LUND INSTITUTE
OF TECHNOLOGY
Lund University

Department of
AUTOMATIC CONTROL

Real-Time Systems

Exam January 8, 2013, hours: 14.00–19.00

Points and grades

All answers must include a clear motivation 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.

Accepted aid

The textbooks Real-Time Control Systems and Computer Control: An Overview - Educational Version. Standard mathematical tables and authorized "Real-Time Systems Formula Sheet". Pocket calculator.

Results

The result of the exam will be posted on the notice-board at the Department. The result as well as solutions will be available on WWW:
http://www.control.lth.se/course/FRTN01/

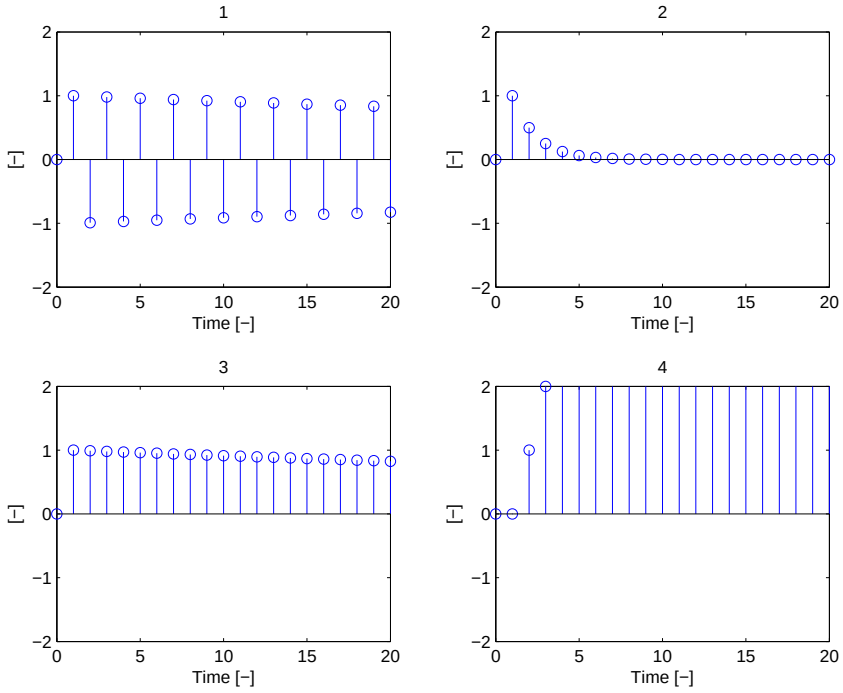

Figure 1 Pulse response for Problem 1.

1.

a. Four pulse responses are shown in Figure 1. Match the pulse responses and the following four transfer functions:

$$G_1(z) = \frac{1}{(z-1)^2}$$
$$G_2(z) = \frac{1}{z+0.99}$$
$$G_3(z) = \frac{1}{z-0.99}$$
$$G_4(z) = \frac{1}{z-0.5}$$

(2 p)

Do not forget to motivate!

b. Which of $G_2(z)$, $G_3(z)$, and $G_4(z)$ can not be the result of zero-order hold sampling a continuous-time system? Motivate! (1 p)

2. Consider the system

$$x(k+1) = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} u(k)$$
$$y(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(k)$$

a. What is the pulse-transfer function of the system? (1 p)

b. Is the system stable? (0.5 p)

c. Design a deadbeat state feedback controller on the form $u(k) = -Lx(k) + l_r r(k)$ with unit static gain from r to y. (1.5 p)

3. What does the acronym PWM mean? Explain what it is and how it works. (1 p)

4. Consider the computer-controlled system in Fig. 2. The P-controller should

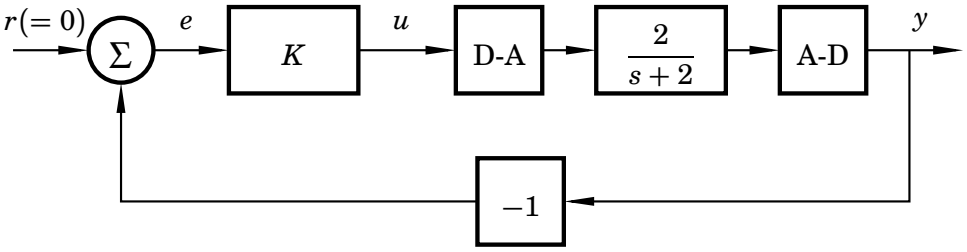

Figure 2 Computer-controlled system.

execute with the sampling interval $h = 0.5$ s, and the controller gain is given by $K = 2$.

a. Consider the following implementation of the controller:

```
LOOP
  y = readInput();
  u = -K*y;
  writeOutput(u);
  waitForNextPeriod();
END;
```

Assume that the execution time of the controller is negligible. Show that the closed-loop system is stable. (1.5 p)

b. Now consider the following implementation of the controller:

```
LOOP
  writeOutput(u);
  y = readInput();
  u = -K*y;
  waitForNextPeriod();
END;
```

Will the closed-loop system still be stable? (1.5 p)

5. Assume that you should implement the following PD control algorithm using a small 8-bit processor:

$$x(k+1) = 0.1466x(k) + 1.050y(k)$$
$$u(k) = 1.517x(k) - 2.278y(k)$$

All variables (y, u, x) and all coefficients should be represented using 8-bit signed integers (`int8_t`). Intermediate results may however use 16 bits (`int16_t`).

a. Select a suitable number of fractional bits and convert the coefficients to fixed-point numbers. (Use the same number of fractional bits for all coefficients.) (1 p)

b. The pole of the original controller is located in 0.1466. Where is the pole of the fixed-point representation of the controller located? (0.5 p)

c. Write C-code, showing how the fixed-point calculations in the control algorithm are implemented. Declare the word-length (size) of each used variable. Write the code so that x and u are guaranteed not to overflow. In order to get full points the code should be written so that the input-output latency is as small as possible.

The measured signal, `y`, is obtained through

```
y = readInput();
```

You may assume that `readInput` returns a value on the same Q-format as you are using for your controller parameters.

You send out the control signal through

```
writeOutput(u);
```

Also here you may assume that the argument has the same Q-format as what you are using for your controller parameters. (2.5 p)

6. In the course we have used the following version of a PID controller.

$$U(s) = K(\beta Y_{ref}(s) - Y(s) + \frac{1}{T_I s}(Y_{ref}(s) - Y(s)) - \frac{sT_D}{1 + sT_D/N}Y(s))$$

The derivative term can be interpreted as an ideal derivative that is filtered using a first-order low-pass filter with the time constant T_D/N. The objective for the low-pass filter is to reduce the effects of high-frequency measurement noise. However, high-frequency noise will still generate significant variations in the control signal.

An alternative to the above version is to use a second-order low-pass filter and instead of filtering only the derivative, filter the measurement signal, y, and then use the filtered measurement signal y_f as an input to a PID controller with an ideal derivative, according to the following transfer functions:

$$Y_f(s) = \frac{1}{T_f^2 s^2 + 1.4T_f s + 1}Y(s) \quad (1)$$

$$U(s) = K(\beta Y_{ref}(s) - Y_f(s) + \frac{1}{T_I s}(Y_{ref}(s) - Y_f(s)) - T_D s Y_f(s))$$

The relative damping of the filter is $\zeta = 1/\sqrt{2}$. The filter constant T_f is typically chosen as T_I/N for PI control or as T_D/N for PID control, where N ranges from 2 to 20. The gain of this controller will go to zero for high frequencies.

a. A good state-space representation of the filter is obtained by chosing the states $x_1(t) = y_f(t)$ and $x_2(t) = dy_f(t)/dt$. The resulting state space representation then becomes

$$\frac{dx_1(t)}{dt} = x_2(t)$$

$$\frac{dx_2(t)}{dt} = -\frac{1.4}{T_f}x_2(t) - \frac{1}{T_f^2}x_1(t) + \frac{1}{T_f^2}y(t)$$

Discretize this state-space system using backward difference approximation. Write your solution on the form $x[k] = f(x[k-1], y[k])$. (2 p)

b. Discretize the PID controller. Use the discretizaton of the filter from the previous subproblem. Use a forward difference approximation for the integral part. Write the result as pseudo-code where the separation between the CalculateOutput and the UpdateState parts is clearly seen. Describe all variables that you define. Ignore anti-windup and auto/manual mode handling. (2 p)

7. The table below describes three processes with their respective periods, deadlines, worst-case execution times and best-case execuion times. The tasks are executed using fixed-priority scheduling with rate-monotonic priority assignment.

Task	T_i	D_i	C_i^{max}	C_i^{min}
A	3	3	1	0.5
B	16	16	7	4
C	50	50	2	1.5

a. Will all deadlines be met? (1 p)

b. What are the worst-case response times of the tasks? (1 p)

c. What are the best-case response times of the tasks? (1 p)

8. The readers-writers problem where we require that writer processes have exclusive access to the critical section and that there may be at maximum three simultaneous reader processes can be modeled using the generalized Petri net shown in Figure 3.

a. Show how readers-writer synchronization can be implemented in Java using a multi-step quantity semaphore (a counting semaphore that permits increments and decrements that are greater than one).

The methods available for a multi-step semaphore are the following:

```
// Constructor that initializes the internal counter to 0
public MultiStepSemaphore();

// Constructor that initializes the internal counter to n
public MultiStepSemaphore(int n);

// Causes the calling thread to block until the counter assumes a value
// greater than or equal to n. Before returning the counter is decremented with n.
```

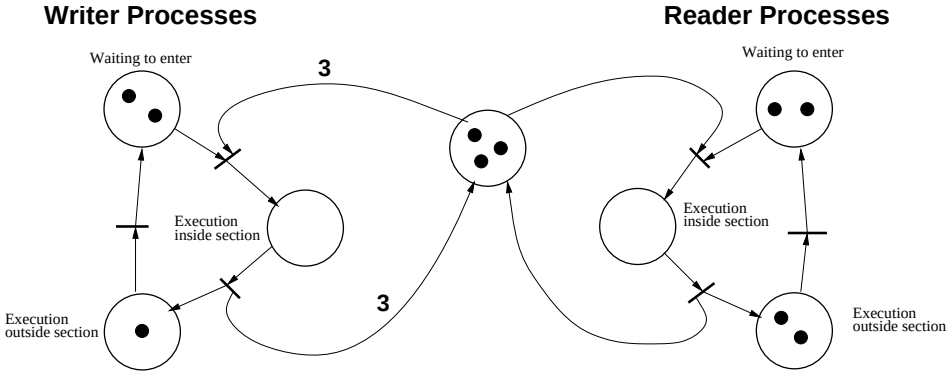

Figure 3 Readers-writers problem modeled with Petri net.

```
public void take(int n);

// Equal to take(1);
public void take();

// Increments the counter by n and releases that many blocked threads,
// if any.
public void give(int n);

// Equal to give(1);
public void give();
```

Use the following code structure as the starting point and fill in the missing parts in the `Writer`, `Reader`, and `Main` classes. It is only the syncronization-related code that you need to add.

```
public class Writer extends Thread {

  MultiStepSemaphore sem;

  public Writer(MultiStepSemaphore s) {
    sem = s;
  }

  public void run() {
    while (true) {

    // access critical section

    }
  }
}

public class Reader extends Thread {

  MultiStepSemaphore sem;

  public Reader(MultiStepSemaphore s) {
    sem = s;
  }
```

```
  public void run() {
    while (true) {

    // access critical section

    }
  }
}

public class Main {

  public static void main(String[] args) {

  MultiStepSemaphore s;
  Writer w;
  Reader r;


  for (int = 1; i==4; i++) {
    w = new Writer(s);
    w.start();
  }
  for (int i = 1; i==4; i++) {
    r = new Reader(s);
    r.start();
  }
}
```

(1 p)

b. In Figure 4 the solution to an extended readers-writers problem is shown where we also allow two writer processes in the critical section at the same time. Still we may have at most three simultaneous reader processes inside the critical section. Also, we do not allow readers and writers to be in the section at the same time.

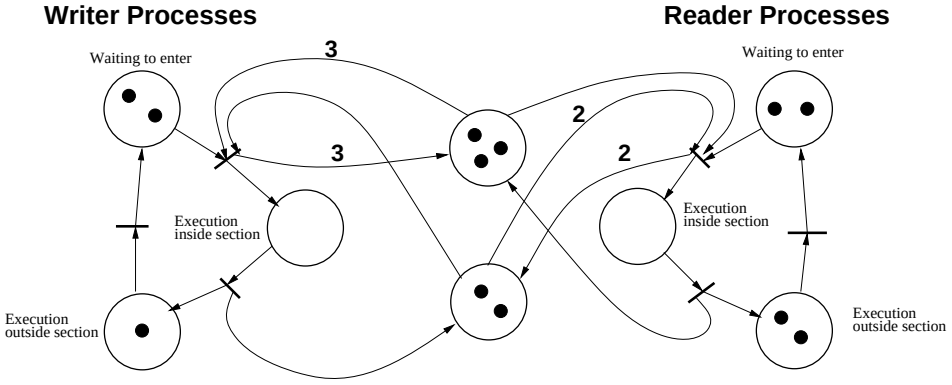

Figure 4 Extended readers-writers problem modeled with Petri net.

Implement the class `ReadersWritersGuard` with the following interface and show using the code skeleton in the first subproblem how the class should be used. You may exclude all exception handling code (try, catch).

```
// Constructor that takes the maximum number of simultaneous writers
// and readers as arguments
```

```
public ReadersWritersGuard(int maxWriters, int maxReaders);

// Equivalent to public ReadersWritersGuard(1,1)
public ReadersWritersGuard();

// Called by a writer process before entering the critical section.
// Blocks if buffer is full of writers or contains any readers
public synchronized void writersTake();

// Called by a reader process before entering the critical section.
// Blocks if buffer is full of readers or contains any writers
public synchronized void readersTake();

// Called by a writer process before exiting the critical section.
// Releases waiting readers and/or writers.
public synchronized void writersGive();

// Called by a reader process before exiting the critical section.
// Releases waiting readers and/or writers.
public synchronized void readersGive();
```

Readers and writers may be considered as equally important when accessing the critical section, i.e., it is not so that writers should have priority over readers or vice versa when both writers and readers are waiting for access to the critical section.

(3 p)