LUNDS
UNIVERSITET

Institutionen för
**REGLERTEKNIK**

# Real Time Systems, FRTN01

**Exam 24 April 2020 kl 08–13**

### Points and Grades

**All answers must include a clear motivation, including the derivation of how the solution has been obtained, and a well-formulated answer.** Answers may be given in **English** or **Swedish**. The total number of points is 25. The maximum number of points is specified for each subproblem.

The number of points needed to pass the exam will not exceed 12. The number of points needed to get grade 4 will not exceed 17 and the number of points needed to get grade 5 will not exceed 22.

### Accepted Aids

On this distance exam all available material is allowed, such as books, old exams, exercise manuals, Google, using Matlab, etc. You may of course not communicate with or ask someone else for help. You will have contact with the teacher using zoom during the exam.

### Results

The result of the exam will become accessible through LADOK. The solutions will be available on the course home page.

**1.** A system is given by the following equation

$$by(k-1) + 0.5y(k) + y(k+1) = 2u(k-1) - u(k)$$

where $b$ is an unkown constant.

**a.** Determine the pulse transfer function of the system. (1 p)

**b.** Is the system stable? **Hint:** The asymptotic stability of a second order system with characteristic polynomial form $F(z) = z^2 + a_1 z + a_2$ are given by $(a_2 < 1)$, $(a_2 > -1 + a_1)$, $(a_2 > -1 - a_1)$ (1 p)

*Solution*

**a.** Advancing the time by 1

$$\begin{aligned} by(k) + 0.5y(k+1) + y(k+2) &= 2u(k) - u(k+1) \\ (z^2 + 0.5z + b)Y(z) &= (2 - z)U(z) \end{aligned}$$

Gives the pulse transfer function

$$H(z) = \frac{(2-z)}{z^2 + 0.5z + b}$$

**b.** Using the 3 given conditions

$$\begin{aligned} b &< 1 \\ b &> -1 + 0.5 \\ b &> -1 - 0.5 \end{aligned}$$

For asymptotic stability we get the condition $(-0.5 < b < 1)$

**2.** Consider the continuous system

$$\begin{aligned} \dot{x}(t) &= \begin{bmatrix} a & 0 \\ b & c \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ d \end{bmatrix} u(t) \\ y(t) &= \begin{bmatrix} 0 & 1 \end{bmatrix} x(t) \end{aligned}$$

**a.** Sample the system using zero-order-hold with a sample time of $h$. (2 p)

**b.** Instead consider the discrete-time system below where $h$ is the sampling period.

$$\begin{aligned} x(k+1) &= \begin{bmatrix} 1 & 0 \\ h & 1 \end{bmatrix} x(k) + \begin{bmatrix} 2h \\ h^2 \end{bmatrix} u(k) \\ y(k) &= \begin{bmatrix} 0 & 1 \end{bmatrix} x(k) \end{aligned}$$

Design a state-feedback controller such that the states go to 0 in a minimal amount of steps. (1.5 p)

**c.** In a general system, the control signal is saturated. Given the initial state $x(0) = [1, 1]^T$ what is the smallest sampling interval that we may use such that the control is limited to $|u(k)| \leq 2$ (assume $h > 0$). You may assume that the control signal takes it largest value at $k = 0$. (1 p)

*Solution*

**a.** Deriving $\Phi$ using the inverse laplace transform method gives us

$$(sI - A)^{-1} = \frac{1}{(s-a)(s-c)} \begin{bmatrix} s-c & 0 \\ b & s-a \end{bmatrix} = \begin{bmatrix} \frac{1}{s-a} & 0 \\ \frac{b}{(s-a)(s-c)} & \frac{1}{s-c} \end{bmatrix}$$
$$\rightarrow \mathcal{L}^{-1} \rightarrow \begin{bmatrix} e^{at} & 0 \\ b\frac{e^{ct}-e^{at}}{c-a} & e^{ct} \end{bmatrix}$$

And since $\Phi = \mathcal{L}^{-1}(sI - A)^{-1}$ we have our $\Phi$ matrix once we insert $t = h$. We acquire $\Gamma$ by

$$\Gamma = \int_0^h e^{As} ds B = \int_0^h \begin{bmatrix} 0 \\ de^{cs} \end{bmatrix} ds = \begin{bmatrix} 0 \\ \frac{d}{c}\left(e^{ch} - 1\right) \end{bmatrix}$$

giving us our discrete time system

$$x(k+1) = \begin{bmatrix} e^{ah} & 0 \\ b\frac{e^{ch}-e^{ah}}{c-a} & e^{ch} \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ \frac{d}{c}\left(e^{ch} - 1\right) \end{bmatrix} u(k)$$
$$y(k) = \begin{bmatrix} 0 & 1 \end{bmatrix} x(k)$$

**b.**

$$u(k) = -Lx(k)$$

We want to match $\det(zI - (\Phi - \Gamma L))$ (where $L = [l_1, l_2]$) to the polynomial $z^2$ (deadbeat controller).

$$\Gamma L = \begin{bmatrix} 2hl_1 & 2hl_2 \\ h^2 l_1 & h^2 l_2 \end{bmatrix}$$
$$\det(zI - \Phi + \Gamma L) = \begin{vmatrix} z - 1 + 2hl_1 & 2hl_2 \\ -h + h^2 l_1 & z - 1 + h^2 l_2 \end{vmatrix}$$
$$= z^2 + (h^2 l_2 - 2 + 2hl_1)z + (h^2 l_2)$$
$$\Longrightarrow \begin{cases} h^2 l_2 - 2 + 2hl_1 & = 0 \\ 1 + h^2 l_2 - 2hl_1 & = 0 \end{cases}$$
$$\Longrightarrow \begin{cases} l_1 & = 3/4h \\ l_2 & = 1/2h^2 \end{cases}$$

**c.** The control signal at $k = 0$ is given by $u(0) = -l_1 x_1(0) - l_2 x_2(0) = -(l_1 + l_2)$ The solution is given by the postive solution to the equation $8h^2 - 3h - 2 = 0$. This second order equation can be solved in different ways, either by hand or using Matlab. The positive solution is $h = 0.7215$, i.e., the answer is that $h \geq 0.7215$.

**3.** A newly graduated engineer has recently taken a course in real-time systems and decided to incorporate his newly acquired knowledge into his control code to make it thread-safe. However, due to an unexpected virus outbreak, the engineer missed a few of the lectures and missed some essential concepts. The code is supposed to execute a P-controller (one controller executing for two different processes), actuate the control signal and print the given control signals to the console without having any contention for the screen. Note that the controller will use the same `K` for both processes. However, this is not a problem in this application (but a feature of the architecture).

The code can be seen below.

```
public class Controller {

    private double K = 1.0;
    private String id = "P-Controller";

    public synchronized double calculate(double r, double y) {
        return K*(r-y);
    }

    public synchronized void setK(double K) {
        this.K = K;
    }

    public synchronized String toString() {
        return id + " with K = " + K;
    }
}
public class Regulator extends Thread {

    private Controller ctrl;
    private int period;
    private AnalogIn in;
    private AnalogOut out;

    public Regulator(Controller ctrl, int period, int processID) {
        this.ctrl = ctrl;
        this.period = period;
        in = new AnalogIn(processID);
        out = new AnalogOut(processID);
    }

    public void run() {
        double u, y;
        double r = 0.0;
        long t = System.currentTimeMillis();
        while (!interrupted()) {
            y = in.get();
            synchronized (this) {
                u = ctrl.calculate(r, y);
                out.set(u);
                System.out.println(ctrl.toString() + ": " + u);
            }
            try {
                t += period;
                long d = t - System.currentTimeMillis();
                if (d > 0) {
                    Thread.sleep(d);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
```

```
    public static void main(String[] args) {
        Controller ctrl = new Controller();
        Regulator r1 = new Regulator(ctrl, 10, 1);
        Regulator r2 = new Regulator(ctrl, 10, 2);
        r1.start();
        r2.start();
    }
}
```

**a.** Can you find the real-time problems with the code? (2 p)

**b.** Fix the problems. (2 p)

*Solution*

**a.**
1. We synchronize on `this` instead of `ctrl`. This does not make us thread-safe since multiple threads can access the mutually exclusive section simultaneously.
2. We print things to the screen without treating the screen as a shared resource. This will imply that text might be mixed on screen and weird behavior will occur. IO-devices (such as the screen) is considered shared variables. Therefore, the screen should be put into a monitor or similar.

**b.**
1. Quick fix is to synchronize on ctrl instead of this. This gives 1 p since the explanation of why we need to synchronize on `ctrl` is important in a. If synchronization has been performed using `this.getClass()` instead 0.5 points have been awarded. This is due to the fact that it is inefficient. For instance, if someone were to add a synchronized method to the Regulator class, the synchronized method in the run loop would have to be changed as well (in order to prevent unnecessary blocking).
2. Create a monitor, semaphore or similar, for the screen, or introduce a semaphore that the regulators share.

Also seen in the code below.

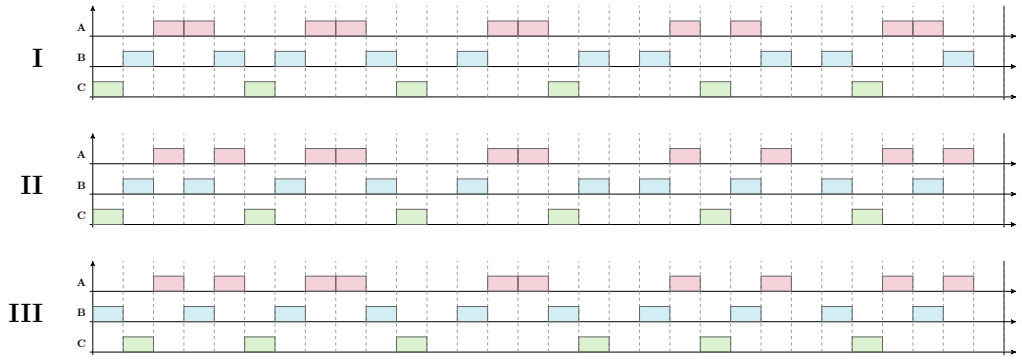
```
public class ScreenMonitor {

    public synchronized void print(String s) {
        System.out.println(s);
    }
}
public class Controller {

    private double K = 1.0;
    private String id = "P-Controller";

    public synchronized double calculate(double r, double y) {
        return K*(r-y);
    }

    public synchronized void setK(double K) {
        this.K = K;
    }

    public synchronized String toString() {
        return id + " with K = " + K;
    }
}
public class Regulator extends Thread {
```

```
    private Controller ctrl;
    private ScreenMonitor screen;
    private int period;
    private AnalogIn in;
    private AnalogOut out;

    public Regulator(Controller ctrl, ScreenMonitor screen,
            int period, int processID) {
        this.ctrl = ctrl;
        this.screen = screen;
        this.period = period;
        in = new AnalogIn(processID);
        out = new AnalogOut(processID);
    }

    public void run() {
        double u, y;
        double r = 0.0;
        long t = System.currentTimeMillis();
        while (!interrupted()) {
            y = in.get();
            synchronized (ctrl) {
                u = ctrl.calculate(r, y);
                out.set(u);
                screen.print(ctrl.toString() + " from process "
                        + processID + ": " + u);
            }
            try {
                t += period;
                long d = t - System.currentTimeMillis();
                if (d > 0) {
                    Thread.sleep(d);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    public static void main(String[] args) {
        Controller ctrl = new Controller();
        ScreenMonitor screen = new ScreenMonitor();
        Regulator r1 = new Regulator(ctrl, screen, 10, 1);
        Regulator r2 = new Regulator(ctrl, screen, 10, 2);
        r1.start();
        r2.start();
    }
}
```

**4.** Three schedules (**I-III**) of the same schedulable task set can be seen in Figure 1 for one hyper-period. All tasks are released simultaneously at the critical instant $t = 0$. The schedules correspond to a rate-monotonic, a deadline-monotonic, and an EDF scheduling strategy.

The schedules in Figure 1 corresponds to the following task set where some values have gotten lost.

| Task name | C | D | T |
|---|---|---|---|
| A | ? | 5 | 6 |
| B | ? | 3 | ? |
| C | 1 | ? | 5 |




**Figur 1** The three schedules necessary to solve Problem 4

For each $i$ it holds that $D_i$ and $T_i$ are unique, i.e., $D_A \neq D_B \neq D_C$ and $T_A \neq T_B \neq T_C$.

**a.** What are the unknown task set parameters? Motivate. (2 p)

**b.** Pair the execution traces I-III with their corresponding scheduling algorithm (rate-monotonic, deadline-monotonic, and EDF). For full points, a thorough motivation is need. (2 p)

*Solution*

**Tabell 1** Taskset for Problem 4

| Task name | C | D | T |
|---|---|---|---|
| A | 2 | 5 | 6 |
| B | 1 | 3 | 3 |
| C | 1 | 2 | 5 |

**a.** The correct task set can be seen in Table 1.

- $D_C = 2$: This is easy to see that it has to be 2. Given every single schedule, it executes within two time units of its release. It also starts execution in two of the traces (the one for EDF and DM), meaning that $D_C < D_B$.
- $C_A = 2$: Given the parameters for task $A$ we can easily see that the only way to satisfy any of the schedules $C_A$ has to be 2.
- $T_B = 3$: Given from schedule $I$. Due to us having a hyperperiod of 30, $T_B \in \{2, 3, 5, 6, 10, 15\}$ we can quickly prune 2. The second thing we can observe is that if $D_B = 3$ and we assume $T_B \geq 5$, we will miss deadlines in schedule $I$. Therefore, the only combination of $T_B$ and $C_B$ that could work is $T_B = 3$ and $C_B = 1$.

  Another argument could be that since $C$ has the highest priority in two of the traces and $B$ has highest priority in one trace could mean that $C$ has an earlier deadline but $B$ has a shorter period meaning that $B$ will preempt $C$ in the rate-monotonic priority assignment. This argument together with the hyperperiod argument, once again gives us $T_B = 3$.

- $C_B = 1$: See reasoning for $T_B$.

**b.** Given the task set deduced in subproblem a, we can easily see that schedule $I$ is RM. schedule $II$ and $III$ can easily be assigned by either drawing the beginning of the schedules or reasoning about the preemption in the early execution times.

- $I$ = EDF
- $II$ = DM
- $III$ = RM

**5.** You friend has discretized a standard PI controller

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

and implemented it with the following pseudo-code:

```
...
while(1){
    t=getCurrentTime();
    y=getMeasurement();
    r=getReference();
    e=r-y;
    P=K*e;
    I=I+(K*h/Ti)*e;
    u=P+I;
    applyControl(u);
    sleepUntil(t+h);
}
...
```

**a.** Which method had been used to discretize the continuous-time integral term? (1 p)

**b.** He then noticed that in some experiments the control signal saturated and the controller performed very poorly with large overshoots and undershoots. What is the likely reason for this? (0.5 p)

**c.** In order to fix this he implemented a periodic reset according to below:

```
...
int i=0;
while(1){
    i++;
    t=getCurrentTime();
    y=getMeasurement();
    r=getReference();
    e=r-y;
    P=K*e;
    I=I+(K*h/Ti)*e;
    if (i==60){
        i=0;
        I=0;
    }
    u=P+I;
    applyControl(u);
    sleepUntil(t+h);
}
...
```

Do you think this controller will work as intended? Explain why or why not. (1 p)

**d.** Correct the code to solve the encountered problem. (1.5 p)

*Solution*

**a.**

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

$$\frac{dI}{dt} = \frac{K}{T_i}e$$

If one uses the common forward difference methods to discretize the I-part then one obtains

$$\frac{I(t_{k+1}) - I(t_k)}{h} = \frac{K}{T_i}e(t_k)$$

```
I(k+1) := I(k) + (K*h/Ti)*e(k)
```

Using this approach the integral part is updated in the UpdateStates part of the code which is not the case here.

If one instead uses the backward difference method one instead obtains

$$\frac{I(t_k) - I(t_{k-1})}{h} = \frac{K}{T_i}e(t_k)$$

```
I(k) := I(k-1) + (K*h/Ti)*e(k)
```

which is what we have in the code above, i.e., the method that has been used is the backward (Euler) approximation method.

**b.** A saturated control signal and bad control performance points at an integrator windup problem. Since the controller code does not contain any anti-windup protection this is probably the cause of the problem.

**c.** The new controller will not solve the problem. The reset of the integrator state is likely to give large steps to the control action and actually decrease the control performance. Assume, e.g., that the control loop is in stationarity with zero error. In this case the entire control signal comes from the integral term. To reset this would cause a very large bump.

**d.** Any form of any-wind-up will solve the problem. The tracking solution is shown below

```
...
while(1){
    t=getCurrentTime();
    y=getMeasurement();
    r=getReference();
    e=r-y;
    I=I+(K*h/Ti)*e + (h/Tr)*(u - v);
```

```
8       v=K*e+I;
9       u = sat(v,umin,umax);
10      applyControl(u);
11      sleepUntil(t+h);
12  }
13  ...
```

**6.** In a shopping mall with two halls there are for safety regulations not supposed to be more than a given number of people in each of the halls. Customers can enter into one of the halls and leave from the other. There are security guards at the entrance and exit that count the people entering or leaving. The problem is knowing exactly how many people are in each of the halls. There exist cameras that allow can count in real time how many people there are in a room but unfortunately these are very expensive. The administrator of the shopping mall believes that he needs one camera per hall. But you disagree with this. You want to show him a way to avoid spending money on cameras.

You therefore use what could be a mathematical model of how people move in the shopping mall. You want to show him how you can use it to estimate the number of people in the halls without the use of cameras, but only using the guards that count the people entering and exiting.

The following model can be used:

$$\begin{aligned} x_1(k+1) &= 0.5x_1(k) + 0.25x_2(k) + u(k) \\ x_2(k+1) &= 0.5x_1(k) + 0.5x_2(k) \\ y(k) &= 0.25x_2(k), \end{aligned}$$

where $x_1(k)$ and $x_2(k)$ represent the number of people in each of the two halls. The second equation can be interpreted in the following way: half of the people in $x_1$ will move to $x_2$ and half of the people in $x_2$ will stay there.

**a.** What do $u(k)$ and $y(k)$ represent in this model? (1 p)

**b.** Design an observer on predictor form to compute the number of people in the different rooms. Place the observer poles in 0 and 0.5. Write the observer equations. (2 p)

**c.** Write an algorithm for implementing the observer. Just write the pseudocode: define the needed variables and constants, and the operations to implement the observer.You can use the functions `getY()`, `getU()` to obtain $y$ and $u$. You may not assume that you have access to matrix algebra. (2 p)

*Solution*

**a.** $u(k)$ represents the number of people entering the mall (they can only enter directly the first hall) that we can measure and eventually control thanks to the bumpers. $y(k)$ represents the number of people leaving the shopping mall that are counted by the security guards.

**b.** The system dynamics and output matrices are:

$$\Phi = \begin{bmatrix} 0.5 & 0.25 \\ 0.5 & 0.5 \end{bmatrix}, C = \begin{bmatrix} 0 & 0.25 \end{bmatrix} \quad (1)$$

They dynamics of the error are driven by the equation: $\tilde{x}(k+1) = (\Phi - KC)\tilde{x}(k)$. The characteristic polynomial of the error dynamics matrix is

$$det(zI - (\Phi - KC)) = det(\begin{bmatrix} z - 0.5 & -0.25 - 0.25k_1 \\ -0.5 & z - 0.5 + 0.5k_2 \end{bmatrix}) =$$

$$= (z - 0.5)(z - 0.5 + 0.5k_2) - (-0.125 - 0.125k_1) =$$

$$z^2 + (-1 + 0.5k_2)z + 0.125 - 0.25k_2 + 0.25k_2$$

If we decide to place the observer poles in 0 and 0.5 the desired characteristic polynomial is: $p_{obs}(z) = z^2 - 0.5z$ hence we obtain the set of equations:

$$\begin{aligned} -0.5 &= -1 + 0.5k_2 \\ 0 &= 0.125 - 0.25k_2 + 0.25k_1 \end{aligned}$$

from which we obtain $K = [0.5, 1]^T$

The equations for the observer are

$$\hat{x_1}(k+1) = 0.5\hat{x_1}(k) + 0.25\hat{x_2}(k) + u(k) + k_1(y(k) - 0.25\hat{x_2}(k))$$
$$\hat{x_2}(k+1) = 0.5\hat{x_1}(k) + 0.5\hat{x_2}(k) + k_2(y(k) - 0.25\hat{x_2}(k))$$

**c.** The observer shall contain the reading of the signals, the storage of the previous estimated values and the implementation of the equations. Likely mistake is that the old variables are not stored and therefore the $x_1$ is already update in the second equation.

```
define k1 0.5;
define k2 1;
double x1_hat, x2_hat;
double x1_hat_new, x2_hat_new;
double u, y;

u = getU();
y = getY();
x1_hat_new = 0.5*x1_hat + 0.25*x2_hat + u + k1*(y-0.25*x2_hat);
x2_hat_new = 0.5*x1_hat + 0.5*x2_hat + k2*(y-0.25*x2_hat);
x1_hat = x1_hat_new;
x2_hat = x2_hat_new; // x_2_hat_new could be skipped
```

**7.** Assume that you have a signed 16 bit word in the memory of a controller that represents a real value using fixed point arithmetic. The sequence of bits is the following:

```
0100 0011 0001 0111
```

Unfortunately you have forgotten how many fractional bits the fixed point number had but you remember that the real number is greater than 9 and smaller than 100. What are the possible number of fractional bits and the corresponding real values based on this information? (1.5 p)

*Solution*

The second most significant bit is nonzero. Since we know that the number is not greater than 100 that bit cannot "weight" more than $2^6$ (in fact $2^7$ is already 128). This means that we cannot have more than 8 integer bits (including the leftmost bit). We also know that the number is greater than 9. This means that the most significant bit cannot "weight" less than 2^4 (in fact $2^3 = 8$ and the next three bits are zero). This leaves the three options: there are 8, 9, or 10 fractional bits.

The integer number corresponding to `0100 0011 0001 0111` is 17175, i.e. hence the variable can have one of the following values:

- 8 fractional bits: $17175 * 2^{-8} = 67.0898$
- 9 fractional bits: $17175 * 2^{-9} = 33.5449$
- 10 fractional bits: $17175 * 2^{-10} = 16.7725$

Alternative solution 1:

With 1 fractional bit the integer part is `100 0011 0001 011`, i.e., larger than $2^{13} = 8192$ and hence too large.
With 2 fractional bits the integer part is `100 0011 0001 01`, i.e., larger than $2^{12} = 4096$ and hence too large.
With 3 fractional bits the integer part is `100 0011 0001 0`, i.e., larger than $2^{11} = 2048$ and hence too large.
With 4 fractional bits the integer part is `100 0011 0001`, i.e., larger than $2^{10} = 1024$ and hence too large.
With 5 fractional bits the integer part is `100 0011 000`, i.e., larger than $2^9 = 512$ and hence too large.
With 6 fractional bits the integer part is `100 0011 00`, i.e., larger than $2^8 = 256$ and hence too large.
With 7 fractional bits the integer part is `100 0011 0`, i.e., larger than $2^7 = 128$ and hence too large.
With 8 fractional bits the integer part is `100 0011`, i.e., larger than $2^6 = 64$ and hence possible.
With 9 fractional bits the integer part is `100 001`, i.e., larger than $2^5 = 32$ and hence possible.
With 10 fractional bits the integer part is `100 00`, i.e., equal to $2^4 = 16$ and hence possible.
With 11 fractional bits the integer part is `100 0`, i.e., equal to $2^3 = 8$ which is too small. The same holds for larger number of fractional bits. Hence, the only possibilities are 8, 9, or 10 fractional bits and then the variable will have one of the following values:

- 8 fractional bits: $17175 * 2^{-8} = 67.0898$
- 9 fractional bits: $17175 * 2^{-9} = 33.5449$
- 10 fractional bits: $17175 * 2^{-10} = 16.7725$

Alternative solution 2:

Exhaustive evaluation. Evaluate $17175 * 2^{-n}$ for all $n$ in the range 1 to 14. Easily done with a for loop in Matlab.