Solutions to the exam in Real-Time Systems 130402

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

1.

a. In order for the controller to have integral action it must have a pole in $z = 1$. This is the case if $k_1 = 1$.

b. In order for the controller to have derivative action it must have a zero in $z = 1$. This is the case if $k_3/k_2 = -1$.

2.

a. Assuming zero initial condition and taking $\mathcal{Z}$-transform gives

$$Y(z) = \frac{1}{z^2 + 0.5} U(z).$$

The poles are located in $p_{1,2} = \pm i/\sqrt{2}$.

b. With $x_1(k) = y(k)$ and $x_2(k) = y(k+1)$ we get

$$\begin{pmatrix} x_1(k+1) \\ x_2(k+1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -0.5 & 0 \end{pmatrix} \begin{pmatrix} x_1(k) \\ x_2(k) \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u.$$

$$y(k) = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} x_1(k) \\ x_2(k) \end{pmatrix}.$$

c. Put $y(k+2) = y(k)$. Then we get that $1.5y(k) = u(k)$, meaning that the stationary gain is $1/1.5$.

3.

a. b and c are the setpoint weights for the proportional and derivative parts, respectively. They are used to tune the setpoint response of the controller. Kb is the tracking constant for the anti-windup, normally expressed as $\frac{1}{T_t}$

b. The integrator and gain blocks implement a lowpass filter for the derivative part. N is the maximum derivative gain.

4 a. The approximate analysis cannot be used since we have $D_i < T_i$ for at least one task.

4 b. With rate monotonic priority assignment we get the priorities: A - high, B - medium, C - low.

The exact analysis method gives:

$R_A = 1 \leq 2$ (OK)

$$\begin{aligned} R_B^0 &= 3 \\ R_B^1 &= 3 + \left\lceil \frac{3}{3} \right\rceil 1 = 4 \\ R_B^2 &= 3 + \left\lceil \frac{4}{3} \right\rceil 1 = 5 \\ R_B^3 &= 3 + \left\lceil \frac{5}{3} \right\rceil 1 = 5 \end{aligned}$$

$R_B = 5 \le 6$ (OK)

$$\begin{aligned}
R_C^0 &= 2 \\
R_C^1 &= 2 + \left\lceil \frac{2}{3} \right\rceil 1 + \left\lceil \frac{2}{7} \right\rceil 3 = 6 \\
R_C^2 &= 2 + \left\lceil \frac{6}{3} \right\rceil 1 + \left\lceil \frac{6}{7} \right\rceil 3 = 7 \\
R_C^3 &= 2 + \left\lceil \frac{7}{3} \right\rceil 1 + \left\lceil \frac{7}{7} \right\rceil 3 = 8 \\
R_C^4 &= 2 + \left\lceil \frac{8}{3} \right\rceil 1 + \left\lceil \frac{8}{7} \right\rceil 3 = 11
\end{aligned}$$

Although R_C has not converged yet, we know already that it is larger than the deadline 10, i.e., the set is not schedulable using rate monotonic fixed-priority scheduling.

4 c. No, the priorities are the same as for rate monotonic priority assignment and thus the results from **??** apply to deadline monotonic assignment as well.

5.

a. Since models are only approximations there is a large probability that there actually will exist frequencies slightly larger than f_0. Thus, a slightly larger sampling frequency should be chosen.

b. Denote the sampling frequency with f_s. By applying the Shannon sampling theorem we get:

- For $f_s \ge 6f_0$ no part of the signal will be aliased.
- For $4f_0 \le f_s < 6f_0$ the disturbance will be aliased to outside $\pm f_0$.
- For $2f_0 \le f_s < 4f_0$ the disturbance will be aliased into the frequency interval $\pm f_0$
- For $f_0 \le f_s < 2f_0$ the signal will be aliased to outside $\pm f_0$.

6.

a. The `Worker` threads should ask for new job parts and execute them as long as there are any left. Then their `run` method should terminate, so that the `join` call in `Pool.runParallel` can finish. A suitable implementation is

```
private class Worker extends Thread {
  public void run() {
    int myPart = getNextPart();
    while (myPart != -1) {
      job.doPart(myPart);
      myPart = getNextPart();
    }
  }
}
```

b. The critical portion of the code is the `getNextPart` method, which assigns the job parts. This method should be synchronized in order to guarantee that each job part is assigned exactly once.

c. As the `Pool` class initializes all its state from scratch in the `runParallel` method, there is no problem in calling `runParallel` multiple times in sequence. The current implementation is however not thread safe, since if one thread calls `runParallel` while it is already running in another thread, both calls will try to use the same `job`, `nextPart`, and `numPart` variables.

7.

a. We start by writing the continuous-time system on state-space form, i.e.,

$$\begin{aligned} dx(t)/dt &= -2x(t) + 2u(t) \\ y(t) &= x(t) \end{aligned}$$

The computational delay is equivalent to a constant input delay, i.e., the continuous-time system will be

$$\begin{aligned} dx(t)/dt &= -2x(t) + 2u(t-L) \\ y(t) &= x(t) \end{aligned}$$

The ZOH-sampled equivalent of this, assuming that $L \leq h$ is

$$\begin{aligned} x(kh+h) &= \Phi x(kh) + \Gamma_0 u(kh) + \Gamma_1 u(kh-h) \\ y(kh) &= x(kh) \end{aligned}$$

where

$$\begin{aligned} \Phi &= e^{-2h} = e^{-1} \\ \Gamma_0 &= 2\int_0^{h-L} e^{-2s} ds = 1 - e^{2L-1} \\ \Gamma_1 &= 2e^{-2(h-L)}\int_0^{L} e^{-2s} ds = e^{2L-1} - e^{-1} \end{aligned}$$

Applying the control law $u(k) = -2y(k) = -2x(k)$ gives the closed loop system

$$x(k+1) = e^{-1}x(k) - 2(1-e^{2L-1})x(k) - 2(e^{2L-1} - e^{-1})x(k-1)$$

The characteristic equation is hence

$$z^2 + (2(1-e^{2L-1}) - e^{-1})z + 2(e^{2L-1} - e^{-1})$$

Introducing $\omega = e^{2L-1}$, the conditions for stability can be written

$$\begin{aligned} &2(\omega - e^{-1}) < 1 \\ &2(\omega - e^{-1}) > -1 + (2(1-\omega) - e^{-1}) \\ &2(\omega - e^{-1}) > -1 - (2(1-\omega) - e^{-1}) \end{aligned}$$

From this follows that

$$\omega < \frac{1}{2} + e^{-1}$$
$$\omega > \frac{1 + e^{-1}}{4}$$

The first inequality leads to

$$2L - 1 < log(1/2 + e^{-1}) = -0.1417$$

from which follows that

$$L < 0.4291$$

From the second inequality we have that

$$2L - 1 > log(1/4 + e^{-1}/4) = 0.3420$$

from which follows that

$$L > 0.6710$$

However, since we have already assumed that $L < h$ the second solution can be disregarded. Hence, the system is stable if $L < 0.4291$.

8.

a. The system can be written as

$$U(s) = K\beta R(s) - KY(s) + I(s)$$
$$I(s) = \frac{K}{sT_i}(R(s) - Y(s))$$

The approximation $s \approx (z-1)/h$ gives

$$(z-1)I(z) = \frac{Kh}{T_i}(R(z) - Y(z)) \Longleftrightarrow$$
$$I(k+1) = I(k) + \frac{Kh}{T_i}(r(k) - y(k))$$

The whole controller is given by

$$u(k) = K\beta r(k) - Ky(k) + I(k)$$
$$I(k+1) = I(k) + \frac{Kh}{T_i}(r(k) - y(k))$$

b. The coefficients to be converted are

$$K = 5$$
$$K\beta = 3.15$$
$$Kh/T_i = 1.66667$$

K requires 3 integer bits, giving $n = 16 - 1 - 3 = 12$ fractional bits. The fixed-point representations are

$$K_{[3.12]} = \text{round}(5 \cdot 2^{12}) = 20480$$
$$K\beta_{[3.12]} = \text{round}(3.15 \cdot 2^{12}) = 12902$$
$$Kh/T_{i[3.12]} = \text{round}(1.667 \cdot 2^{12}) = 6827$$

c.

```
// define your parameters here
#define K      20480
#define Kb     12902
#define KhTi    6827

int32_t I = 0;

// Called periodically every 0.1 s.
void do_control(int16_t r) {
  int16_t y = read_input();
  int32_t u = ((int32_t)Kb*r - (int32_t)K*y + I) >> 12;

  // Limit the signal
  if (u > 511) {
    u = 511;
  } else if (u < -512) {
    u = -512;
  }

  write_output(u);

  // Note: no shifting here
  I = I + (int32_t)KhTi*(r-y);
}
```