Solutions to the exam in Real-Time Systems 150113

These solutions are available on WWW: *http://www.control.lth.se/course/FRTN01/*

1.

a. To verify that the system is observable we compute the observability matrix as

$$W_o = \begin{bmatrix} C \\ C\Phi \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \end{bmatrix}$$

which has full rank, since $\det\{W_o\} = 0 - 1/4 = -1/4$. Hence, the system is observable.

b. Letting $K = \begin{bmatrix} k_1 \\ k_2 \end{bmatrix}$ the gain of the deadbeat observer, we compute

$$\begin{aligned} \Phi - KC &= \begin{bmatrix} -1 & 0 \\ 1 & -1 \end{bmatrix} - \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} [0 \ \ \tfrac{1}{2}] = \\ &= \begin{bmatrix} -1 & 0 \\ 1 & -1 \end{bmatrix} - \begin{bmatrix} 0 & \frac{k_1}{2} \\ 0 & \frac{k_2}{2} \end{bmatrix} = \begin{bmatrix} -1 & -\frac{k_1}{2} \\ 1 & -1-\frac{k_2}{2} \end{bmatrix}. \end{aligned}$$

The characteristic polynomial can thus be computed as

$$\det\{zI - \Phi + KC\} = (z+1)\left(z + 1 + \frac{k_2}{2}\right) + \frac{k_1}{2} = z^2 + \left(2 + \frac{k_2}{2}\right) z + \frac{k_1}{2} + \frac{k_2}{2} + 1$$

The desired characteristic equation is

$$z^2 + p_1 z + p_2 = 0$$

with $p_1 = p_2 = 0$. Therefore, we can find the observer gain as

$$\begin{cases} 2 + \dfrac{k_2}{2} = 0 \\ \dfrac{k_1}{2} + \dfrac{k_2}{2} + 1 = 0 \end{cases} \quad \Rightarrow \quad \begin{cases} k_1 = 2 \\ k_2 = -4 \end{cases}$$

The resulting observer is thus

$$\hat{x}(k+1) = \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix} \hat{x}(k) + \begin{bmatrix} 2 \\ 0 \end{bmatrix} u(k) + \begin{bmatrix} 2 \\ -4 \end{bmatrix} y(k).$$

2. First of all we need to represent x and y in $Q5.2$-format

- x= 12.4 $\Rightarrow$ $X = \text{round}\left(12.4 \cdot 2^2\right) = \text{round}\,(49.6) = 50;$
- y= 0.4 $\Rightarrow$ $X = \text{round}\left(0.4 \cdot 2^2\right) = \text{round}\,(1.6) = 2;$

Then we can compute the value $Z = (X \cdot 2^n)/Y$

$$Z = (50 \cdot 2^2)/2 = 200/2 = 100$$

and the result z can be obtained as

$$\mathrm{z} = 100/2^2 = 25$$

When implementing the division using code it is necessary to use a 16 bit int to represent the intermediate value since 200 does not fit in an 8-bit signed int.

3. First note that the continuous-time system is unstable, so we obviously want the discretized system to also have this feature.

The transfer function of the system is given by:

$$G(s) = \frac{1}{s-1}$$

by substituting $s = \frac{q-1}{qh}$ where q is the forward-shift operator gives the discretized system $H(z)$ with backward Euler.

$$H(z) = \frac{qh}{q-1-qh} = [h=3] = \frac{3q}{-1-2q} = \frac{-3/2q}{q+1/2}$$

Hence, the discretized system is stable with an oscillating mode caused by the pole on the negative real axis, i.e., the behaviour is not at all the same as for the continuous-time system.

4.

$$\begin{aligned}
&p = 0.8 \\
&G_1 = 1/z && \rightarrow G_C \\
&G_2 = 1/(z+p) && \rightarrow G_D \\
&G_3 = z/(z+p) && \rightarrow G_A \\
&G_4 = 1/(z-p) && \rightarrow G_F \\
&G_5 = z/(z^2-p) && \rightarrow G_E \\
&G_6 = 1/(z^2+p) && \rightarrow G_B
\end{aligned}$$

G_{1C} This is a simple unit time delay.

G_{2D} G_D has a pole on the negative real axis and should thus have an oscillating pulse response. The pole excess is one, so a unit delay is expected. This matches only G_2

G_{3A} G_A is like G_D with an additional zero in $z = 0$. The pole excess is therefore zero and an oscillating pulse response without time delay is expected, which corresponds to G_3.

G_{4F} G_F has a pole on the positive real axis and a pole excess of one. A unit delay is therefore expected and apart from the initial injection of energy, a monotonically decreasing pulse response. This corresponds to G_4.

G_{5E} G_E has a double pole on the positive real axis and a pole excess of one. The pulse response should exhibit a unit time delay and be closely related to G_{4F}. The switch from z to z^2 in the denominator causes the output at every second time instant to be zero (no z^1 term present).

G_{6B} G_B has a pole excess of two and is therefore the only transfer function which exhibits a pulse response with two samples time delay.

p The constant is common to almost all transfer functions, and is most clearly visible in response G_4, where the first decrease from 1 to 0.8 matches $G_F = \frac{1}{z-0.8}$

5.

a. The ordinary RMS scheduling condition gives that

$$\frac{2}{4}+\frac{1}{3}+\frac{0.5}{6}=0.92>3(2^{1/3}-1)=0.78$$

Hence, using this we cannot tell whether the task set is schedulable or not. Using the hyperbolic condition leads to the same conclusion since

$$(\frac{2}{4}+1)(\frac{1}{3}+1)(\frac{0.5}{6}+1)=2.17>2$$

Finally, using response time analysis we have that

$$R_B^0=0, R_B^1=C_B=1\le D_B=4$$
$$R_A^0=0, R_A^1=C_A=2,$$
$$R_A^2=C_A+\left\lceil\frac{2}{T_B}\right\rceil C_B=3,$$
$$R_A^3=C_A+\left\lceil\frac{3}{T_B}\right\rceil C_B=3\le D_A=4$$
$$R_C^0=0, R_C^1=C_C=0.5,$$
$$R_C^2=C_C+\left\lceil\frac{0.5}{T_A}\right\rceil C_A+\left\lceil\frac{0.5}{T_B}\right\rceil C_B=C_C+C_A+C_B=3.5$$
$$R_C^3=C_C+\left\lceil\frac{3.5}{T_A}\right\rceil C_A+\left\lceil\frac{3.5}{T_B}\right\rceil C_B=C_C+C_A+2C_B=4.5$$
$$R_C^4=C_C+\left\lceil\frac{4.5}{T_A}\right\rceil C_A+\left\lceil\frac{4.5}{T_B}\right\rceil C_B=C_C+2C_A+2C_B=6.5>D_C=6$$

Hence, the task set is not schedulable.

b. Under EDF the schedulability condition is

$$\frac{2}{4}+\frac{1}{3}+\frac{0.5}{6}=0.92<1$$

Hence, the task set is schedulable under EDF scheduling.

c. The schedule is shown in Fig. 1.

d. Since the response time analysis for fixed priority scheduling does not apply to EDF, the response times are obtained from the drawn schedule.

The response time is the difference between the job release time and the job finishing time. The first job of Task A is released at $t=0$ and finishes at $t=3$. Hence, the response time is 3. The second job of Task A is released at $t=4$ and finishes at $t=6.5$. Hence, the response time is 2.5. The third job of Task A is released at $t=8$ and finishes at $t=11$. Hence, the response time is 3. Hence the worst-case response time for Task A is 3.

The first job of Task B is released at $t=0$ and finishes at $t=1$. Hence, the response time is 1. The second job of Task B is released at $t=3$ and finishes at $t=4$. Hence, the response time is 1. The third job of Task B is released at $t=6$ and finishes at $t=7.5$. Hence, the response time is 1.5.

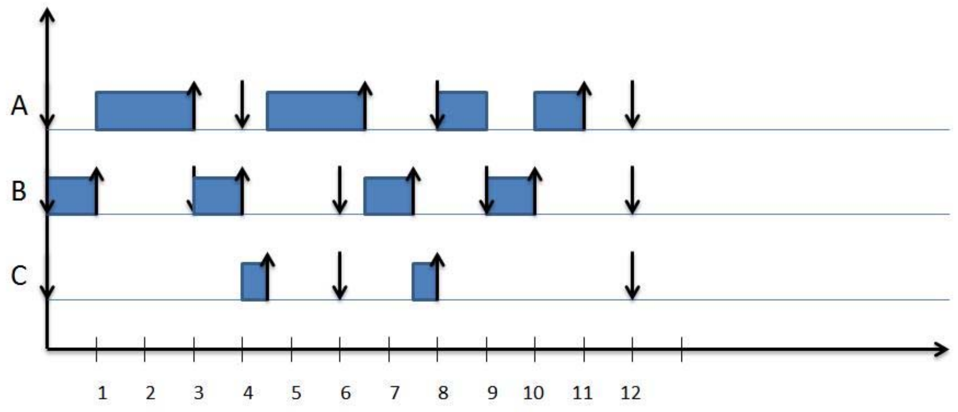

Figure 1 EDF schedule

The fourth job of Task B is released at $t = 9$ and finishes at $t = 10$. Hence, the response time is 1. Hence the worst-case response time for Task B is 1.5.

The first job of Task C is released at $t = 0$ and finishes at $t = 4.5$. Hence, the response time is 4.5. The second job of Task B is released at $t = 6$ and finishes at $t = 8$. Hence, the response time is 2. Hence the worst-case response time for Task C is 4.5.

6.

a. The closed loop pulse transfer function is given by:

$$H_{cl}(z) = \frac{K}{z^2 + 0.4z + K}$$

so the roots of the denominator polynomial are located in

$$\lambda_{1,2} = -0.2 \pm \sqrt{0.04 - K}.$$

For $0 < K \leq 0.04$ the poles are real-valued and within the unit circle and for $K > 0.04$ the poles become complex:

$$\lambda_{1,2} = -0.2 \pm i\sqrt{K - 0.04}.$$

The poles will cross the unit circle when $|\lambda_i| = \sqrt{0.2^2 + (K - 0.04)} = 1$ which happens when $K = 1$.

b. Studying the stationary equation

$$y + 0.4y = 0.5(1 - y)$$

gives $y = 5/19$, hence the stationary error is $14/19$.

c. The pulse-transfer function of the PI controller is given by the sum of the pulse-transfer functions of the P and I part:

$$P(z) = KE(z),\ I(k+1) = I(k) + \frac{Kh}{T_i}e(k),\ \Rightarrow\ I(z) = \frac{Kh}{T_i(z-1)}E(z).$$

Thus the controller transfer function $C(z)$ is given by:

$$C(z) = K + \frac{Kh}{T_i(z-1)} = \frac{T_iK(z-1)+Kh}{T_i(z-1)}.$$

The closed loop transfer function with a PI controller is given by:

$$H_{cl}(z) = \frac{\frac{T_iK(z-1)+Kh}{T_i(z-1)(z^2+0.4z)}}{1+\frac{T_iK(z-1)+Kh}{T_i(z-1)(z^2+0.4z)}} = \frac{T_iK(z-1)+Kh}{T_i(z-1)(z^2+0.4z)+T_iK(z-1)+Kh}$$

Given that $H_{cl}(z)$ is asymptotically stable, the static gain can be computed by inserting $z=1$, and clearly it is 1 since $\frac{Kh}{Kh}=1$.

7.

a.

$$G_a(s) = \frac{1}{s+1}$$
$$G_g(s) = 1 - \frac{1}{s+1} = \frac{s}{s+1}$$

b. From Table 3 in IFAC Professional Brief it immediately follow that

$$H_a(z) = \frac{1-e^{-h}}{z-e^{-h}}$$
$$H_g(z) = 1 - \frac{1-e^{-h}}{z-e^{-h}} = \frac{z-1}{z-e^{-h}}$$

c. To implement the filters H_a and H_g in code, they must first be transformed to a difference equation

$$\begin{aligned} y_f(k) &= H_a(z)y_a(k) + H_g(z)y_g(k) = \\ = y_f(k) &= \frac{1-e^{-h}}{z-e^{-h}}y_a(k) + \frac{z-1}{z-e^{-h}}y_g(k) \\ (z-e^{-h})y_f(k) &= (1-e^{-h})y_a(k) + (z-1)y_g(k) \\ y_f(k+1) - e^{-h}y_f(k) &= (1-e^{-h})y_a(k) + y_g(k+1) - y_g(k) \end{aligned}$$

The equation for the filtered output is then shifted back in time one sample to obtain

$$y_f(k) = e^{-h}y_f(k-1) + (1-e^{-h})y_a(k-1) + y_g(k) - y_g(k-1) \qquad (1)$$

The code to implement the filter is given below

```
double compFilt(double ya, double yg, double h){
    /* Declare and initialize variables */
    static double yf = 0;  // Filter state
    static double ygs = 0; // Gyroscope state
    static double yas = 0;  // Old accelerometer value
    double e = Math.exp(-h); // Filter constant
    double em1 = 1 - e;      // Filter constant


    /* Perform calculation */
    yf  = e*yf + em1*yas + yg - ygs; // Difference equation
    ygs = yg; // Update state
    yas = ya; // Update state

    return yf;
}
```

In a real good solution the filter constants should be pre-calculated.

8.

a. Using the formula

$$\omega = |(\omega_1 + \omega_N) \ mod \ \omega_s - \omega_N|$$

for the first frequency $\omega_1 = \pi$, $\omega_N = \pi$ and $\omega_s = 2\pi$, gives $\omega = \pi$. For the second frequency $\omega_2 = 3\pi/2$ the aliased frequency is $\pi/2$.

b. The first solution will work since the wanted signal will not be aliased by the high sample rate, however the disturbance will be aliased to 490. It will then be attenuated by the digital filter.

The second solution will not work since the sampling rate at 250 will alias the frequencies wanted from 175-250 Hz to 0 - 175 Hz.

However, in order for first solution to be a really good solution it should be complemented with an analog low-pass filter designed for the sampling frequency 1000 Hz.

9.

a. Without pre-calculations the code would look like

```
Read $y$ and $u_c$
$u_{ff} = -L_m x_m + l_r u_c$
$u = L(x_m - \hat{x}) + u_{ff}$
Output $u$
$\hat{x} = \Phi\hat{x} + \Gamma u + K(y - C\hat{x})$
$x_m = \Phi x_m + \Gamma u_{ff}$
```

Using pre-calculations the following is achieved:

```
Read $y$ and $u_c$
$u_{ff} = u_{ff} + l_r u_c$
$u = u + u_{ff}$
Output $u$
$\hat{x} = \Phi\hat{x} + \Gamma u + K(y - C\hat{x})$
$x_m = \Phi x_m + \Gamma u_{ff}$
$u_{ff} = -L_m x_m$
$u = L(x_m - \hat{x})$
```

b. Since the observer now has a direct term it must be placed in Calculate-Output part of the code, i.e.,

```
Read $y$ and $u_c$
$\hat{x} = (I - KC)(\Phi\hat{x} + \Gamma u) + Ky$
$u_{ff} = -L_m x_m + l_r u_c$
$u = L(x_m - \hat{x}) + u_{ff}$
Output $u$
$x_m = \Phi x_m + \Gamma u_{ff}$
```

However, a large part of the observer computations can be pre-calculated in UpdateState, as shown below.

```
Read $y$ and $u_c$
$\hat{x} = \hat{x} + Ky$
$u_{ff} = u_{ff} + l_r u_c$
$u = L(x_m - \hat{x}) + u_{ff}$
Output $u$
$\hat{x} = (I - KC)(\Phi\hat{x} + \Gamma u)$
$x_m = \Phi x_m + \Gamma u_{ff}$
$u_{ff} = -L_m x_m$
```

Note that we cannot precalculate parts of u any longer.

10.

a. The solution is

```
public class Writer extends Thread {

  MultiStepSemaphore sem;

  public Writer(MultiStepSemaphore s) {
    sem = s;
  }

  public void run() {
    while (true) {
      sem.take(3);
    // access critical section
      sem.give(3);
    }
  }
}

public class Reader extends Thread {

  MultiStepSemaphore sem;

  public Reader(MultiStepSemaphore s) {
    sem = s;
  }

  public void run() {
    while (true) {
      sem.take();
    // access critical section
      sem.give();
    }
  }
}

public class Main {

  public static void main(String[] args) {

  MultiStepSemaphore s;
  s = new MultiStepSemaphore(3);
  Writer w;
  Reader r;
  for (int i = 1; i==3; i++) {
    w = new Writer(s);
    w.start();
  }
  for (int i = 1; i==4; i++) {
    r = new Reader(s);
    r.start();
  }
}
```

b. The class `ReadersWritersGuard` is given below (with all exception handling excluded):

```
public class ReadersWritersGuard {

    int maxWriters = 1;
    int maxReader = 1;
// the number of writer processes inside the section
    int writersCounter = 0;
// the number of reader processes inside the section
    int readersCounter = 0;

    public ReadersWritersGuard() {}

    public ReadersWritersGuard(int maxW, int maxR) {
        this();
        maxWriters = maxW;
        maxReaders = maxR;
    }

    public synchronized void writersTake() {
        while ((writersCounter == maxWriters) | (readersCounter > 0)) {
            wait();
        }
        writersCounter++;
    }

    public synchronized void readersTake() {
        while ((readersCounter == maxReaders) | (writersCounter > 0) ) {
            wait();
        }
        readersCounter++;
    }

    public synchronized void writersGive() {
        writersCounter--;
        notifyAll();
    }

    public synchronized void readersGive() {
        readersCounter--;
        notifyAll();
    }
}
```

It should be used as

```
public class Writer extends Thread {

  ReadersWritersGuard sem;

  public Writer(ReadersWritersGuard s) {
    sem = s;
  }

  public void run() {
    while (true) {
      sem.writersTake();
    // access critical section
      sem.writersGive();
    }
  }
```

```
}

public class Reader extends Thread {

  ReadersWritersGuard sem;

  public Reader(ReadersWritersGuard s) {
    sem = s;
  }

  public void run() {
    while (true) {
      sem.readersTake();
    // access critical section
      sem.readersGive();
    }
  }
}

public class Main {

  public static void main(String[] args) {

  ReadersWritersGuard s;
  s = new ReadersWritersGuard(2,3);
  Writer w;
  Reader r;
  for (int i = 1; i==3; i++) {
    w = new Writer(s);
    w.start();
  }
  for (int i = 1; i==4; i++) {
    r = new Reader(s);
    r.start();
  }
}
```