LUND INSTITUTE
OF TECHNOLOGY
Lund University

Department of
AUTOMATIC CONTROL

Real-Time Systems

Exam December 17, 2013, hours: 8.00–13.00

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. In the course two different semantics for counting semaphores have been presented, the ordinary semaphore implementation (pg. 42-43 in RTCS) and the alternative semaphore implementation (pg. 45 in RTCS).

Match each implementation with one of the following descriptions. Also describe what it is in the definition of `wait` and `signal` that achieves this functionality. (2 p)

> A: When a task leaves a critical section, i.e., release the semaphore, it checks if there are any tasks waiting for the semaphore and in that case wakes up the highest priority task among those, so that it may compete for the semaphore with other tasks.
>
> B: When a task leaves a critical section, i.e., release the semaphore, it checks if there are any tasks waiting for the semaphore and in that case wakes up the highest priority task among those and hands over the semaphore to it.

2. Consider the following discrete time system

$$x(k+1) = \begin{bmatrix} 0.75 & 0.1 \\ 0 & 0.5 \end{bmatrix} x(k) + \begin{bmatrix} 1 \\ 2 \end{bmatrix} u(k)$$

$$y(k) = \begin{bmatrix} 1 & 2 \end{bmatrix} x(k).$$

The input to this system is constant, $u(k) = 5$.

a. Simulate the system with the initial state $x(0) = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$, and calculate $y(3)$ (the output at time $k = 3$). (1.5 p)

b. Will the system converge, and in that case, to what output value? (1 p)

3. A process has the discrete state-space representation

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

a. Design a dead-beat predictor-form observer for the process. (1 p)

b. The observer will be used together with a state feedback on the form $u(k) = -Lx(k) + l_r r(k)$. Design a dead-beat state feedback. Determine the closed loop pulse transfer function from reference signal r to output y and decide l_r to make sure that $y(k) = r(k)$ in stationarity. (2 p)

c. Implement the observer and the state feedback based on the following pseudo-code skeleton. In order to get full points the code should be written so that the input-output latency is minimized. Insert code at the places indicated by . . . . . Make sure that all variables are properly declared. (2.5 p)

```
private double x1Hat = 0, x2Hat = 0, u = 0, y = 0, r = 0;
...

while (1) {
  y = getY();
  r = getReference();
  ....
  outputU(u);
  ....
  sleep()
}
```

4. Two controllers are to be implemented in the same hardware. They both use a common data logging function which is running in a separate task. The tasks are executing under fixed priority scheduling. The table below shows periods (T_i), deadlines (D_i), worst-case execution times (C_i) and the priorities for the tasks (low value means high priority).

Task	T_i	D_i	C_i	Priority
Controller 1	8	8	2	1
Controller 2	12	12	4	2
Logger	T_{logger}	T_{logger}	5	3

How often the logger task should run has, however, not yet been decided.

a. What choices of logger task period (T_{logger}) guarantee that the controller tasks meet their deadlines according to the approximate rate monotonic analysis? (1 p)

b. What choices of logger task period guarantee that all tasks (including the logger) meet their deadlines according to the approximate rate monotonic analysis? Since there are two formulae for the approximate schedulability test, you should use the formula that gives the least restrictive result. (1 p)

c. What choices of logger task period guarantee that all tasks (including the logger) meet their deadlines if response time analysis is used? (2 p)

5. To create the wobble bass sound that is typical for dubstep music one of the most important digital effects is the lowpass filter where the cut-off frequency is decided by a low frequency oscillator (LFO). A continuous time low pass filter with cut-off frequency f_c (in Hz) has the transfer function

$$G(s) = \frac{2\pi f_c}{s + 2\pi f_c}.$$

Since we will be filtering sound, the cut-off frequency f_c will be in the range $[20, 20000]$. The sampling frequency of the filter is 96 kHz.

a. Approximate the low pass filter using forward difference and find the pole(s) of the discrete filter at $f_c = 20$ and $f_c = 20000$. Is there any problem with the poles of the system and how they approximate the original filter? (1 p)

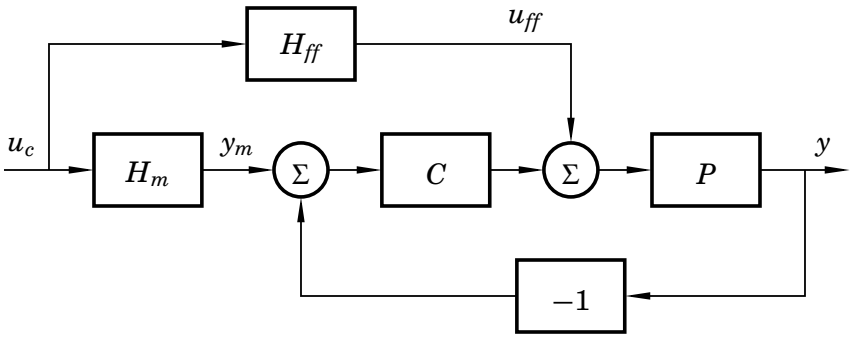


Figure 1 Feedforward Design

b. Approximate the filter again but using Tustin's approximation instead. Find the pole(s) of the discrete filter at $f_c = 20$ and $f_c = 20000$. Is there any problem with the poles of the system and how they approximate the original filter? (1 p)

c. Write a Java implementation of the Tustin approximation based on the following code skeleton.

```
public static final double h = 1.0 / 96000;
// Insert variables for old values here...
public double tustinLP(double u, double fc) {
   double y;
   // Insert code here...
   return y;
}
```

The method `tustinLP` will be called once for every sample. `u` is the input to the filter, `fc` is the cutoff frequency in Hz. The method should return the filtered samples. (1 p)

6. The block diagram for feedforward control design using the transfer function approach is shown in Fig. 1.

In the figure

- H_m – is a model that describes the desired servo performance, and
- H_{ff} – is a feedforward generator that makes y follow y_m.

a. Assume that P, C, and H_m are given. How should H_{ff} be chosen to achieve perfect model following? (1 p)

b. In order for H_{ff} to be implementatable, it must be both causal and stable. What are the requirements in order for this to be the case? The requirements should be stated in terms of conditions on H_m, P, and C. (2 p)

c. When this is applied to PID control of the double tank process the following holds.

The continuous-time process is given by

$$G(s) = \frac{3}{(1+60s)^2}.$$

The ZOH-sampled process is given by

$$P(z) = \frac{0.003627(z + 0.9672)}{(z - 0.9512)^2}.$$

The continuous-time reference model is given by

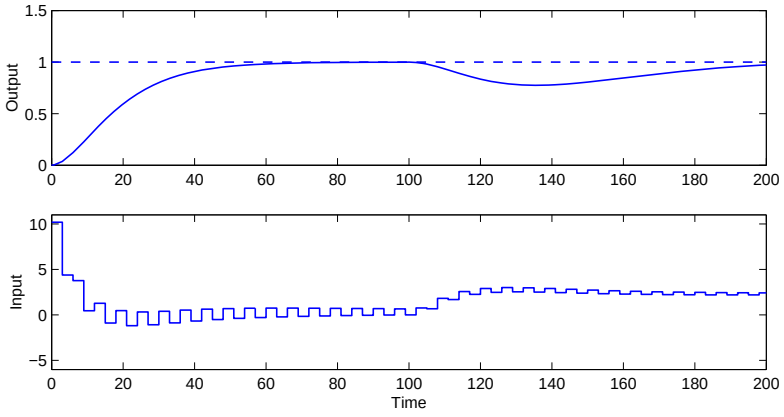
$$G_m(s) = \frac{1}{(1 + 10s)^2}.$$

The ZOH-sampled reference model is given by

$$H_m(z) = \frac{0.036936(z + 0.8187)}{(z - 0.7408)^2}.$$

The controller used was an ordinary PID controller.

When the closed loop system was simulated with a step in the reference signal and with a load disturbance entering at $t = 100$, the response looked as follows



What is the reason for the ringing in the control signal? Show the problem by calculating the transfer function from u_c to u (where u is the input to the process P). How should the problem be avoided? (2 p)

7. In the `java.util.concurrent` package a number of synchronization mechanisms are available. One of them is the barrier. Assume that we have a number of threads executing part of an overall application followed by a point at which they must coordinate their results. The barrier is simply a waiting point where all the threads can sync up either to merge results or to safely move on to the next part of the application. The Java class implementing barriers is called `CyclicBarrier`. The reason it is called cyclic is that it can be re-used after the waiting threads are released.

A slightly simplified version of the interface to the `CyclicBarrier` class is:

```
public class CyclicBarrier {
  public CyclicBarrier(int parties);
  public int await();
}
```

The core of the class is the `await()` method. This is called by each thread that needs to wait until the required number of threads are waiting on the

barrier. In the constructor the number of threads (parties) using the barrier is specified. This number is used to trigger the barrier; the threads are all released when the number of threads waiting on the barrier is equal to the number of parties specified.

Each thread that calls the `await()` method gets back a unique return value. This value is related to the arrival order of the thread at the barrier. The first thread to arrive gets a value that is one less than the number of parties, the last thread to arrive will get a value of zero.

The barrier is very simple. All the threads wait until the number of required parties arrive. Upon arrival of the last thread, the waiting threads are released, and the barrier can be reused. Since the barrier is so simple it is straightforward to implement it using the Java synchronization mechanisms that are part of the course (`synchronized`, `wait()`, `notify()` and `notifyAll()`).

Implement the class `CyclicBarrier` with the interface and semantics described above. In order to get full points you must ensure that all threads that are released really will be released, also if some other thread has started to reuse the barrier before the released threads have executed. Correct handling of exceptions is not required. You may also disregard any spurious wakeup issues. (3 p)