

Real-Time Systems

Solutions to Exercise 6: Implementation of Control Algorithms

1 a. The forward difference approximation is obtained if we put

$$p = \frac{q-1}{h}$$

where q is the forward shift operator and h is the sampling period. This gives

$$u(t) = K \left(1 + \frac{h}{(q-1)T_i} + \frac{T_d(q-1)}{h} \right) e(t) \Rightarrow$$

$$(q-1)hT_i u(t) = K((q-1)hT_i + h^2 + T_d T_i (q-1)^2) e(t).$$

Notice that on the left hand side we have terms including $u(t)$ and $u(t+h)$ and that we have terms including $e(t+2h)$, $e(t+h)$, and $e(t)$ on the right hand side. Thus the difference equation is non-causal and is therefore impossible to implement.

b. If we use the same approximation as in a) we get:

$$u(t) = K \left(1 + \frac{h}{(q-1)T_i} + \frac{T_d(q-1)}{h} \frac{1}{1 + \frac{T_d(q-1)}{Nh}} \right) e(t) \Rightarrow$$

$$(q-1)(q + \frac{Nh - T_d}{T_d})T_i T_d u(t) = K \left((q-1)(q + \frac{Nh - T_d}{T_d})T_i T_d + hT_d(q + \frac{Nh - T_d}{T_d}) + NT_d T_i (q-1)^2 \right) e(t).$$

This is a causal difference equation. The poles of the transfer function are located in $z = 1$ (integrator) and in $z = 1 - Nh/T_d$. The second pole is unstable (outside the unit circle) if h is large or if T_d is small. Also, the case $T_d = 0$ must be treated separately. These problems do not arise when the backward difference approximation is used.

2 a. Applying Tustin's approximation to the polynomial $s_0 p + s_1$ gives

$$s_0 \frac{2}{h} \frac{q-1}{q+1} + s_1.$$

Repeating the same operations for $p + r_1$ and $t_0 p + t_1$ and putting it all together gives

$$((2 + hr_1)q + (hr_1 - 2))u(t) = -((2s_0 + hs_1)q + (hs_1 - 2s_0))y(t) + ((2t_0 + ht_1)q + (ht_1 - 2t_0))u_c(t)$$

after multiplication with $h(q+1)$ on both sides.

```

b. public class RSTController {
    private double h, rd1, sd0, sd1, td0, td1;
    private double uold = 0.0, yold = 0.0, ucold = 0.0, preu = 0.0;

    RSTController(double h) {
        this.h = h;
    }

    public void setParameters(double r1, double s0, double s1,
                             double t0, double t1) {
        rd0 = 2.0 + h*r1;
        rd1 = (h*r1-2.0)/rd0;
        sd0 = (2.0*s0+h*s1)/rd0;
        sd1 = (h*s1-2.0*s0)/rd0;
        td0 = (2.0*t0+h*t1)/rd0;
        td1 = (h*t1-2.0*t0)/rd0;
    }

    public double calculateOutput(double uc, double y) {
        uold = preu - sd0*y + td0*uc;
        yold = y;
        ucold = uc;
        return uold;
    }

    public void updateState() {
        preu = -rd1*uold - sd1*yold + td1*ucold;
    }
}

```

3 a. Eliminating the equation for $\hat{x}(k | k)$, we get

$$\begin{aligned}\hat{x}(k+1 | k) &= \Phi \hat{x}(k | k-1) + \Gamma u(k) + K(y(k) - C \hat{x}(k | k-1)) \\ u(k) &= -L(\hat{x}(k | k-1) + K_f(y(k) - C \hat{x}(k | k-1)))\end{aligned}$$

Inserting the expression for $u(k)$ into the first equation and simplifying, we get

$$\begin{aligned}\hat{x}(k+1 | k) &= (\Phi - KC - \Gamma L + \Gamma L K_f C) \hat{x}(k | k-1) + (K - \Gamma L K_f) y(k) \\ u(k) &= (-L + L K_f C) \hat{x}(k | k-1) - L K_f y(k)\end{aligned}$$

We thus have

$$\begin{aligned}\Phi_c &= \Phi - KC - \Gamma L + \Gamma L K_f C \\ \Gamma_c &= K - \Gamma L K_f \\ C_c &= -L + L K_f C \\ D_c &= -L K_f\end{aligned}$$

```

b. public class LQGController {
    private Matrix Phic, Gammac, Cc, Dc;

```

```

private Matrix xhat;
private double y, preu = 0.0;

LQGController() {
    // create matrices, etc.
    ...
}

public void setParameters(Matrix Phi, Matrix Gamma, Matrix C,
                          Matrix L, Matrix K, Matrix Kf) {
    Phic = Phi-K*C-Gamma*L+Gamma*L*Kf*C;
    Gammac = K-Gamma*L*Kf;
    Cc = -L+L*Kf*C;
    Dc = -L*Kf;
}

public double calculateOutput(double y) {
    this.y = y;
    return preu + Dc*y;
}

public void updateState() {
    xhat = Phic*xhat + Gammac*y;
    preu = Cc*xhat;
}
}

```

4 a. SP: Set-point, reference value

MV: Measured variable

TR: Tracking signal

M: Manual control signal

b. The block diagram is shown in Figure 1

```

5. y := ADIn(ychan)
   e := yref - y
   D := ad * D - bd * (y - yold)
   v := K*(beta*yref - y) + I + D
   if mode = auto then u := sat(v,umax,umin)
   else u := sat(uman,umax,umin)
   DAOut(u,uchan)
   I := I + (K*h/Ti)*e + (h/Tr)*(u - v)
   if increment then uinc := 1
   elsif decrement then
       uinc := -1
   else
       uinc := 0
   uman := uman + (h/Tm) * uinc + (h/Tr) * (u - uman)
   yold := y

```

ad and *bd* are pre-calculated parameters given by the backward difference approximation of the D-term, i.e., as


```

private class ServoThread extends Thread {
    private long duration, T, t;
    private double h, u, x1 = 0.0, x2 = 0.0, newx1, newx2;

    ServoThread(long T) {
        this.T = T;
        h = 0.001*(double)T;
    }

    public void run() {
        t = System.currentTimeMillis();
        while(true) {
            // get input
            u = getU();
            // compute new states
            newx1 = x1 + h * (-1368.0 * x1 - 17.5 * x2 + 175.0 * u);
            newx2 = x2 + h * (5000.0 * x1 - 0.5 * x2);
            setY(newx2);
            x1 = newx1;
            x2 = newx2;

            t += T;
            duration = t - System.currentTimeMillis();
            if (duration < 1) duration = 1;
            try {
                sleep(duration);
            } catch (Exception e) {}
        }
    }
}

```

7. The delay margin is $49^\circ \frac{\pi}{180^\circ} / 1.7 = 0.50$ s.