Solutions to the exam in Real-Time Systems 150603

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

1.

a. The control law for state feedback is $u = l_r r(k) - Lx(k)$, where $L = \begin{bmatrix} l_1 & l_2 \end{bmatrix}$. The given system thus becomes:

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

The poles are given by the characteristic polynomial:

$$\begin{aligned} \det(zI - \Phi + \Gamma L) &= \det\left(\begin{bmatrix} z & 1 \\ -1 & z \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ l_1 & l_2 \end{bmatrix} \right) = \\ &= \det \begin{bmatrix} z & 1 \\ l_1 - 1 & z + l_2 \end{bmatrix} = z(z + l_2) - l_1 + 1 = \\ &= z^2 + l_2 z - l_1 + 1 = 0 \end{aligned}$$

Which gives $l_1 = 1$ and $l_2 = 0$, and thus $L = \begin{bmatrix} 1 & 0 \end{bmatrix}$.
In order to have unit static gain then

$$l_r = \frac{1}{C(I - \Phi + \Gamma L)^{-1}\Gamma} = -1$$

b. An observer is given as

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

Where $K = \begin{bmatrix} k_1 & k_2 \end{bmatrix}^T$. The poles of the observer are determined by the eigenvalues of $(\Phi - KC)$, which should be located at $z = 1/2$. The characteristic polynomial is thus given by:

$$\begin{aligned} \det(zI - \Phi + KC) &= \det\left(\begin{bmatrix} z & 0 \\ 0 & z \end{bmatrix} - \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} k_1 & 0 \\ k_2 & 0 \end{bmatrix} \right) = \\ &= \det \begin{bmatrix} z + k_1 & 1 \\ k_2 - 1 & z \end{bmatrix} = z(z + k_1) - k_2 + 1 = \\ &= z^2 + k_1 z - k_2 + 1 \\ &\Rightarrow \begin{cases} k_1 = -1 \\ k_2 = 3/4 \end{cases} \end{aligned}$$

The observer is then given by:

$$\hat{x}(k+1) = \begin{bmatrix} 1 & -1 \\ 1/4 & 0 \end{bmatrix} \hat{x}(k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(k) + \begin{bmatrix} -1 \\ 3/4 \end{bmatrix} y(k)$$

2. The Laplace transfer function of the system is

$$G(s) = C(sI - A)^{-1}B = \frac{3}{(s+1)(s-2)} = -\frac{1}{2}\frac{-2}{s-2} - \frac{1}{s+1}.$$

The two terms can be seen as two parallel systems and we can sample them separately and add the results. The zoh-sampled system is then found in the formula sheet Zero-order hold table as

$$H(z) = -\frac{1}{2}\frac{1-e^{2h}}{z-e^{2h}} - \frac{1-e^{-h}}{z-e^{-h}}.$$

The problem can also be solved with the Laplace transform method. Φ is calculated as

$$\Phi = \mathcal{L}^{-1}\{(sI-A)^{-1}\}\Big|_{t=h} = \begin{bmatrix} e^{-h} & \frac{e^{2h}-e^{-h}}{3} \\ 0 & e^{2h} \end{bmatrix}$$

Γ can then be found using

$$\Gamma = \int_0^h e^{As}B\,\mathrm{d}s = \begin{bmatrix} \frac{(e^h-1)2(2e^{-h}+1)}{3} \\ e^{2h}-1 \end{bmatrix}$$

And to get the pulse-transfer function we do

$$H(z) = C(zI-\Phi)^{-1}\Gamma = \frac{\left(3e^h - e^{3h} - 2\right)z + \left(3e^{2h} - 2e^{3h} - 1\right)}{2e^h z^2 + \left(-2e^{3h} - 2\right)z + 2e^{2h}}$$

3.

a. To find $y(3)$ we iterate the system equation.

$$x(1) = \Phi x(0) + \Gamma u(0) = \begin{bmatrix} 0.9 & 0.8 \\ 0.5 & 0.0 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$x(2) = \Phi x(1) + \Gamma u(1) = \begin{bmatrix} 0.9 & 0.8 \\ 0.5 & 0.0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}1 = \begin{bmatrix} 1.9 \\ 0.5 \end{bmatrix}$$

$$x(3) = \Phi x(2) + \Gamma u(2) = \begin{bmatrix} 0.9 & 0.8 \\ 0.5 & 0.0 \end{bmatrix}\begin{bmatrix} 1.9 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}1 = \begin{bmatrix} 3.11 \\ 0.95 \end{bmatrix}$$

Then we calculate the output of the system using

$$y(3) = Cx(3) = \begin{bmatrix} 1 & 1 \end{bmatrix}\begin{bmatrix} 3.11 \\ 0.95 \end{bmatrix} = 4.06$$

b. To find out if the system will converge we must check that it is stable. A discrete linear time-invariant system is stable if the poles of the pulse transfer function (or equivalently the eigenvalues of the dynamics matrix) all are inside the unit circle. The pulse transfer function of the system is

$$H(z) = C(zI-\Phi)^{-1}\Gamma = \frac{z+0.5}{z^2 - 0.9z - 0.4}.$$

The system has one pole outside of the unit circle and will not converge.

4.

a. The EDF schedulability condition is that the utilization

$$U = \sum_i \frac{C_i}{T_i} \leq 1.$$

The current utilizations $U_A = 0.7, U_B = 0.2$ and $U_C = 0.4$ add up to $U = 1.3$, clearly not schedulable. Thus the utilization has to be lowered by 0.3.

To reduce the utilization by ΔU, task i has to be made N times faster,

$$N = \frac{C_i}{C_i'} = \frac{U_i}{U_i'} = \frac{U_i}{U_i - \Delta U}.$$

This means that task A has to be made 1.75 times or that task C has to be made 4 times faster. Since $U_B < \Delta U$, it is impossible to make the task set schedulable by just optimizing task B.

b. If we make T_C 4 times longer, it has the same effect on the utilization as to make the code run 4 times faster.

5.

a. The parameters of the different matrices are given by $\text{round}(M \cdot 2^4)$ where M is either Φ, Γ or C. Hence

$$x(k+1) = \begin{bmatrix} 6 & 68 \\ 1 & 10 \end{bmatrix} x(k) + \begin{bmatrix} 16 \\ 32 \end{bmatrix} u(k)$$

$$y(k) = \begin{bmatrix} 20 & 33 \end{bmatrix} x(k).$$

b. Not a good choice! The approximated system is given by converting back to reals, i.e., by multiplying each parameter with 2^{-4}. This gives

$$x(k+1) = \begin{bmatrix} 0.375 & 4.25 \\ 0.0625 & 0.625 \end{bmatrix} x(k) + \begin{bmatrix} 1 \\ 2 \end{bmatrix} u(k)$$

$$y(k) = \begin{bmatrix} 1.25 & 2.0625 \end{bmatrix} x(k).$$

The original system has poles in $\text{eig}(\Phi) = \{0.015, 0.98\}$. The approximated system has poles in $\text{eig}\left(\text{round}\left(\Phi \cdot 2^4\right) \cdot 2^{-4}\right) = \{-0.03, 1.03\}$. Hence, $n = 4$ makes the approximation unstable even though the original system was stable! $n = 6$ is the smallest number which keeps stability of the system. So either one have to increase the word length to, e.g., 16 or one must transform the system into a state-space realization that is less sensitive to quantization errors.

6.

a. Writing the I-part in the time-domain we obtain

$$I(t) = \frac{K}{T_i} \int_0^t e(\tau) d\tau$$

Taking the derivate of this leads to

$$\frac{dI}{dt} = \frac{K}{T_i} e(t)$$

and

$$\frac{I(kh) - I(kh-h)}{h} = \frac{K}{T_i} e(kh)$$

from which one obtains the answer

$$i(kh) = i(kh-h) + \frac{Kh}{T_i} e(kh)$$

b. The result is obtained by substituting

$$s' = \frac{2}{h} \frac{z-1}{z+1}$$

in

$$D(s') = -\frac{s' K T_d Y(s)}{1 + s' T_d / N}$$

Doing this one obtains

$$d(kh) = \frac{2T_d - Nh}{2T_d + Nh} d(kh-h) - \frac{2KNT_d}{2T_d + Nh} (y(kh) - y(kh-h))$$

c.
```
y = yIn.get();
e = yref - y;
i = i + (K*h/Ti)*e;
d = d - bd * (y - yold);
u = K*Beta*yref - K*y + i + d;
uOut.put(u);
d = ad*d;
yold = y;
```

The precalculated constants `ad` and `bd` are updated when the controller parameters are updated as follows:

```
ad = (2*Td - N*h)/(2*Td + N*h);
bd = 2*K*N*Td / (2*Td + N*h);
```

The engineer has made the following misstakes:

1. The transition where the parallel tracks join will never be true. The condition should always be true.
2. The production of fire units is started but never stopped. There are different ways of stopping the production. You could for instance change the line
   `S ProduceFire = 1;` to `N ProduceFire;`. Another possibility is to add an exit action in the same block that starts the production i.e., add the line
   `X ProduceFire = 0;`. A third option is to end the production when 50 units are produced by adding `S ProduceFire = 0;` to the step after the transition `FireUnitsDone == 50`.

3. The conditions for when to produce another batch and when to go back to the start should be switched. According to the conditions in the original diagram the system goes back to the initial step if `Batches` is less than ten.
4. The integer `Batches` needs to be reset in the inital step i.e, the line `S Batches = 0;` needs to be added to the inital step.

8. The code does not support bumpless parameter changes when `Beta` is not equal to 1. When `Beta`$\neq 1$ then the control signal in stationarity, i.e., when the error is zero, is given by

$$u = K(\beta y_{sp} - y) + I.$$

When either `K` or `Beta` is changed the control signal before and after the change should be the same in order to avoid a bump in the control signal, i.e.,

$$I_{new} + K_{new}(\beta_{new} y_{sp} - y) = I_{old} + K_{old}(\beta_{old} y_{sp} - y)$$

This is the case if the I-part is updated as follows when `K` or `Beta` are changed:

$$I_{new} = I_{old} + K_{old}(\beta_{old} y_{sp} - y) - K_{new}(\beta_{new} y_{sp} - y)$$

This is obtained if the code is modified as follows. We update the integrator state in `updateState` whenever `Beta` is different from 1.

```
public class PI {
  private PIParameters p;
  private double I; // Integrator state
  private double v; // Desired control signal
  private double e; // Current control error
  private double yref; // Setpoint
  private double y; // Measurement value

  //Constructor
  public PI(String name) {
    PIParameters p = new PIParameters();
    p.Beta = 0.8;
    p.H = 0.1;
    p.integratorOn = false;
    p.K = 1.0;
    p.Ti = 0.0;
    p.Tr = 10.0;
    new PIGUI(this, p, name);
    setParameters(p);

    this.I = 0.0;
    this.v = 0.0;
    this.e = 0.0;
  }

  public synchronized double calculateOutput(double y, double yref) {
```

```
    this.yref = yref;
    this.y = y;
    this.e = yref - y;
    this.v = p.K * (p.Beta * yref - y) + I;
    return this.v;
  }

  public synchronized void updateState(double u) {
    if (p.integratorOn) {
      I = I + (p.K * p.H / p.Ti) * e + (p.H / p.Tr) * (u - v);
    } else {
      I = 0.0;
    }
  }

  public synchronized long getHMillis() {
    return (long)(p.H * 1000.0);
  }

  public synchronized void setParameters(PIParameters newP) {
    if ((p.Beta != 1) || (newP.Beta != 1)) {
      I = I + p.K*(p.Beta*yref-y) - newP.K*(newP.Beta*yref-y);
    }
    p = (PIParameters)newP.clone();
    if (!p.integratorOn) {
      I = 0.0;
    }
  }
}
```

9.

a. The implementation set `weShouldRun` to `false` and then the idea is that the Regul thread should set the control signal to 0 before it terminates. However, it is very possible that the Regul thread is terminated before it gets to do this.

A proper solution needs to ensure that

- The output must be set to 0 before the call to `System.exit(0)` is performed.
- It is should not be possible for a thread to set the control signal to something different once it has been set to 0

b. The following implementation of the Regul thread solves the problem. It is, however, not the only solution.

```
public class Regul extends Thread {

  private PI inner = new PI("PI");

  private AnalogIn analogInAngle;
  private AnalogOut analogOut;
```

```
private ReferenceGenerator referenceGenerator;
private OpCom opcom;

private int priority;
private volatile boolean weShouldRun = true;
private long starttime;

private Semaphore mutex; // Used for synchronization at shutdown


public Regul(int pri) {
  priority = pri;
  mutex = new Semaphore(1);
  try {
    analogInAngle = new AnalogIn(0);
    analogOut = new AnalogOut(0);
  } catch (IOChannelException e) {
    System.out.print("Error: IOChannelException: ");
    System.out.println(e.getMessage());
  }
}


// Called in every sample in order to send plot data to OpCom
private void sendDataToOpCom(double yref, double y, double u) {
// Omitted
}

// Called from OpCom when shutting down
public synchronized void shutDown() {
  weShouldRun = false;
  mutex.take();
  setOut(0);
}


public void run() {
  long duration;
  long t = System.currentTimeMillis();
  starttime = t;

  setPriority(priority);
  mutex.take();
  while (weShouldRun) {
    double u = 0.0;
    double y = 0.0;
    double yRef = 0.0;

    yRef = referenceGenerator.getRef();
    y = getIn(analogInAngle);
    synchronized (inner) {
```

```
        u = limit(inner.calculateOutput(y, yRef));
        setOut(u);
        inner.updateState(u);
      }
      sendDataToOpCom(yRef, y, u);

      // sleep
      t = t + inner.getHMillis();
      duration = t - System.currentTimeMillis();
      if (duration > 0) {
        try {
          sleep(duration);
        } catch (InterruptedException x) {
        }
      }
    }
    mutex.give();
  }

  private double getIn(AnalogIn ain) {
  // Omitted
  }

  private void setOut(double u) {
  // Omitted
  }
}
```

Using this solution Regul is forced to terminate in an orderly fashion and Opcom sets the control signal to 0 before the real shutdown takes place.