LUND
UNIVERSITY

Department of
**AUTOMATIC CONTROL**

# Real-Time Systems

**Exam June 3rd, 2017, hours: 08:00–13:00**

### 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 become accessible through LADOK. The solutions will be available on WWW:
*http://www.control.lth.se/course/FRTN01/*

**1.** A system is composed of three concurrent tasks, that use 4 semaphores to synchronize their execution. The three tasks lock and unlock the semaphores as follows.

**Task A**
```
...
lock(s1);
lock(s3);
...
lock(s4);
unlock(s1);
...
unlock(s4);
unlock(s3);
...
```

**Task B**
```
...
lock(s1);
lock(s2);
lock(s4);
...
...
unlock(s4);
unlock(s2);
unlock(s1);
...
```

**Task C**
```
...
lock(s2);
...
lock(s3);
lock(s4);
...
unlock(s4);
unlock(s3);
unlock(s2);
...
```

Assuming that the priority ceiling protocol is used and that Task A has priority 10, Task B has priority 5 and Task C has priority 8 (the higher the number, the higher the priority of the process). Compute the ceiling for all the semaphores. Remember to justify your answer. (1 p)

**2.** Several of the controller and/or estimator structures presented in the course themselves contain an internal feedback loop. Describe two of these controller and/or estimator structures. In order to get full points, for each structure you should:

- describe what the process that is controlled corresponds to,
- describe what type of controller is used,
- write down the control law.

(2 p)

**3.** A new drone has to be released on the market, and the code has passed all the correctness tests, but it has not yet been decided how to configure the scheduler. The drone's software is composed of five different tasks and the real-time operating system of choice only supports Fixed Priority Scheduling. The tasks have the following characteristics, where numbers are given in clock time units (therefore only integer numbers are possible).

| $Task_i$ | $T_i$ | $D_i$ | $C_i$ |
|---|---|---|---|
| T1 | 10 | 5 | 1 |
| T2 | 50 | 50 | 10 |
| T3 | 30 | 20 | 2 |
| T4 | 20 | 20 | 2 |
| T5 | 4 | 4 | 1 |

**a.** You have to decide which priorities are to be assigned to the tasks. Should you use Deadline Monotonic or Rate Monotonic? Justify your answer and assign the priorities to the task according to the chosen algorithm as numbers from 1 to 5 (where 1 is the highest priority task). (1 p)

**b.** According to the approximate analysis (not the exact one), is the taskset schedulable? If yes, draw the schedule for the first hyperperiod (until the schedule then repeats itself). If no, assume that you can reduce the worst case execution time for task T2 and compute how much should the WCET of task T2 be to make the taskset schedulable according to the approximate analysis. (1 p)

**4.** Consider the harmonic oscillator given by

$$\dot{x}(t) = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} x(t) + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u(t)$$
$$y(t) = \begin{pmatrix} 1 & 0 \end{pmatrix} x(t)$$

**a.** Derive the continuous-time transfer function $G(s)$ and compute the poles. (0.5 p)

**b.** Derive the discrete-time pulse transfer function $H(z)$ by approximating the continuous-time transfer function $G(s)$ using **forward difference** (or Euler's method) with sampling period $h$.

Compute the poles and find for which sampling period $h$ the poles are located within (or on) the unit circle.

(2 p)

**c.** Derive the discrete-time pulse transfer function $H(z)$ by approximating the continuous-time transfer function $G(s)$ using **backward difference** with sampling period $h$.

For which sampling periods, $h$, are the poles located within (or on) the unit circle?

(1 p)

**5.** You are realizing a P controller with a fixed-point implementation. Your controller reads the error and should compute the control signal as

$$u(k) = k_p \cdot e(k)$$

where $k_p$ is a constant, equal to 4.5. You have done some analysis on your system and found out that the error will always belong to the interval $[-35.98, 20.02]$. Furthermore, you want to make sure that you have a resolution of at least 0.01.

**a.** What is the smallest number of bits that you need to represent the entire range with the given resolution constraint? Encode the constant $k_p$ with the given representation, providing both the decimal and the binary value. (2 p)

**b.** Given that you can choose the microprocessor that should execute the code, what choice would you do for the word length of the processor? (0.5 p)

**6.** An important tool within the field of molecular biology is the PCR (Polymerase Chain Reaction) method. Using this method it's possible to isolate and mass produce specific genes and DNA sequences. The number of copies of the desired DNA sequence is doubled in each cycle, and after 20 cycles about 1 million ($2^{20}$) copies are obtained.

A very simplified description of the method is the following:

1. Fill up a test tube with DNA samples, primers, DNA polymerase and nucleotides.
2. Heat until 95°C. (Separation of DNA strings.)
3. Cool until 37°C. (Primers bind to separated DNA strings.)
4. Heat until 72°C. (DNA polymerase continues to add nucleotide bases starting at the primers and new DNA copies are obtained.)
5. Repeat points 2–4 desired amount of cycles.

Construct a GRAFCET describing a PCR sequence with 20 cycles. The sequence should start when the signal **Start** becomes true. Additionally, you have access to the following variables:

**Temp** measurement signal for temperature (0–100°C)

**Heat** control signal for heating element (on/off)

**Cool** control signal for cooling element (on/off)

**Count** integer for keeping track of count

**Fill** signal to start filling the test tube with desired ingredients

**Ready** signal that indicates that the test tube is filled and ready to start

(3 p)

**7.** In continuous-time, purely oscillatory behavior (i.e., an oscillation with constant amplitude) occurs for systems with eigenvalues on the imaginary axis.

**a.** When does oscillatory behavior occur for discrete-time systems? (0.5 p)

**b.** Since the eigenvalues of real matrices come in complex-conjugated pairs one cannot have an purely oscillatory first-order system in continuous time. Is this possible for discrete-time systems? If yes, give an example. If no, explain why. (1 p)

**8.** Consider the following pseudo-code implementation of state feedback and observer controller. The observer is a predictor-form observer, i.e., with a one-sample delay. Assume that the implementation platform supports vector and matrix operations. The ReadInput and WriteOutput functions behave in the same was as the corresponding functions used in Laboratory 1.

```
while (true) {
 y = ReadInput();
 ref = getReference;
 u = lr*ref - L*x_hat;
 WriteOutput(u);
 x_hat = (Phi - Gamma*L - K*C)*x_hat + Gamma*lr*ref + K*y;
 //sleep
}
```

**a.** When the controller was tested it was found that it worked good in some conditions, but in other conditions it did not work at all. The controller was designed to be quite aggressive and the problem seemed to mainly occur in connection with large step changes in the reference signal. What was the problem with the implementation above? (1 p)

**b.** Give a correct pesudo-code implementation of the state-feedback and observer controller. (1 p)

**9.**

**a.** Implement a binary semaphore using the basic synchronization provided by java monitors. The public interface to the class is

```
public class Semaphore {
    /* Take the semaphore */
    public void give(void);

    /* Give the semaphore */
    public void take(void);

    /* Try to take the semaphore, return false without
     * blocking if the semaphore is already taken, otherwise
     * take the semaphore and return true. */
    public boolean tryTake(void);
}
```

Note that the keyword `synchronized` has been left out from the specification, though it could be needed in the solution.

(2 p)

**10.** You are implementing your control system in a microcontroller that has three different levels of preemptive interrupt handlers. Level 1 is the highest possible level, while Level 2 and 3 have respectively lower priority. In your implementation you have used an interrupt handler of Level 1 (I1) to handle your controller computation and programmed a timer to trigger the interrupt every $50\,ms$. You have timed your controller code and you are sure that its execution takes exactly $5\,ms$.

The same microcontroller is used by one of your colleagues to control a different plant. The controller for this plant is also implemented using interrupt handlers but it is composed of two different tasks. The first task, is written in an interrupt handler of Level 3 (I3), it samples the plant and it is executed every $10\,ms$ with a duration of $2\,ms$. The second task (I2) computes and actuates the control signal with a larger period, being executed in an interrupt handler of Level 2 every $100\,ms$.

**a.** Assuming that the deadline of each task is equal to the next triggering instant, what is the maximum duration that task I2 can have to ensure that no task is missing its deadline? (1 p)

**b.** Your colleague was able to optimize her code, and task I2 terminates its computation in $1\,ms$. However, due to a failure in the microcontroller, the timers do not work properly and all the interrupts are triggered every $10\,ms$, but they are not synchronized. The first trigger of each interrupt occurs at different times (I1 = $1\,ms$, I2 = $2\,ms$, I3 = $0\,ms$) and after that they are triggered every $10\,ms$). Assuming that switching from one interrupt handler to the other takes no time, what is the maximum amount of time it takes for each of the interrupts to be terminated (from the triggering moment to its completion). Justify your answer. (1.5 p)

**11.** The following code shows the standard implementation of the PID controller that we use in the course. It is based on a forward difference approximation of the I-part and a backward difference approximation of the D-part. Write the corresponding code if you instead use a backward approximation for the I-part and a forward approximation for the D-part (the latter is actually not a good idea because it may lead to instability, but in this problem we still want you to do this). Write the code so that the input-ouput latency is minimized, including using as many intermediate parameters (similar to ad and bd) as possible. (3 p)

```
// Code executed in the Regul thread

y := AIIn(ychan); // Sample y
e := yref - y;
D := ad*D + bd*(y - yold);
v := K*(beta*yref - y) + I + D;
u := sat(v,umax,umin)
DaOut(u,uchan); // Output u
I := I + (K*h/Ti)*e + (h/Tr)*(u - v);
yold := y;

// Code that is executed only when a controller parameter is updated
ad := Td/(Td-N*h);
bd := -K*N*ad;
```