LUND INSTITUTE
OF TECHNOLOGY
Lund University

Department of
AUTOMATIC CONTROL

Real-Time Systems

Exam April 2, 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/

1. Consider the following discrete-time controller structure

$$u(k) = k_1 u(k-1) + k_2 e(k) + k_3 e(k-1)$$

where $e(k)$ is the control error and k_1 to k_3 are the controller parameters.

a. How should the controller parameters be selected for the controller to have integral action? (1 p)

b. How should the controller parameters be selected for the controller to have derivative action? (1 p)

2. Assume a system with the dynamics

$$y(k+2) + 0.5y(k) = u(k),$$

a. Find the pulse transfer function from u to y and give the poles of the system. (2 p)

b. Find a state-space realization of the system (1 p)

c. Determine the stationary gain from u to y. (1 p)

3. The block diagram in Figure 1 shows how the PID Controller (2DOF) block in MATLAB/Simulink is implemented internally.

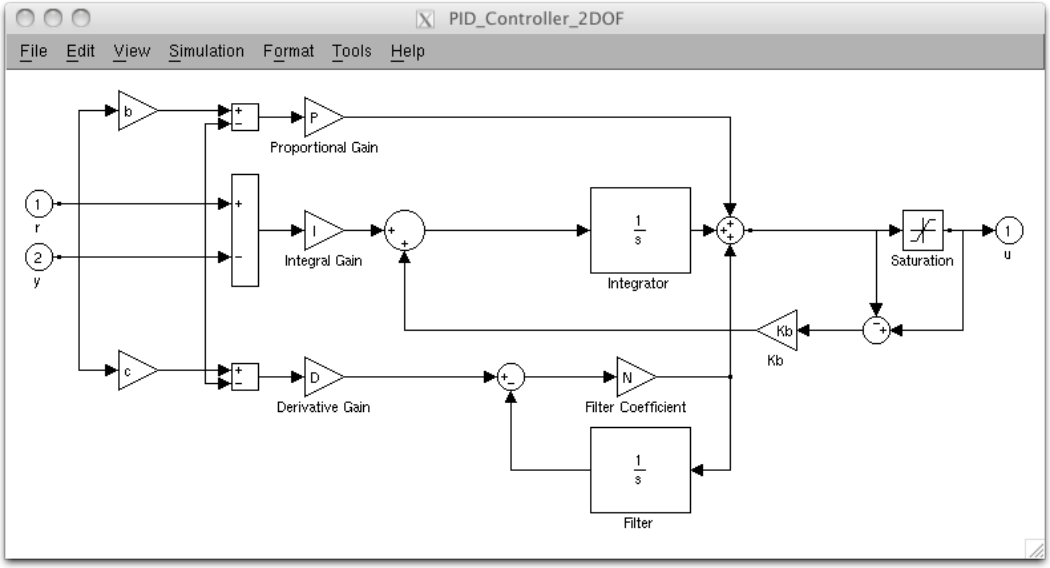

Figure 1 2-DOF PID controller in Simulink.

a. Explain the purpose of each of the gain blocks b, c and Kb. (1 p)

b. Explain the purpose of the Filter Coefficient and Filter blocks in the lower part of the diagram. (1 p)

4. The following set of tasks needs to run on a single core microprocessor that uses fixed-priority scheduling.

Name	WCET [ms]	Cycle time [ms]	Deadline [ms]
A	1	3	2
B	3	7	6
C	2	150	10

a. Is the set of tasks schedulable according to the approximate schedulability analysis assuming rate-monotonic priority assignment? (1 p)

b. Is the set of tasks schedulable if fixed-priority scheduling with rate-monotonic priority assignment is used? (1 p)

c. Is the set of tasks schedulable if fixed-priority scheduling with deadline monotonic priority assignment is used? (1 p)

5. Assume that you are interested in observing a system which has been modeled to have dynamics in the frequencies up to but not including f_0 Hz.

a. Why is it possibly advantageous to have a sample frequency larger than $2f_0$? Give exactly one reason. (1 p)

b. Assume that a disturbance exists which has frequency $f_1 = 3f_0$. For what choice of sampling frequencies will the disturbance be aliased into the frequency range of interest? (2 p)

6. Multicore CPU:s are becoming very common these days. To exploit the full capacity of such a CPU, a program must execute in several threads at the same time. Sometimes the majority of the work that has to be done can be split into parallel parts in a straightforward manner. The main program can still run in a single thread, branching out into multiple threads to handle a big job, and resuming execution in the single thread once the job is done.

A simple interface for a parallellizable job is

```
public interface Job {
  int getNumParts();     // How many parallell parts?
  void doPart(int part); // Execute one part of the job
}
```

The job consists of parallel parts numbered 0, 1, 2, ..., $n-1$. To finish the job, all parts should be done exactly once; however, they may be done in parallel and in any order.

To support this model of parallellization, it is useful to have a class that takes a `Job` and splits it among multiple threads. Incomplete java code for such a class is shown below:

```
public class Pool {
  private Job job;
  private int nextPart, numParts;

  public void runParallel(Job job, int numCores) {
```

```
    this.job = job;
    numParts = job.getNumParts();
    nextPart = 0;

    Worker[] workers = new Worker[numCores];
    for (int i=0; i < numCores; i++) {
      workers[i] = new Worker();
      workers[i].start();
    }

    for (int i=0; i < numCores; i++) {
      try {
        workers[i].join(); // The calling thread will be blocked
                           // until the workers[i] thread has terminated
      }
      catch (InterruptedException e) {}
    }
  }

  private int getNextPart() {
    if (nextPart < numParts) {
      int part = nextPart;
      nextPart++;
      return part;
    }
    else return -1;
  }

  private class Worker extends Thread {
    // ...
  }
}
```

a. Define the `run` method of the inner `Thread` class `Worker` so that `Pool.runParallel` works as expected. (2 p)

b. Two worker threads should never try to execute the same job part. How is this guaranteed in the code/how can you change the code to guarantee it? (1 p)

c. The simplest way to use the `Pool` class is to create an instance and then call its `runParallel` method once. Can the instance also handle

1. When `runParallel` is called several times in sequence from the same thread?
2. When `runParallel` is called several times from different threads?

Why (not)? (1 p)

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

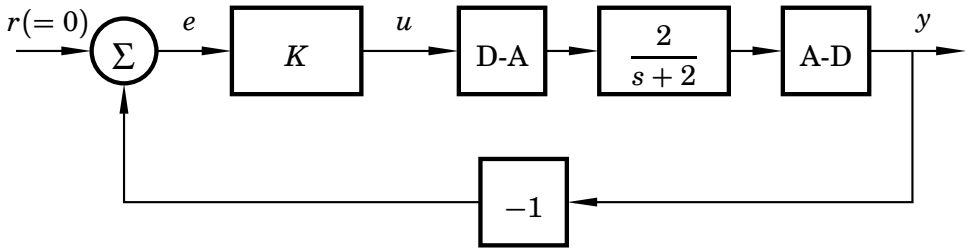

Figure 2 Computer-controlled system.

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

Assume that the execution time of the controller is not negligible, i.e. that there is a delay, L, between `readInput()` and `writeOutput(u)`. Given that L is constant and $L < h$, what is the largest value of L for the system to be stable? Hint: The stability conditions for a second-order discrete-time system with the characteristic polynomial $A(z) = z^2 + a_1 z + a_2$ are given by

$$\begin{aligned} a_2 &< 1 \\ a_2 &> -1 + a_1 \\ a_2 &> -1 - a_1 \end{aligned}$$

(2 p)

8. You should make a fixed-point implementation of a PI controller without any anti-windup. The controller equations in continuous time are given by

$$U(s) = K\left(\beta R(s) - Y(s) + \frac{1}{sT_i}(R(s) - Y(s))\right)$$

a. Discretize the controller using forward difference approximation and the sampling time h. The controller should consist of one part that calculates the output and one part that updates the integral term. It should be possible to update the value of T_i without any bump in the control signal (1 p)

b. The coefficients of the controller should be represented by 16-bit signed integers. All calculations that can be done in advance should be performed (so for instance there is no point in converting β to fixed-point). Choose the number of fractional bits for the coefficients to be the largest possible value that does not cause overflow. You may use the same number of fractional bits for all coefficients. Compute the fixed-point representations of the coefficients. Use the values $K = 5$, $\beta = 0.63$, $T_i = 0.3$ and $h = 0.1$. (2 p)

c. Implement the controller in C by filling in the skeleton below. The signals r, y and u are all represented using zero fractional bits and should be in the range $[-512, 511]$. If the desired control signal is outside the range of

u, it should be limited. The integral part `I`, defined in the code skeleton, is initialized to zero when the program starts and can be accessed from within the functions. `I` is stored in a signed 32-bit variable and should be represented with the same number of fractional bits as the coefficients. Values that are not needed until next time sample should be calculated after the call to `write_output(u)`. (2 p)

```
// define your parameters here
...

int32_t I = 0;

// Called periodically every 0.1 s.
void do_control(int16_t r) {
  ...
  int16_t y = read_input();
  ...
  write_output(u);
  ...
}
```