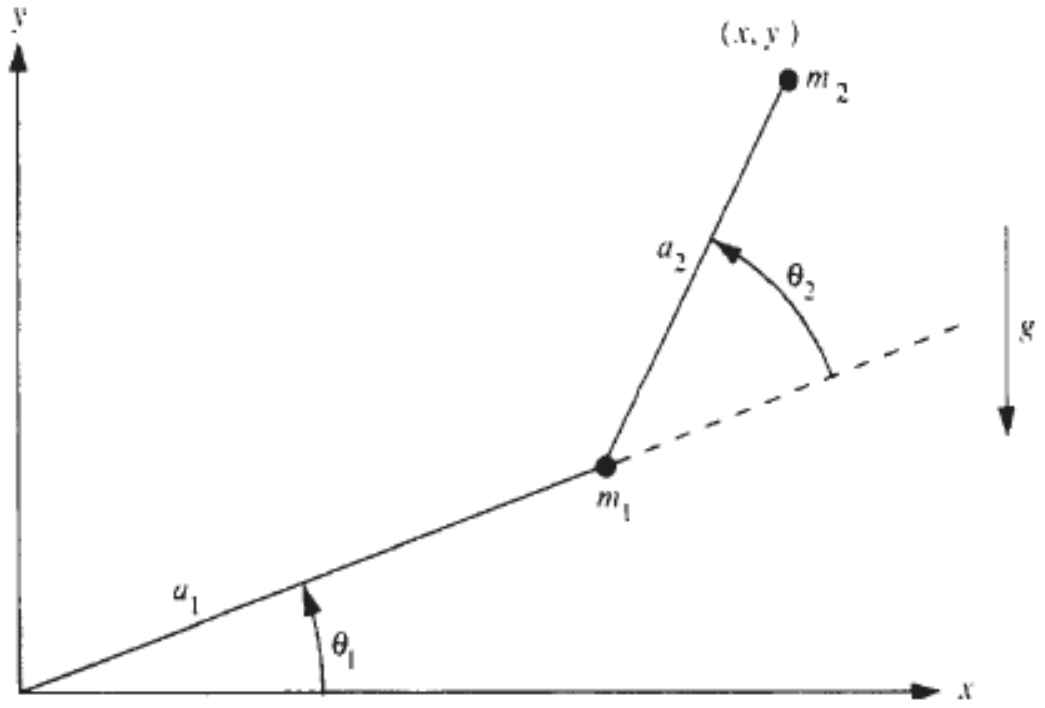


Figure 4.2.1: Two-link planar elbow arm.

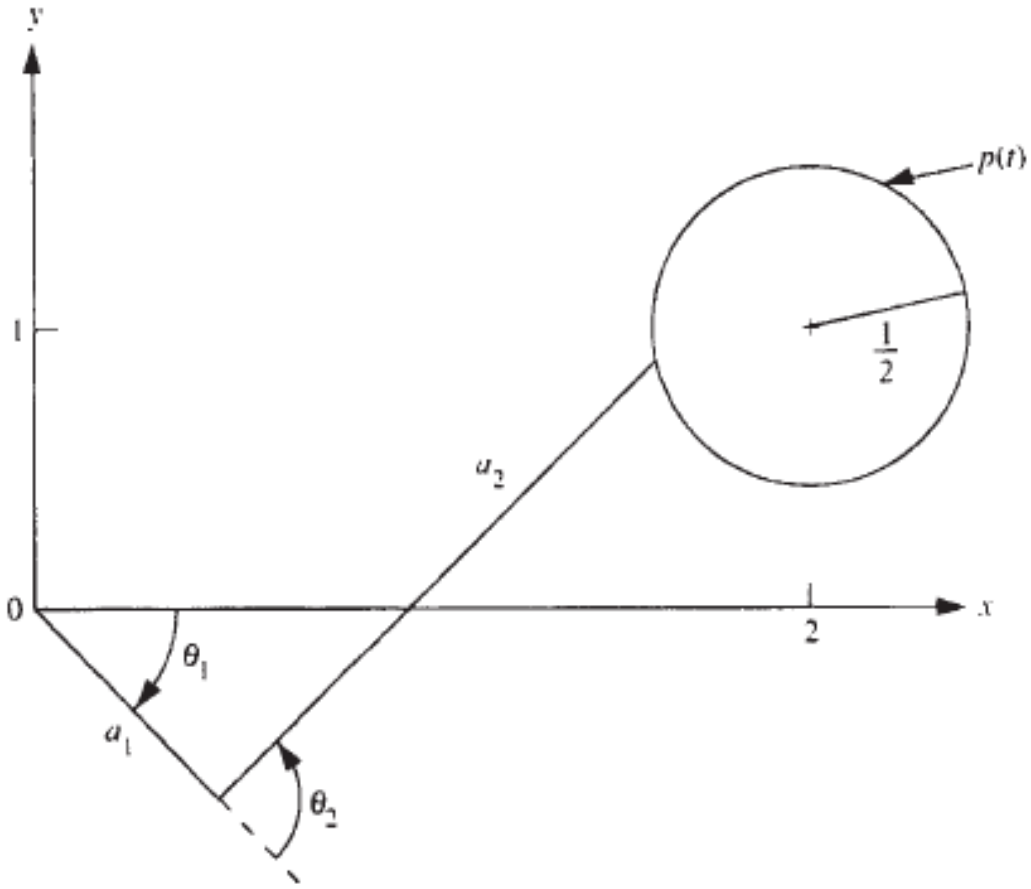
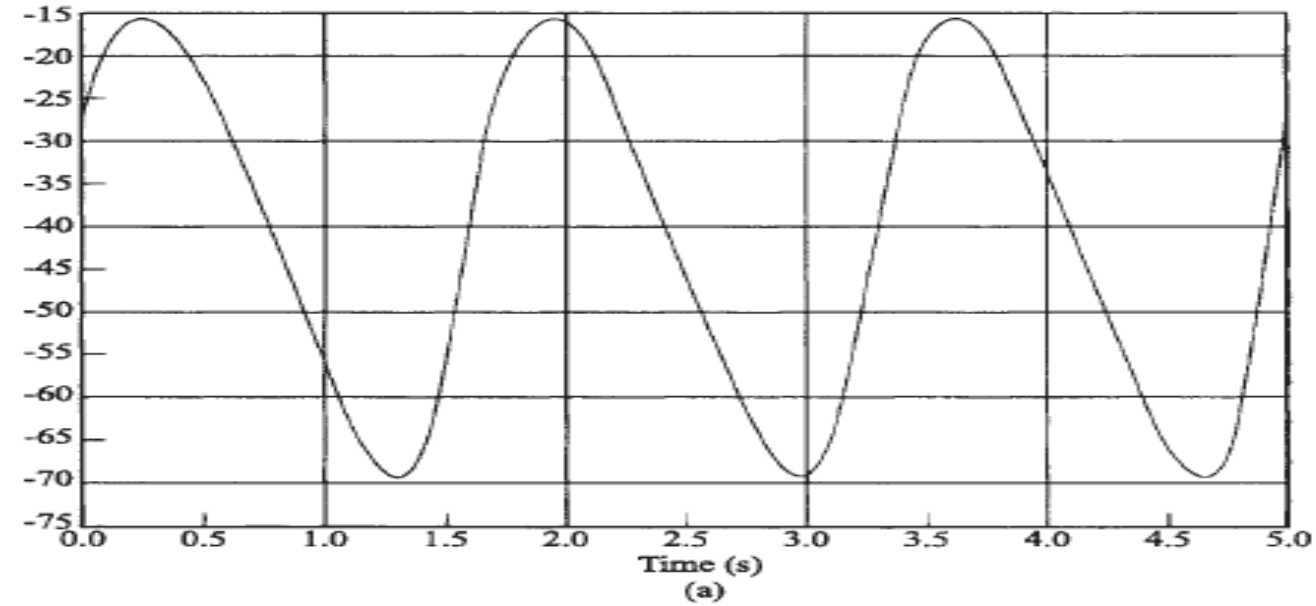


Figure 4.2.2: Desired Cartesian trajectory.



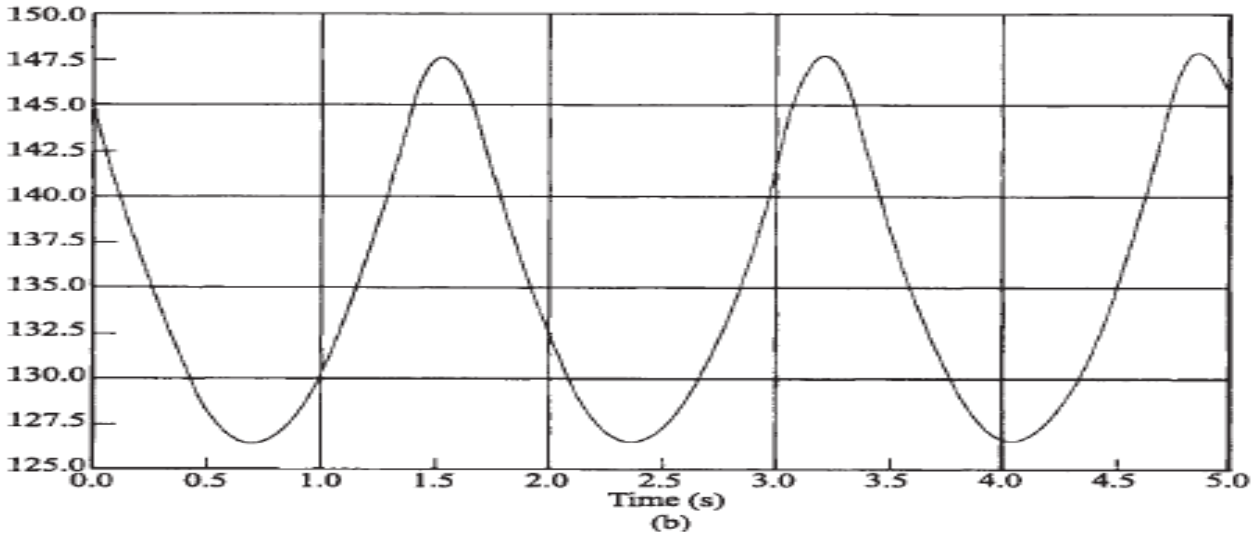
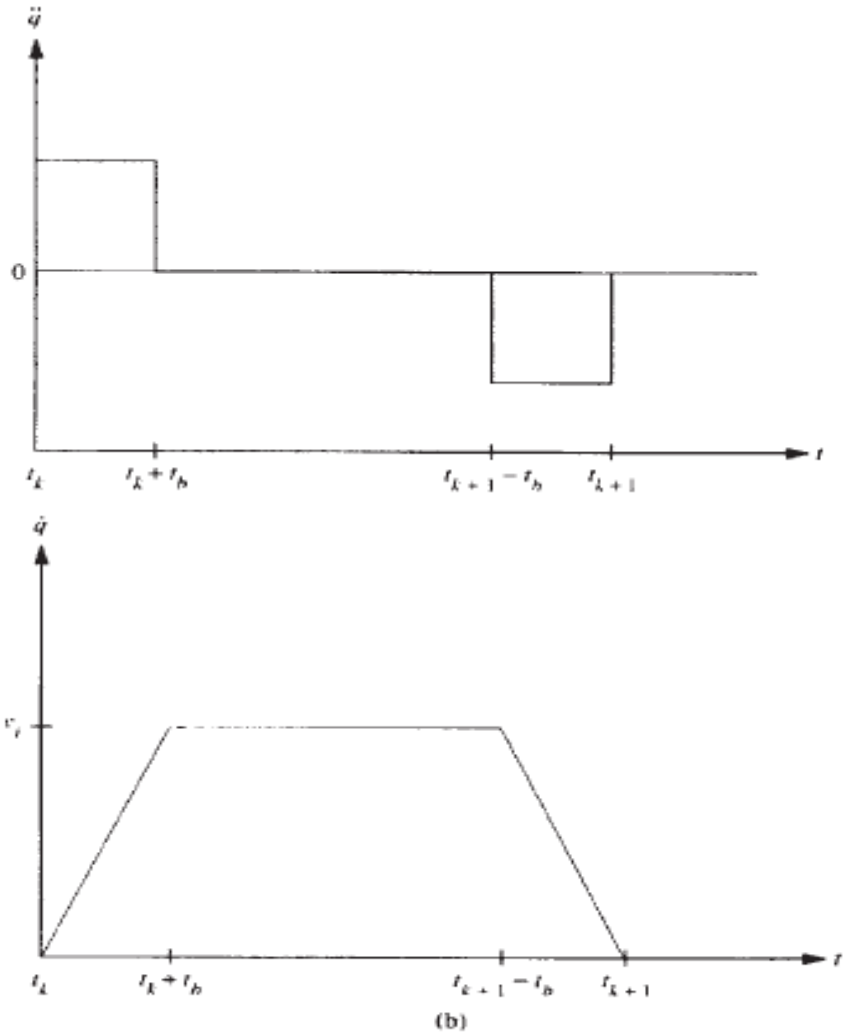


Figure 4.2.3: Required joint-space trajectories: (a) θ_1 (deg); (b) θ_2 (deg).



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

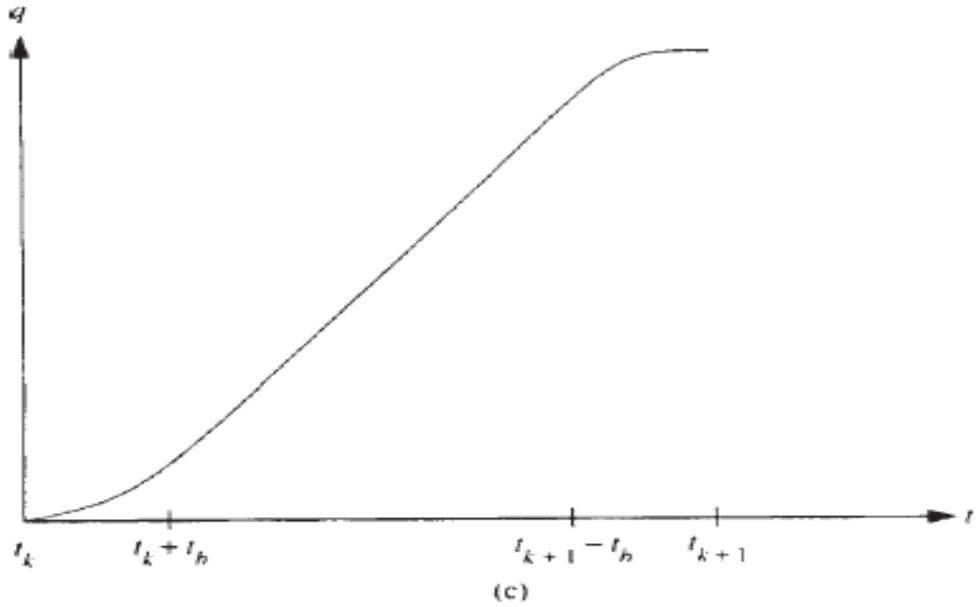


Figure 4.2.4: (*Cont.*)(c) position.

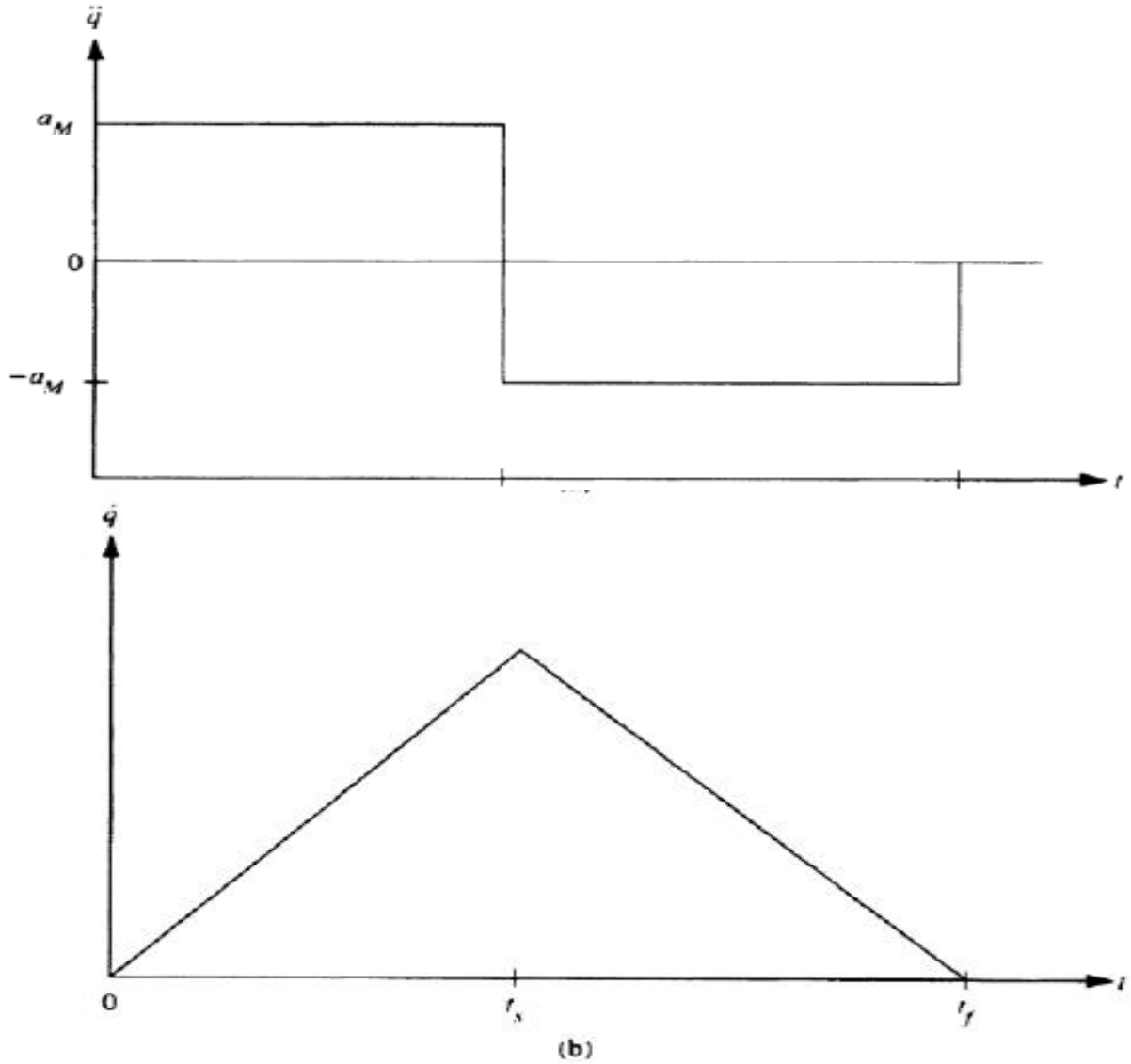


Figure 4.2.5: Minimum-time trajectory: (a) acceleration; (b) velocity.

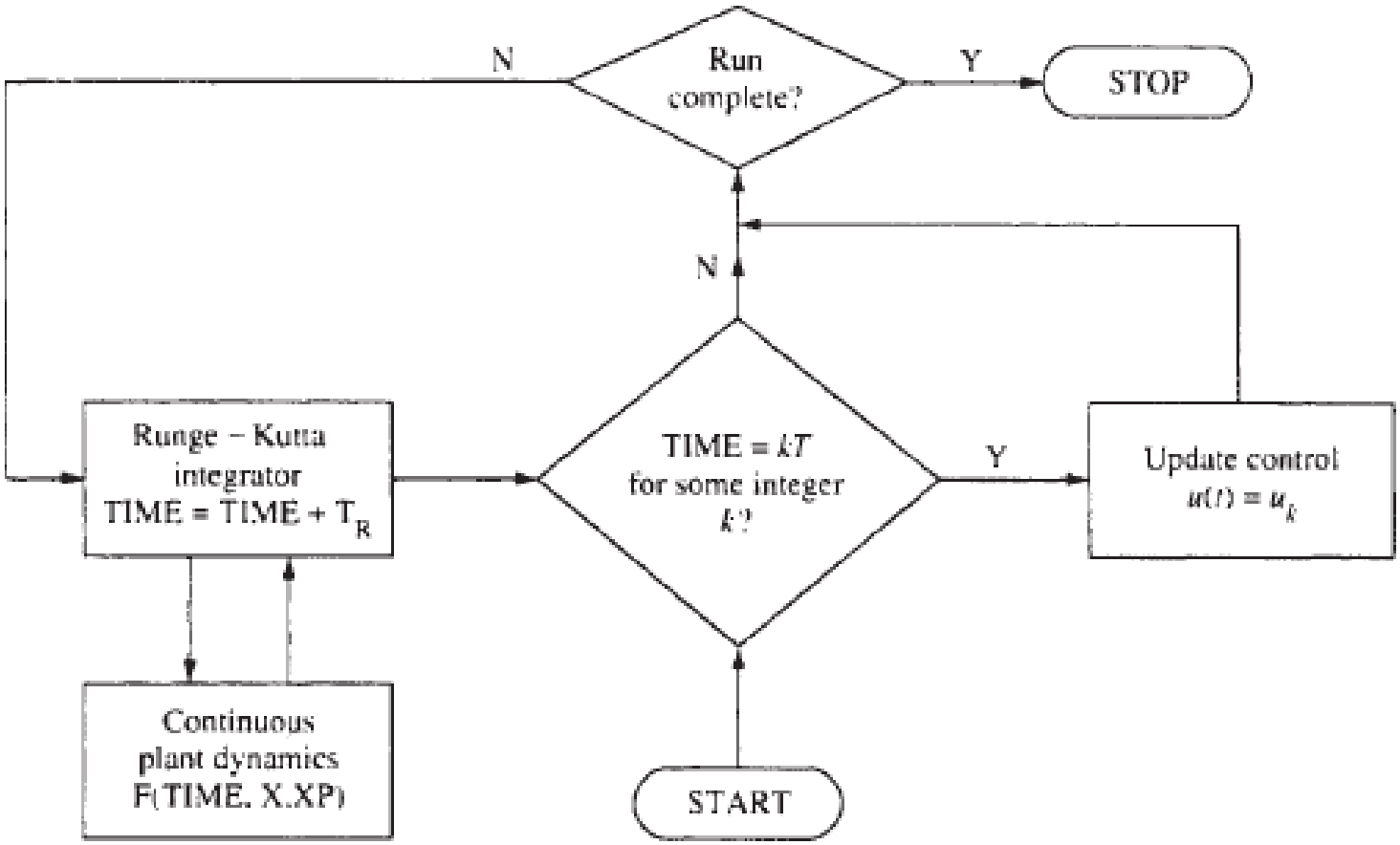


Figure 4.3.3: Digital control simulation scheme.

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

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

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

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

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

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 g1, 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)=    g1*sin(fact*t)
       qd(2)=    g2*cos(fact*t)
       qdp(1)=   g1*fact*cos(fact*t)
       qdp(2)=  -g2*fact*sin(fact*t)
       qdpp(1)= -g1*fact**2*sin(fact*t)
       qdpp(2)= -g2*fact**2*cos(fact*t)
C
       RETURN
       END
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   COMPUTED-TORQUE CONTROLLER SUBROUTINE
      SUBROUTINE CTL(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   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

      y1= a1*cos(x(1)) + a2*cos(x(1)+x(2))
      y2= a1*sin(x(1)) + a2*sin(x(1)+x(2))

      RETURN
      END
```

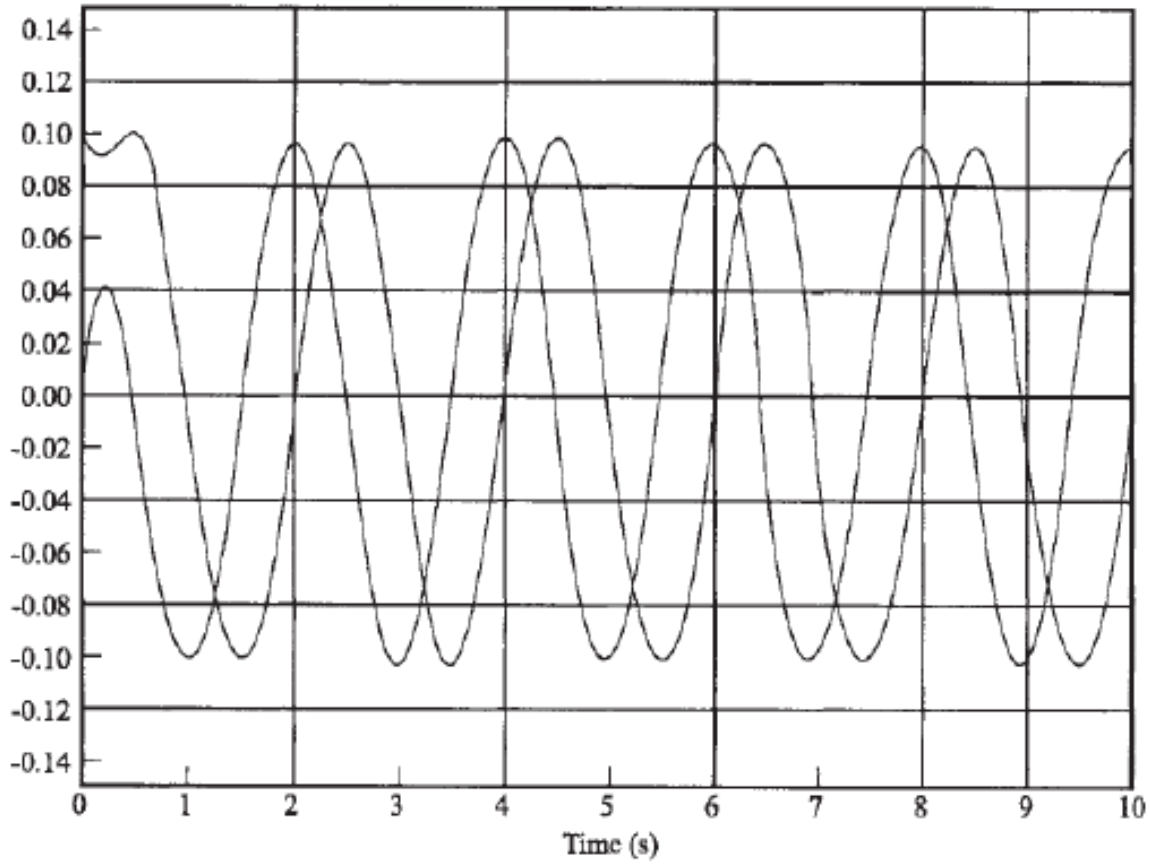


Figure 4.4.3: Joint angles $\theta_1(t)$ and $\theta_2(t)$ (red).

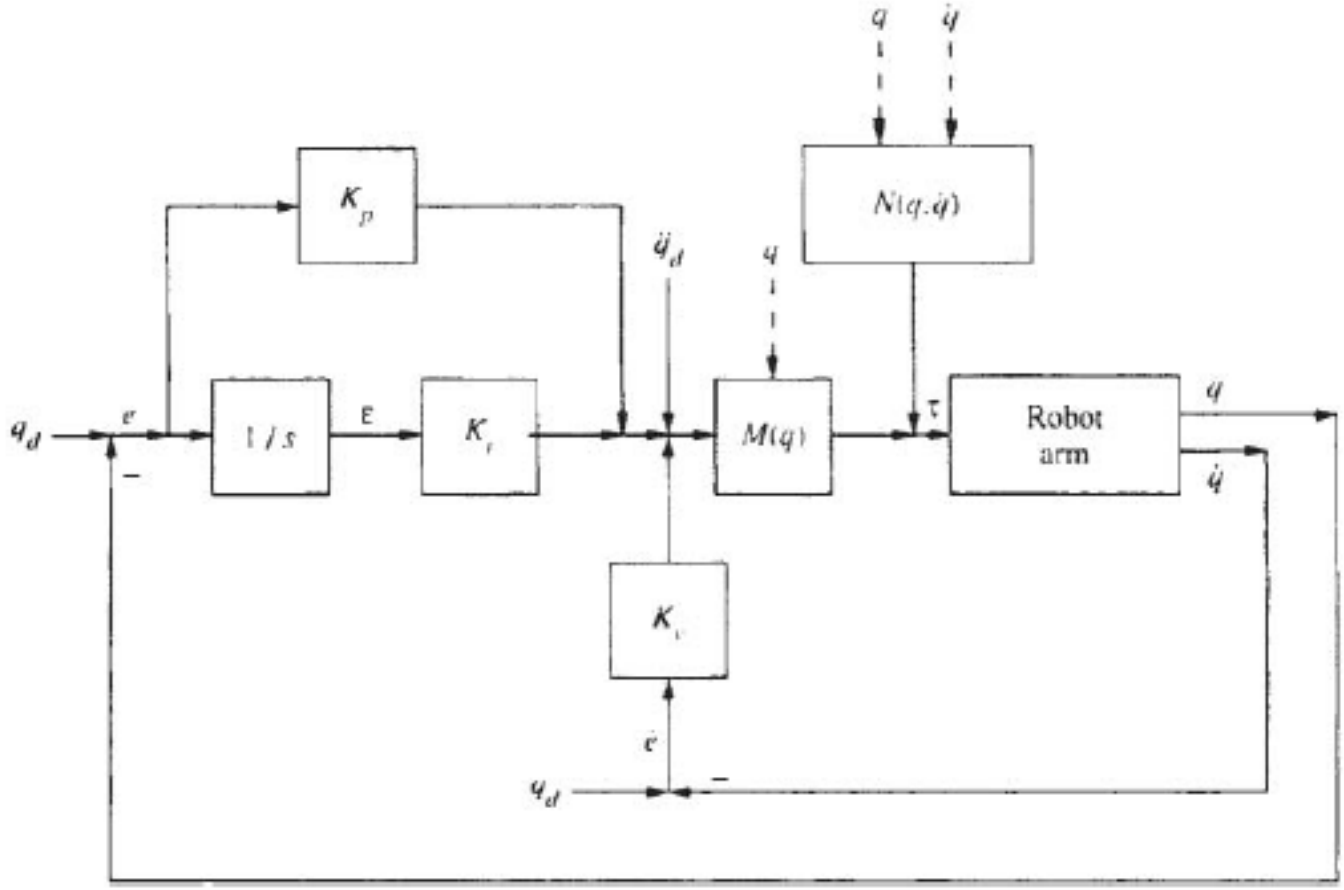


Figure 4.4.6: PID computed-torque controller.

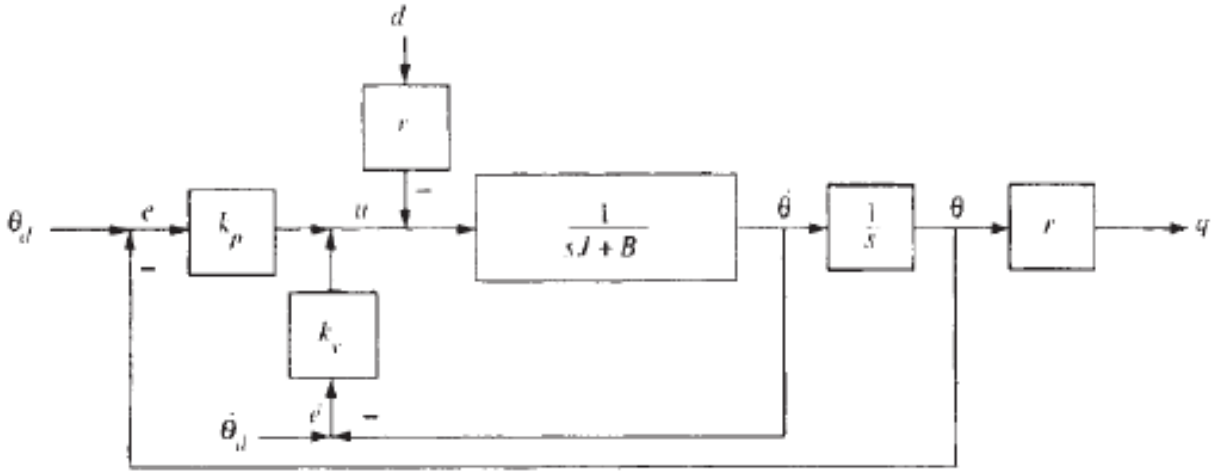


Figure 4.4.11: PD independent joint control.

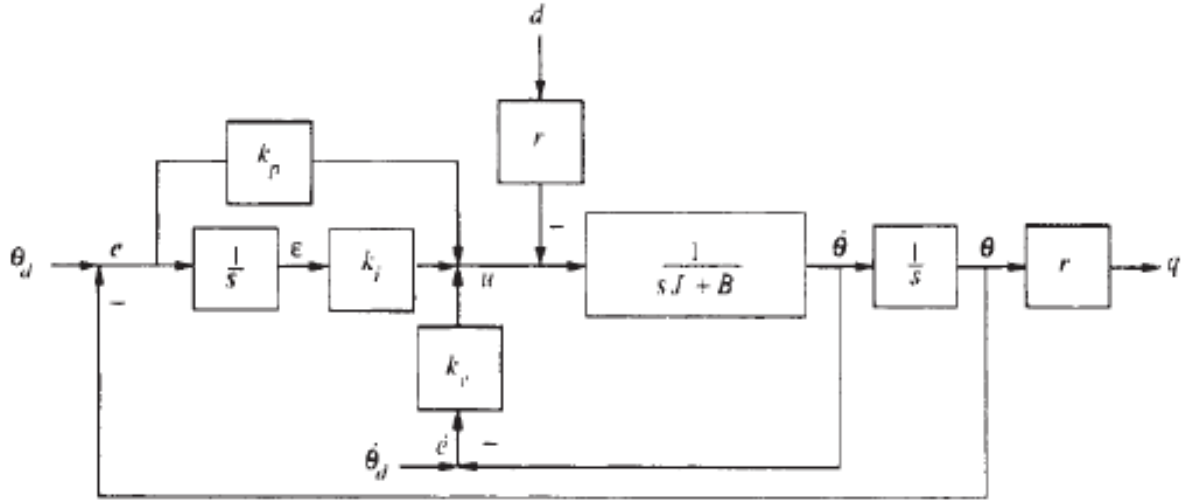


Figure 4.4.12: PID independent joint control.

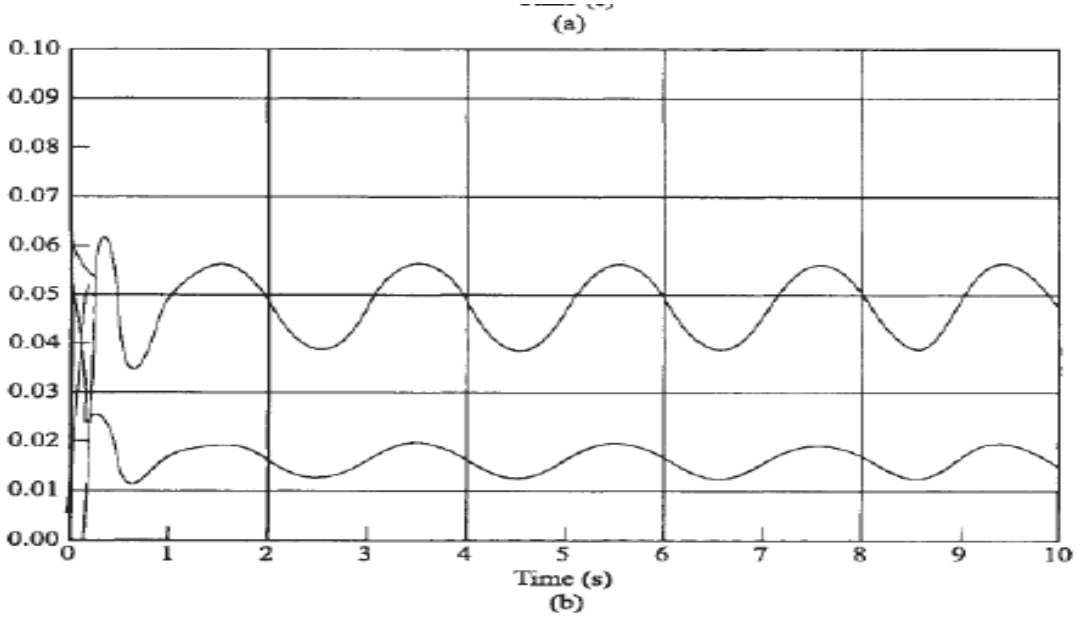


Figure 4.4.13: PD classical joint control tracking error $e_1(t)$, $e_2(t)$ (red): (a) ω_n=10 rad/s; (b) ω_n=25 rad/s.

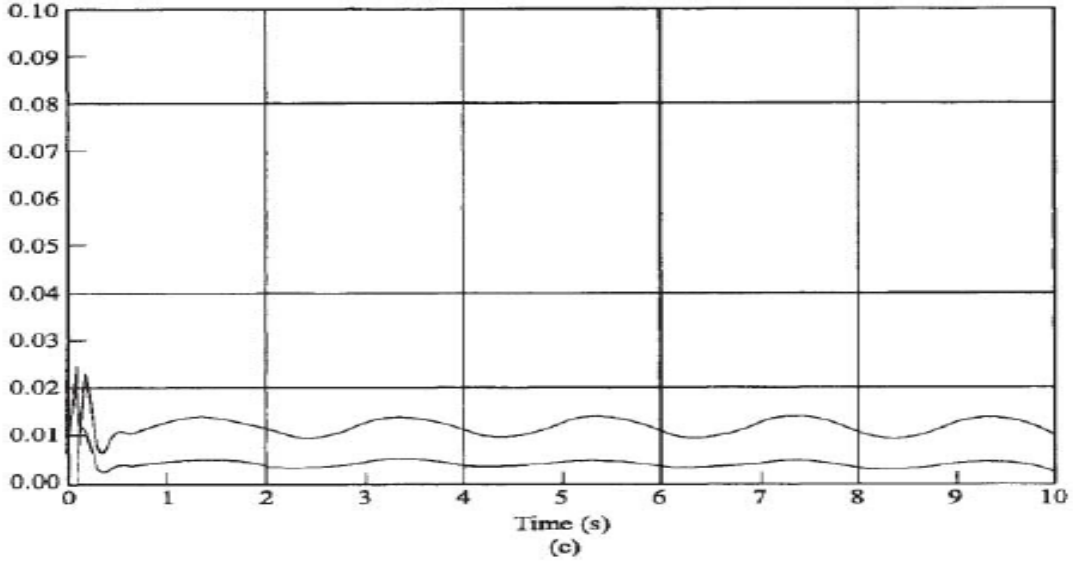


Figure 4.4.13 (*Cont.*) (c) ω_n=50 rad/s.

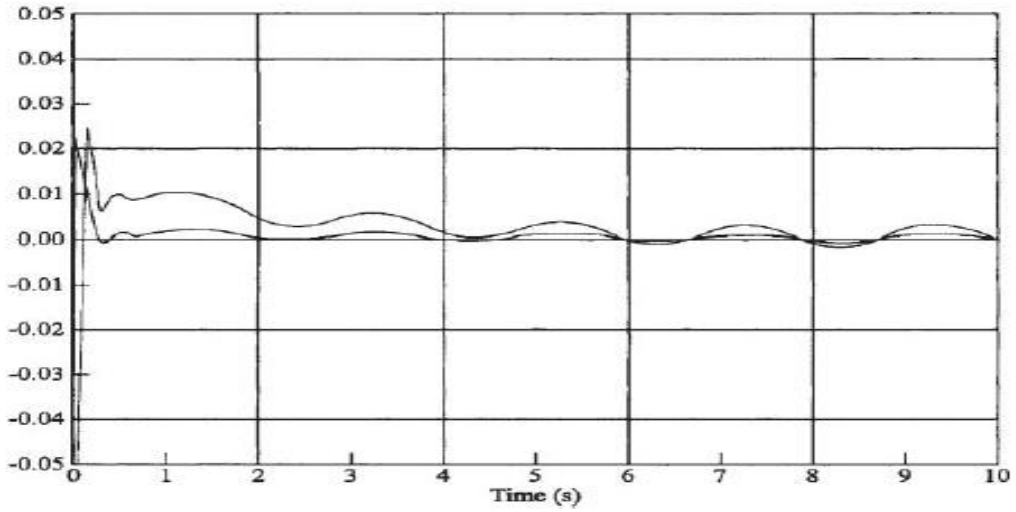


Figure 4.4.15: PD classical joint control: k_v=100, k_p=2500, k_i=1000.

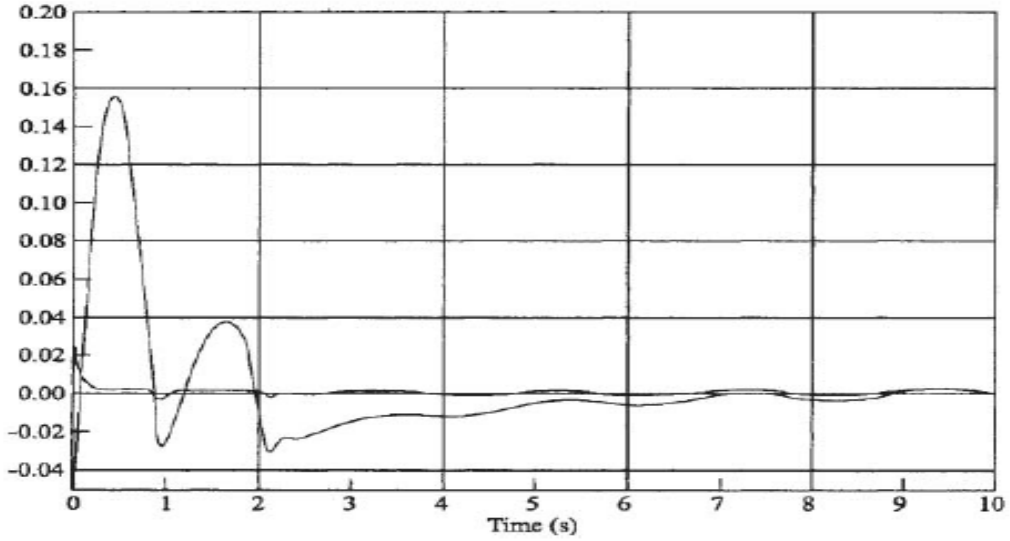


Figure 4.4.16: PID classical joint control with torque limits of ±35 N-m. Tracking error in rads.

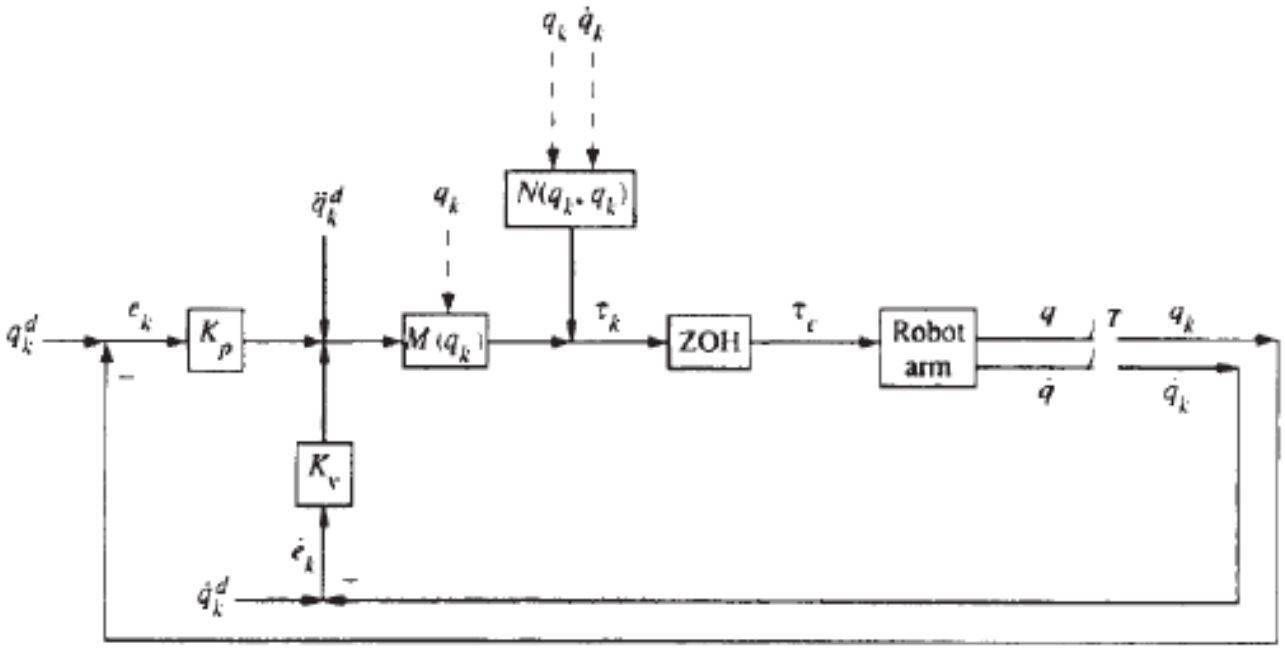


Figure 4.5.1: Digital robot control scheme.

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):

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