Solutions to the exam in Real-Time Systems 140107

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

1.

a. Selecting the state variables as $[y(t)\ \dot{y}(t)\ \ddot{y}(t)]^T$ gives the state space representation

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

Since $A^3 = 0$ it is convenient to compute $\Phi = e^{Ah}$ using a series expansion, i.e.,

$$\Phi = e^{Ah} = I + Ah + A^2h^2/2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} h + \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} h^2/2$$

$$= \begin{bmatrix} 1 & h & h^2/2 \\ 0 & 1 & h \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1/2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\Gamma = \int_0^h e^{As} B\, ds = \int_0^h \begin{bmatrix} s^2/2 \\ s \\ 1 \end{bmatrix} ds = \begin{bmatrix} h^3/6 \\ h^2/2 \\ h \end{bmatrix} = \begin{bmatrix} 1/6 \\ 1/2 \\ 1 \end{bmatrix}$$

It is then straightforward to compute the pulse transfer from the expression

$$H(z) = C(zI - \Phi)^{-1}\Gamma = \frac{\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}}{(z-1)^3} \begin{bmatrix} (z-1)^2 & (z-1) & 1/2(z+1) \\ 0 & (z-1)^2 & (z-1) \\ 0 & 0 & (z-1)^2 \end{bmatrix} \begin{bmatrix} 1/6 \\ 1/2 \\ 1 \end{bmatrix}$$

$$= \frac{1/6z^2 + 2/3z + 1/6}{(z-1)^3}$$

The transfer function has three poles in 1 (as could be expected since this is what characterizes a discrete-time triple integrator) and two zeros, one in -3.7321 and one in -0.2679. The sampling, hence, turns a continuous-time minimum-phase system into a discrete-time non-minimum phase system (due to the zero outside the unit circle).

2.

a. The PI-controller consists of the P-part and I-part:

$$P(k) = Ke(k),\ I(k+1) = I(k) + (hK/T_i)e(k)$$

Applying the Z-transform gives

$$P(z) = KE(z),\ I(z) = \frac{hK}{T_i(z-1)}E(z)$$

This gives the pulse transfer function $H(z)$

$$U(z) = H(z)E(z) = (K + \frac{hK}{T_i(z-1)})E(z) = \frac{K(z + (h - T_i)/T_i)}{(z-1)}E(z)$$

The pulse transfer function has a pole in 1 and a zero in $1 - h/T_i$.

b.
```
private double u = 0, y = 0, r = 0; I = 0, e = 0;
private double K = ...;
private double h = ...;
private double Ti = ...;


while (1) {
  y = getY();
  r = getReference();

  e = r-y;
  u = K*e + I;

  outputU(u);

  I = I + (K*h/Ti)*e;

  sleep();
}
```

3. Feedback vector L_1 gives the closed-loop system matrix

$$\Phi_1 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

which has two eigenvalues in 1, hence corresponding to λ_C. Two eigenvalues on the unit circle means that the system response should be unstable, which agrees with I.

Feedback vector L_2 gives the closed-loop system matrix

$$\Phi_2 = \begin{pmatrix} -2 & -2 \\ 1 & 1 \end{pmatrix}$$

which has the eigenvalues -1 and 0, hence corresponding to λ_A. An eigenvalue in -1 means that the system response should be oscillating, which agrees with II.

Feedback vector L_3 gives the closed-loop system matrix

$$\Phi_3 = \begin{pmatrix} -1 & -1 \\ 1 & 1 \end{pmatrix}$$

which has two eigenvalues in 0, hence corresponding to λ_B. All eigenvalues in 0 implies deadbeat control, which agrees with III.

4 a. The system is sampled with $f_s = 50$ Hz, so the Nyquist frequency is 25 Hz. The disturbance component with a frequency of 20 Hz is below the Nyquist frequency and thus gives a control signal disturbance at 20 Hz.

The disturbance component with a frequency of 550 Hz is aliased down to exactly 0 Hz, where the second component of the control signal disturbance appears. (as a constant bias!). This is given by

$$\begin{aligned} f_1 &= |(550 + 25) \mathrm{mod} (50) - 25| \\ &= |575 \bmod 50 - 25| = |25 - 25| = 0 \end{aligned}$$

b. The frequencies 20 – 25 Hz are within the Nyquist frequency, and are not aliased. The frequencies 25 – 30 Hz are just outside the Nyquist frequency, and are aliased to 20 – 25 Hz. Thus the disturbances in the control signal will be in the range 20 – 25 Hz.

5 a. In order to guarantee that the task set is schedulable when the total number of tasks is unknown the total CPU utilization should be less than 69%. Since 39% are reserved for the existing applications this means that the three new tasks may use maximum 30% of the CPU. The utilization of the three tasks is

$$U = \sum_{i=1}^{i=3} \frac{C_i}{T_i} = \frac{0.5}{6} + \frac{1}{x} + \frac{1}{12}$$

Setting U equal to 0.3 gives $x = 7.5$.

b. Under EDF the schedulability bound is 1. Using the above formulae gives $x = 1.2$.

c. Using the above formulae again gives $x = 1.5$.

d. The utilization is 0.83333 which is larger than

$$3(2^{\frac{1}{3}} - 1) = 0.779.$$

Hence the ordinary sufficient-only schedulability test is not enough. However, the hyperbolic schedulability condition is fulfilled, i.e.,

$$(0.5/6 + 1)(1/1.5 + 1)(1/12 + 1) = 1.956 < 2$$

Hence, the task set is schedulable.

Alternatively one can calculate the response times for the tasks. The priority assignment gives Task B the highest priority, Task A the medium priority, and Task C the lowest priority.

Since Task B has the highest priority the response time for task B, R_B, is equal to 1. This is smaller than the deadline for task B. So far so good.

$$R_A^0 = 0, R_A^1 = C_A = 0.5,$$

$$R_A^2 = C_A + \left\lceil \frac{0.5}{T_B} \right\rceil C_B = 1.5,$$

$$R_A^3 = \ldots = 1.5$$

This is also less than the deadline. For Task C

$$R_C^0 = 0, R_C^1 = C_C = 1,$$

$$R_C^2 = C_C + \left\lceil \frac{1}{T_A} \right\rceil C_A + \left\lceil \frac{1}{T_B} \right\rceil C_B = C_C + C_A + C_B = 2.5$$

$$R_C^3 = \ldots = 3.5, R_C^4 = \ldots = 4.5, R_C^5 = \ldots = 4.5$$

This, finally, is also less than the deadline. Hence, the task set is schedulable.

6.

a. From the formula sheet it immediately follows that

$$H(z) = \frac{1 - e^{-ah}}{z - e^{-ah}}$$

b. A time delay equal to one sample is equivalent to an extra pole in the origin, i.e.,

$$H(z) = \frac{1 - e^{-ah}}{z(z - e^{-ah})}$$

c. Sampling the continuous-time system gives that

$$x(kh + h) = \Phi x(kh) + \Gamma_0 u(kh) + \Gamma_1 u(kh - h)$$

where

$$\Phi = e^{-ah}$$

$$\Gamma_0 = \int_0^{h-\tau} e^{-as}\, ds\; a = 1 - e^{-a(h-\tau)}$$

$$\Gamma_1 = e^{-a(h-\tau)} \int_0^{\tau} e^{-as}\, ds\; a = e^{-a(h-\tau)} - e^{-ah}$$

The corresponding state-space model is obtained from the augmented state $[x(kh)\; u(kh-h)]^T$ as

$$\begin{pmatrix} x(kh+h) \\ u(kh) \end{pmatrix} = \begin{pmatrix} \Phi & \Gamma_1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x(kh) \\ u(kh-h) \end{pmatrix} + \begin{pmatrix} \Gamma_0 \\ I \end{pmatrix} u(kh)$$

The corresponding pulse tranfer function is given by

$$H(z) = \frac{\Gamma_0 z - \Gamma_1}{z(z - \Phi)} = \frac{(1 - e^{-a(h-\tau)})z - (e^{-a(h-\tau)} - e^{-ah})}{z(z - e^{-ah})}$$

7. The corresponding Grafcet diagram is shown in Fig. 1.

8 a. The problem is that the lock is held while waiting for user confirmation, which means that the controller cannot execute until the parameter change has been reviewed.

```
public void setParameters(PIDParameters p) {
  System.out.println("Got new parameters:");
  printParameters();
  System.out.println("Apply these parameters? (Y/N)");
  if (getUserConfirmation()) {
    synchronized (this) {
      this.p = (PIDParameters)p.clone();
      // ... update ad and bd
    }
  }
}
```

8 b. Some additional issues and possible solutions are:

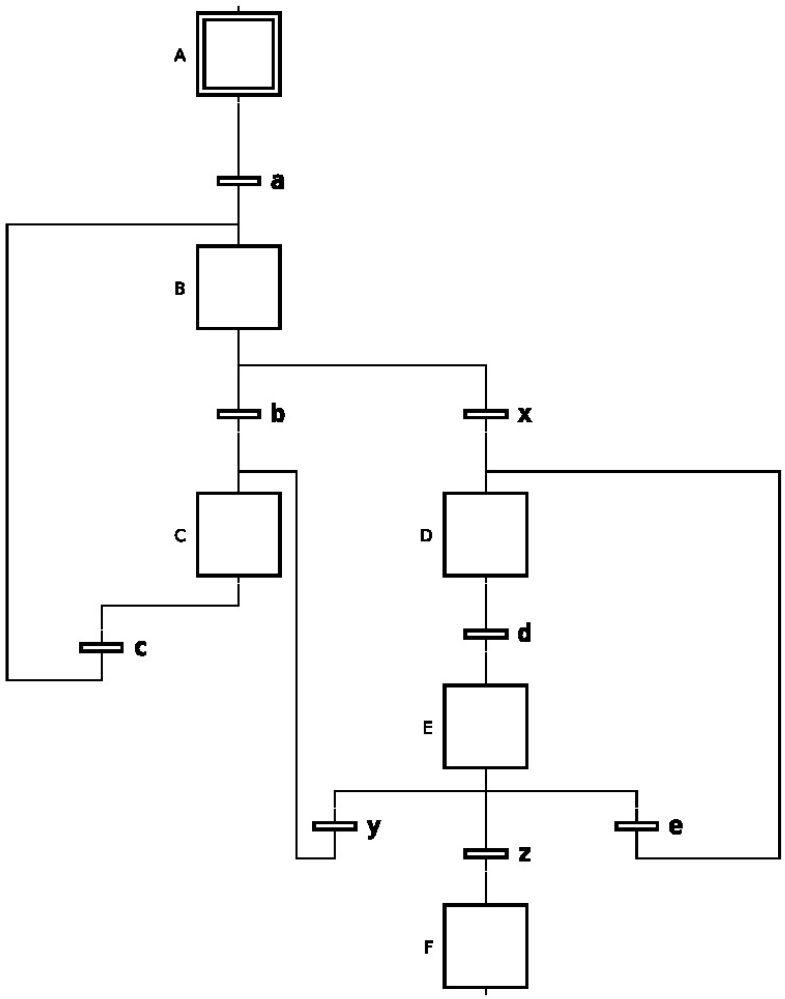

Figure 1 Grafcet diagram

- *Problem*: Parameters must be confirmed when creating PID instances.
  Solution: Use a separate private method for just setting the parameters and use that from both setParameters() and the constructor.
- *Problem*: The GUI freezes while waiting for user confirmation. This is especially critical if the getUserConfirmation() also uses the GUI as the GUI will then freeze indefinitely.
  Solution: Schedule parameter update in separate threads.
- *Problem*: Calls to `setPIDParameters()` may be interleaved in the output.
  Solution: Use a separate lock for this.

9. A. `fixX * fixY` has `nx` fractional bits. `fixZ` is first shifted from `nz` to `nx` fractional bits. Then the two terms are added and the `nx` fractional bits are shifted away.

$$\frac{x \cdot 2^{nx} \cdot y + z \cdot 2^{nz} \cdot 2^{nx-nz}}{2^{nx}} = \frac{x \cdot y \cdot 2^{nx} + z \cdot 2^{nx}}{2^{nx}} = x \cdot y + z$$

The calculation is correct.

B. The `nx` fractional bits of `fixX` is shifted away before they are used. Precision is lost, and the calculation is incorrect.

C. `fixX * fixY` has `nx` fractional bits. The term is shifted down to `nz` fractional bits before it is added to `fixZ`. The result of the addition is then shifted down to zero fractional bits.

$$\frac{\frac{x \cdot 2^{nx} \cdot y}{2^{nx-nz}} + z \cdot 2^{nz}}{2^{nz}} = \frac{x \cdot y \cdot 2^{nz} + z \cdot 2^{nz}}{2^{nz}} = x \cdot y + z$$

The calculation is correct.

D. Adding a variable with zero fractional bits with a variable with non-zero fractional bits makes no sense:

$$\frac{\frac{x \cdot 2^{nx} \cdot y}{2^{nx}} + z \cdot 2^{nz}}{2^{nz}} = \frac{x \cdot y}{2^{nz}} + z \neq x \cdot y + z$$

The calculation is incorrect.

10. a) is syntactically correct, deadlock free, live, and bounded (always 1 token in the net).

b) is syntactically correct and bounded (at most 1 token). The net always deadlocks when the bottom right transition fires. Not live since not deadlock free.

c) is not syntactically correct since it has an arc between two places.

d) is syntactically correct, deadlock free, and live. After the source transition has fired once, both transitions are always enabled. Nets with source transitions are not bounded.

e) is syntactically correct, deadlock free, and live. After firing the upper transition, the lower place will always contain at least one token. Then it is always possible to fire lower and then upper, thus the net is live and deadlock free. Each such firing increases the number of tokens in the net by one and thus it is not bounded.

f) is not syntactically correct since it has an arc between two transitions.

g) is syntactically correct and deadlock free. After firing the upper transition, that transition can never be fired again, thus the net is not live. Then the lower transition can always be fired, thus the net is deadlock free. Doing so increases the number of tokens in the net by one, thus the net is not bounded.

11.
```
public class CountDownLatch {

  private int count;

  public CountDownLatch(int count) {
    this.count = count;
  }

  public synchronized void await() throws InterruptedException {
    while (count > 0) {
```

```
        wait();
      }
    }

    public synchronized void countDown() {
      if (count > 0) {
        count--;
        if (count == 0) {
          notifyAll();
        }
      }
    }

    public synchronized int getCount() {
      return count;
    }
  }
```

The inverse of an upper-triangular 3x3 matrix

If

$$A = \begin{bmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{bmatrix}$$

then, assuming that the matrix is invertible, the inverse is given by

$$A^{-1} = \frac{1}{adf} \begin{bmatrix} df & -bf & (be - cd) \\ 0 & af & -ae \\ 0 & 0 & ad \end{bmatrix}$$