Solutions to the exam in Real-Time Systems 190503

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

1.

a. With the state feedback control law $u(k) = -Lx(k)$ the closed loop system will be

$$\begin{aligned} x(k+1) &= (\Phi - \Gamma L)x(k) \\ y(k) &= Cx(k) \end{aligned}$$

The characteristic polynomial of the closed loop system is

$$z^2 + (0.2l_1 + 0.3l_2 - 1.9)z - 0.1l_1 - 0.18l_2 + 0.76$$

Comparing this with the desired deadbeat characteristic polynomial z^2 gives $L = (18.995 \quad -6.33)$.

b. The result is obtained by instead using the control law $u(k) = L \cdot (x_{ref} - x(k))$ with the same L as before.

2.

a. Since A is nilpotent ($A^2 = 0$), the Taylor series expansion method is straightforward.

$$\Phi = e^{Ah} = I + Ah + \frac{(Ah)^2}{2} + ... = \begin{pmatrix} 1 & 0 \\ 3h & 1 \end{pmatrix}$$

$$\Gamma = \begin{pmatrix} 0 \\ h \end{pmatrix}$$

b. Here the Laplace transform method is the most convenient.

$$\Phi = e^{Ah} = \mathcal{L}^{-1}(sI - A)^{-1} = \begin{pmatrix} 1 & \frac{1}{4}(1 - e^{-4h}) \\ 0 & e^{-4h} \end{pmatrix}$$

$$\Gamma = \frac{1}{4} \begin{pmatrix} h + \frac{1}{4}e^{-4h} - \frac{1}{4} \\ 1 - e^{-4h} \end{pmatrix}$$

3.

a. Since the process contains two integrators there is no need to also have an integrator in the controller. The closed loop system will both follow step setpoint changes and step load disturbances without any stationary error.

b. If you can measure the full state vector and there is no measurement noise then there will be no difference. If you cannot measure the state vector then you will have to implement it using an observer. The dynamics of the observer will cause a small difference during transients. The observer will also act as a noise filter. Using a PD controller you need to take the derivative of the process output which can be quite noise sensitive. In the PD controller you normally include a low-pass filter on the derivative term.

4.

a. The CPU utilization is

$$\sum_i \frac{C_i}{T_i} = \frac{3}{7} + \frac{1}{3} + \frac{3}{15} \approx 0.96 < 1$$

b. The sufficient conditions for rate monotonic schedulability are only valid when $D_i = T_i$, so we need to do an exact analysis. The priorities are in descending order B, A, C. The worst case response times R_i for the different tasks are

$$R_B = C_B = 1 \leq D_B = 2$$

$$\begin{aligned}
R_A^1 &= C_A = 3 \\
R_A^2 &= C_A + \left\lceil \frac{R_A^1}{T_B} \right\rceil C_B = 3 + \left\lceil \frac{3}{3} \right\rceil 1 = 4 \\
R_A^3 &= C_A + \left\lceil \frac{R_A^2}{T_B} \right\rceil C_B = 3 + \left\lceil \frac{4}{3} \right\rceil 1 = 5 \\
R_A^4 &= C_A + \left\lceil \frac{R_A^3}{T_B} \right\rceil C_B = 3 + \left\lceil \frac{5}{3} \right\rceil 1 = 5 \leq D_A = 5
\end{aligned}$$

$$\begin{aligned}
R_C^1 &= C_C = 3 \\
R_C^2 &= C_C + \left\lceil \frac{R_C^1}{T_A} \right\rceil C_A + \left\lceil \frac{R_C^1}{T_B} \right\rceil C_B = 3 + \left\lceil \frac{3}{7} \right\rceil 3 + \left\lceil \frac{3}{3} \right\rceil 1 = 7 \\
R_C^3 &= C_C + \left\lceil \frac{R_C^2}{T_A} \right\rceil C_A + \left\lceil \frac{R_C^2}{T_B} \right\rceil C_B = 3 + \left\lceil \frac{7}{7} \right\rceil 3 + \left\lceil \frac{7}{3} \right\rceil 1 = 9 \\
R_C^4 &= C_C + \left\lceil \frac{R_C^3}{T_A} \right\rceil C_A + \left\lceil \frac{R_C^3}{T_B} \right\rceil C_B = 3 + \left\lceil \frac{9}{7} \right\rceil 3 + \left\lceil \frac{9}{3} \right\rceil 1 = 12 \\
R_C^5 &= C_C + \left\lceil \frac{R_C^4}{T_A} \right\rceil C_A + \left\lceil \frac{R_C^4}{T_B} \right\rceil C_B = 3 + \left\lceil \frac{12}{7} \right\rceil 3 + \left\lceil \frac{12}{3} \right\rceil 1 = 13 \\
R_C^6 &= C_C + \left\lceil \frac{R_C^5}{T_A} \right\rceil C_A + \left\lceil \frac{R_C^5}{T_B} \right\rceil C_B = 3 + \left\lceil \frac{13}{7} \right\rceil 3 + \left\lceil \frac{13}{3} \right\rceil 1 = 14 \\
R_C^7 &= C_C + \left\lceil \frac{R_C^6}{T_A} \right\rceil C_A + \left\lceil \frac{R_C^6}{T_B} \right\rceil C_B = 3 + \left\lceil \frac{14}{7} \right\rceil 3 + \left\lceil \frac{14}{3} \right\rceil 1 = 14 > D_C = 8
\end{aligned}$$

Hence the deadline for task C can not be guaranteed to be met.

5.

a. The transfer function to be discretized is

$$(k_p + \frac{k_i}{s} + k_d s) = \frac{k_p s + k_i + k_d s^2}{s}$$

Replacing s with $(q-1)/h$ generates the following difference equation

$$(q-1)u(k) = (k_d/hq^2 + (k_p - k_d/h)q + (k_i h - k_p + k_d/h))e(k)$$

Already at this stage we can see that the resulting system is noncausal and if we try to implement it then $u(k)$ will depend on $e(k+1)$ which is not realizable.

b. Replacing s with $(q-1)/qh$ we instead get the difference equation

$$(q^2-q)u(k) = ((k_p + k_i h)q^2 + (k_d/h - k_p)q - k_d/h)e(k)$$

which is causal. This is equivalent to

$$u(k) = u(k-1) + (k_p + k_i h)e(k) + (k_d/h - k_p)e(k-1) - k_d/he(k-2)$$

Using the code template we get the following implementation:

```
CalculateOutput:
  e = r - y;
  u =  u + (kp + ki*h)*e + eold;

UpdateState:
  e2 = e1;
  e1 = e;
  eold = (kd/h - kp)*e1 - (kd/h)*e2;
```

To further decrease the latency we can also precalculate `(kp + ki*h)` and make sure that this is updated only when any of the involved parameters are updated.

6.

a. The call to `Schedule` will always lead to a context switch (assuming that the kernel has an Idle process doing busy-wait).

b. The call to `Schedule` will lead to a context switch if the process (thread) moved from the semaphore waiting queue to ReadyQueue has higher priority than the currently running process (thread).

c. The call to `Schedule` will lead to a context switch in two cases. First, if a process (thread) has been moved from TimeQueue to ReadyQueue and this process (thread) has higher priority than the currently executing one. Second, if the currently executing process (thread) has executed longer than its round-robin time-slice and there is another ready process (thread) that has the same priority.

7. The first error is that none of the operands is cast to 16 bits in the multiplication. The result of this is that the multiplication will be performed with 8 bits rather than 16 bits, which results in loss of precision.

The second error is that instead of adding 0.5 before right shifting 1.0 is added, i.e., no interpolation is performed and the result of the multiplication will be incorrect. The line `temp = temp + (1 << n);` should instead be `temp = temp + (1 << n-1);`

The third error is that the intermediate 16 bit result of the multiplication is assigned to the 8 bit Z variable before the test for overflow is performed. The

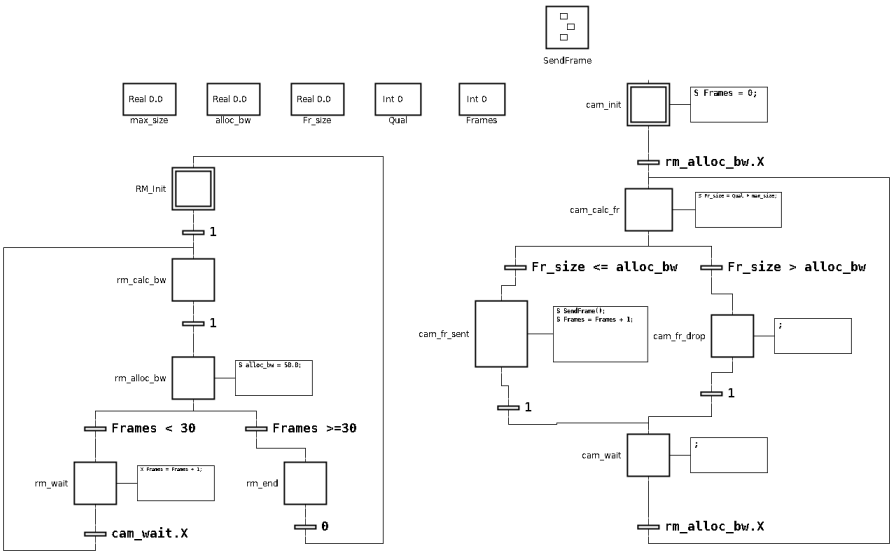

Figure 1 Solution to the Grafcet problem.

consequence of this is that the overflow and possible wraparound will have occurred before the overflow test is performed. Since it is impossible for Z to be larger than `INT8_MAX` or smaller than `INT8_MIN` the overflow test will have no effect.

8. One solution is shown in Figure 1.

9 a. The sampling interval, H, in the code will always be 0. This means that the PI controller will execute as a P-Controller. The Regul thread will never sleep. This means that the sampling interval of the P-controller will be very short and will depend on the speed of the computer.

b. Kalle thought that he had implemented a PI controller which should remove the stationary error. However, with the P controller that he in fact had implemented the stationary error was not removed.

c. With a real-time JVM Regul will take all the CPU time and no other threads will run, i.e. starvation. With an ordinary JVM, the execution of the Java threads is mapped down to the underlying OS threads, e.g., Linux threads. These threads normally execute at the same priority and employ time-sharing. The effect of this is that the threads with lower priority than Regul will still execute, albeit possibly with a slower speed.