# Control Systems and Real Time Scheduling Systems

Based on:

- **Towards the integration of Control and Real Time Scheduling, Anton Cervin, Licentiate Thesis, Lund University, 2000.**
- **Integrated Control and Real Time Scheduling, Anton Cervin, Ph.D. Thesis, Lund University, 2003.**

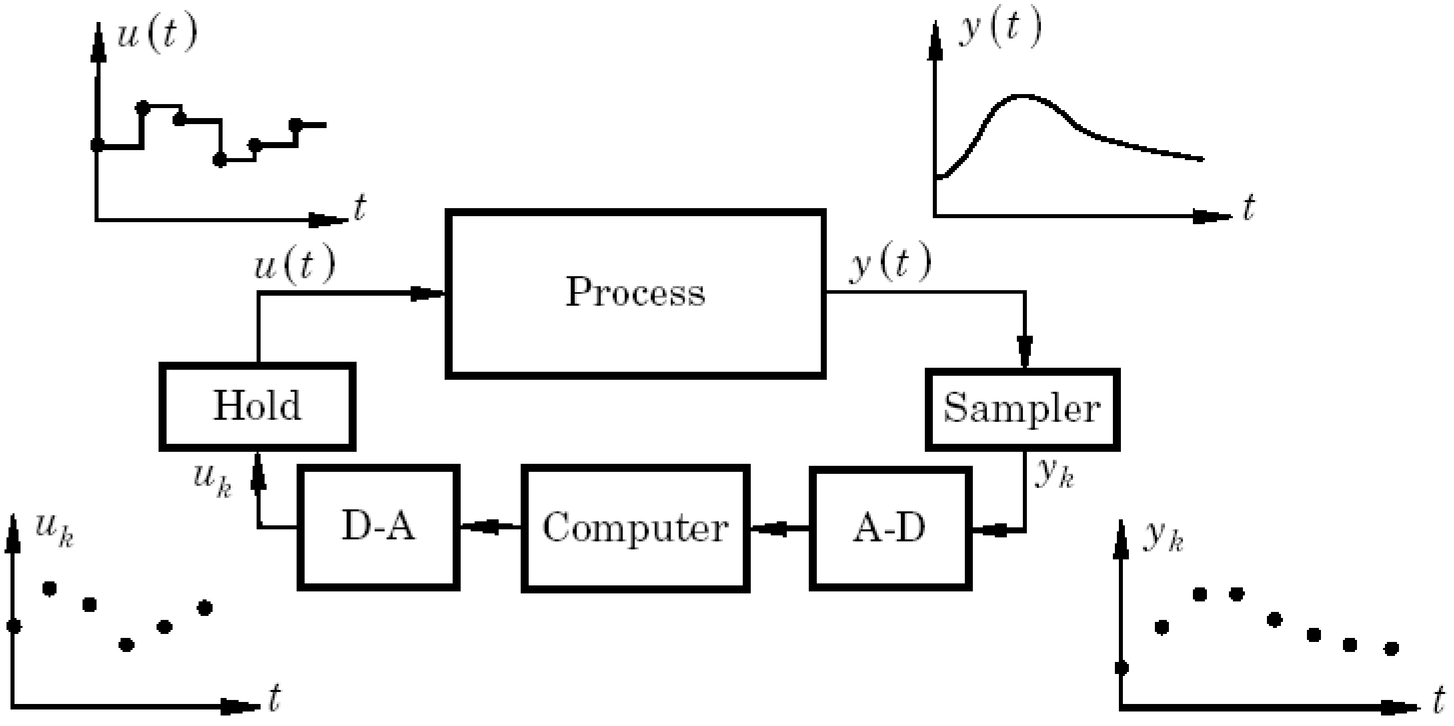


**Figure 2.** Sampled control loop.

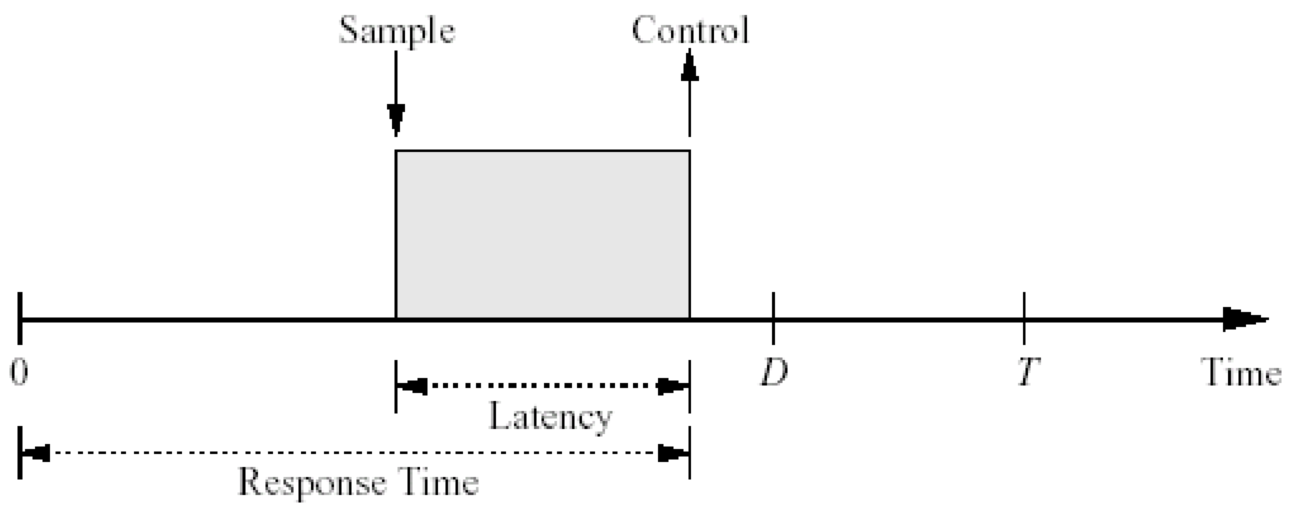


**Figure 2.** Illustration of the relationship between deadline $(D)$, response time, and input-output latency for a control task. The latency is bounded by the response time, which in turn is bounded by the deadline. The task is assumed to be released at time zero.

$$\begin{aligned} \frac{dx}{dt} &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t), \end{aligned}$$

$$\begin{aligned} x(t) &= e^{A(t-t_k)}x(t_k) + \int_{t_k}^{t} e^{A(t-s')}Bu(s')\,ds' \\ &= e^{A(t-t_k)}x(t_k) + \int_{t_k}^{t} e^{A(t-s')}\,ds'\,Bu(t_k) \quad (u \text{ piecewise constant}) \\ &= e^{A(t-t_k)}x(t_k) + \int_{0}^{t-t_k} e^{As}\,ds\,Bu(t_k) \quad (\text{variable change}) \\ &= \Phi(t,t_k)x(t_k) + \Gamma(t,t_k)u(t_k) \end{aligned}$$

From this the values at $t = t_{k+1}$ are given by

$$\begin{aligned} x(t_{k+1}) &= \Phi(t_{k+1}, t_k)x(t_k) + \Gamma(t_{k+1}, t_k)u(t_k) \\ y(t_k) &= Cx(t_k) + Du(t_k) \end{aligned}$$

where

$$\begin{aligned} \Phi(t_{k+1}, t_k) &= e^{A(t_{k+1} - t_k)} \\ \Gamma(t_{k+1}, t_k) &= \int_0^{t_{k+1} - t_k} e^{As} ds \ B \end{aligned}$$

$$\begin{aligned} x(kh+h) &= \Phi x(kh) + \Gamma u(kh) \\ y(kh) &= Cx(kh) + Du(kh) \end{aligned}$$

$$\begin{aligned} \Phi &= e^{Ah} \\ \Gamma &= \int_0^h e^{As}\,ds\,B \end{aligned}$$

A wide range of discrete-time controller design methods can then be applied, e.g. pole placement design, linear quadratic design, or model predictive control. The sampling intervals for discrete-time control designs are normally based on the desired speed of the closed loop system. A common rule-of-thumb is that one should sample 4 to 10 times per rise time $T_r$ of the closed loop system.

$$N_r = \frac{T_r}{h} \approx 4 \text{ to } 10$$

This gives relatively long sampling intervals, compared to what is used when discretization-based design is used. The long sampling interval also means that it may take long time before, e.g., load disturbances are detected by the controller. The reason for this is that the disturbances are not synchronized with the sampling.

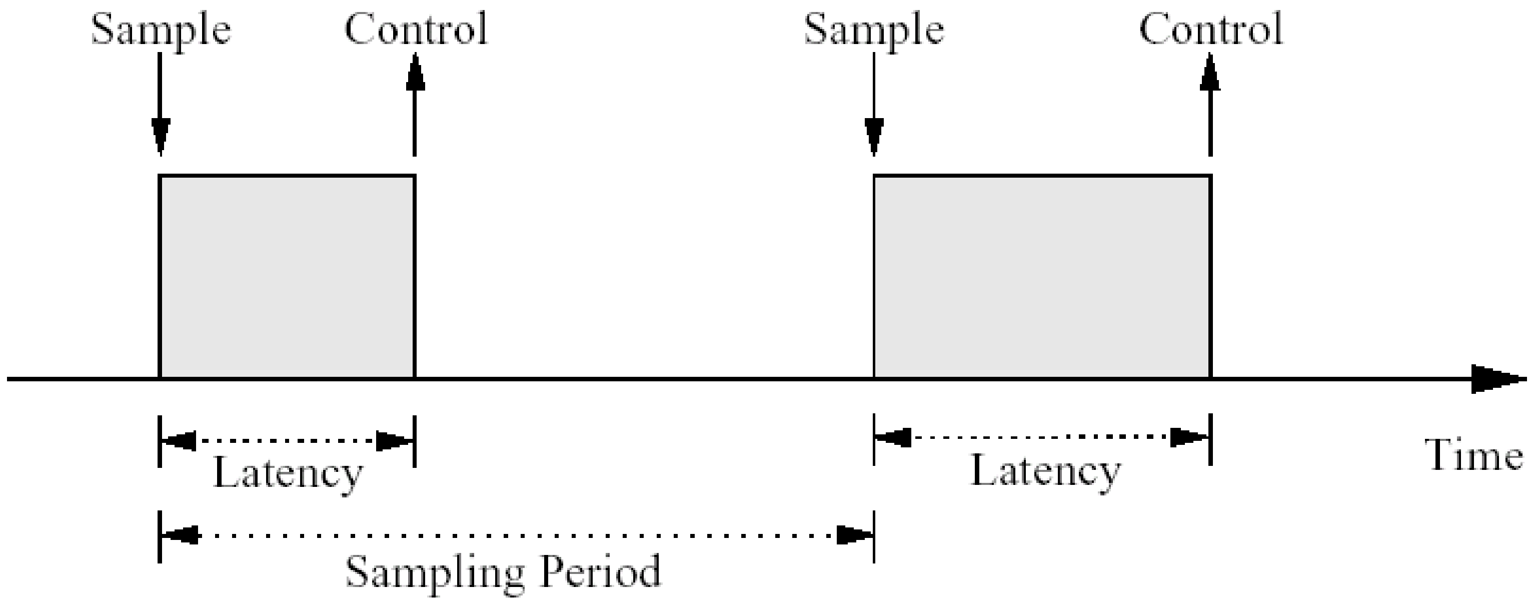


**Figure 4.** Basic timing constraints of a control loop.

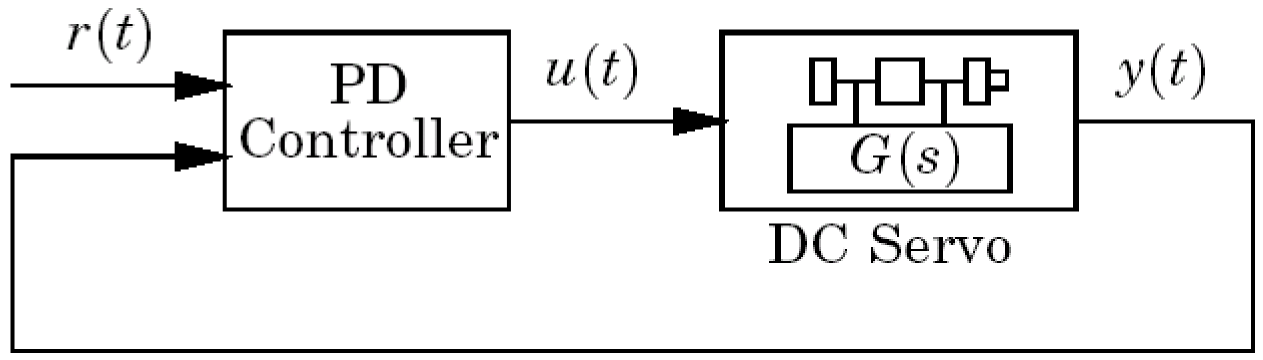


**Figure 5.** A DC Servo is being controlled by a PD controller.

Let the servo be described by the continuous-time transfer function

$$G(s) = \frac{1000}{s(s+1)}.$$

A good discrete-time implementation of the PD controller, which includes filtering of the derivative part, is

$$\begin{aligned} P(t) &= K(r(t) - y(t)), \\ D(t) &= a_d D(t-h) + b_d(y(t-h) - y(t)), \\ u(t) &= P(t) + D(t), \end{aligned}$$

where $a_d = \frac{T_d}{Nh+T_d} D(t-h)$ and $b_d = \frac{NKT_d}{Nh+T_d}$.

A nominal sampling period of $h = 10$ ms is chosen, and the PD controller is tuned to give a fast and well-damped response to set-point changes. The resulting parameters are $K = 1$, $T_d = 0.04$, and $N = 30$. The parameters $a_d$ and $b_d$ are normally precalculated, assuming that the sampling interval is constant.

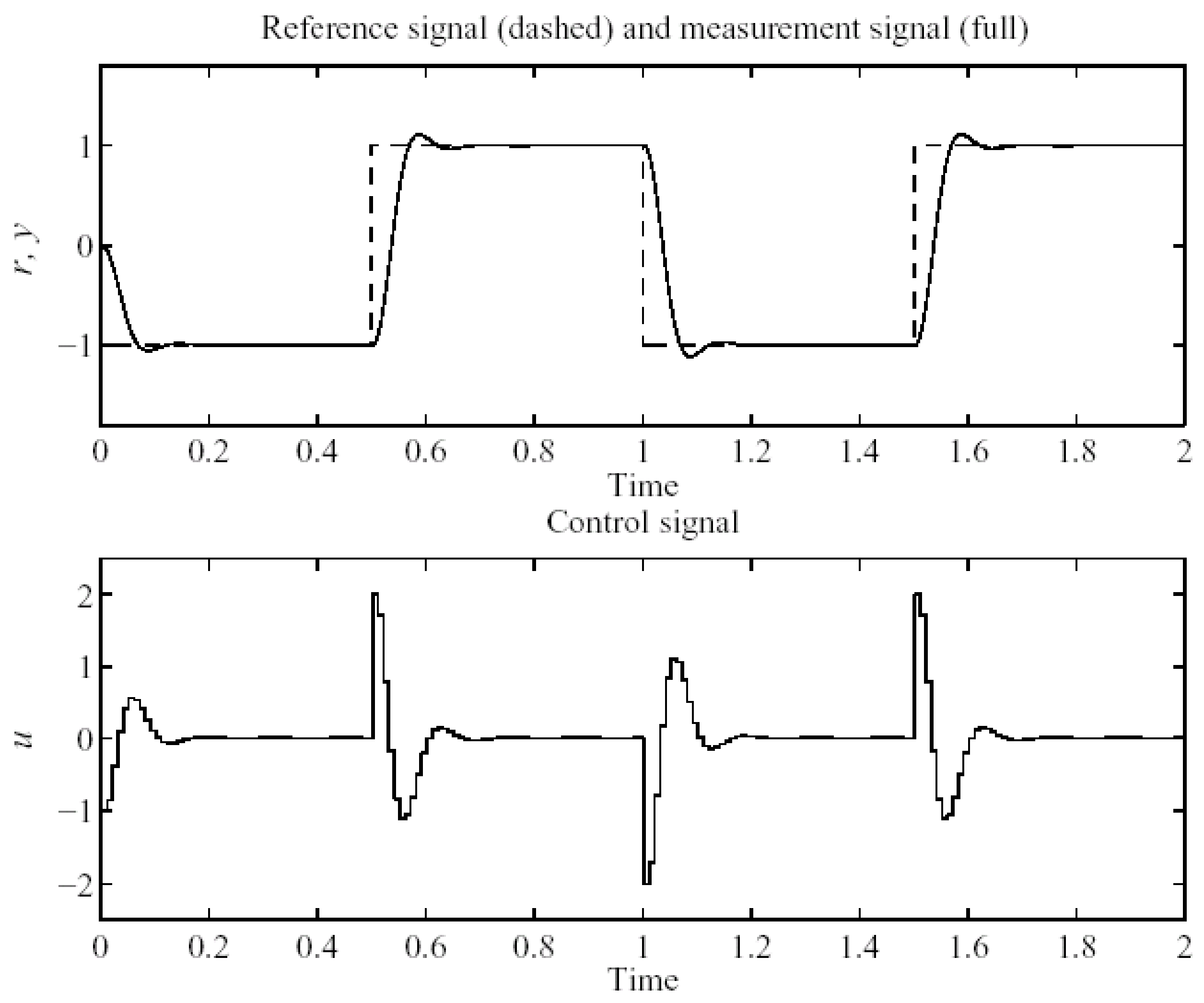


**Figure 6.** When no sampling jitter is present, the control performance is good.

## Sampling Jitter

A second simulation, where the actual sampling interval varies randomly between $h_{min} = 2$ ms and $h_{max} = 18$ ms, is shown Fig. 7. The discrepancy between the nominal and the actual sampling interval causes the controller to repeatedly take either too small or too large actions. The resulting performance is quite poor. This is especially visible in the control signal.

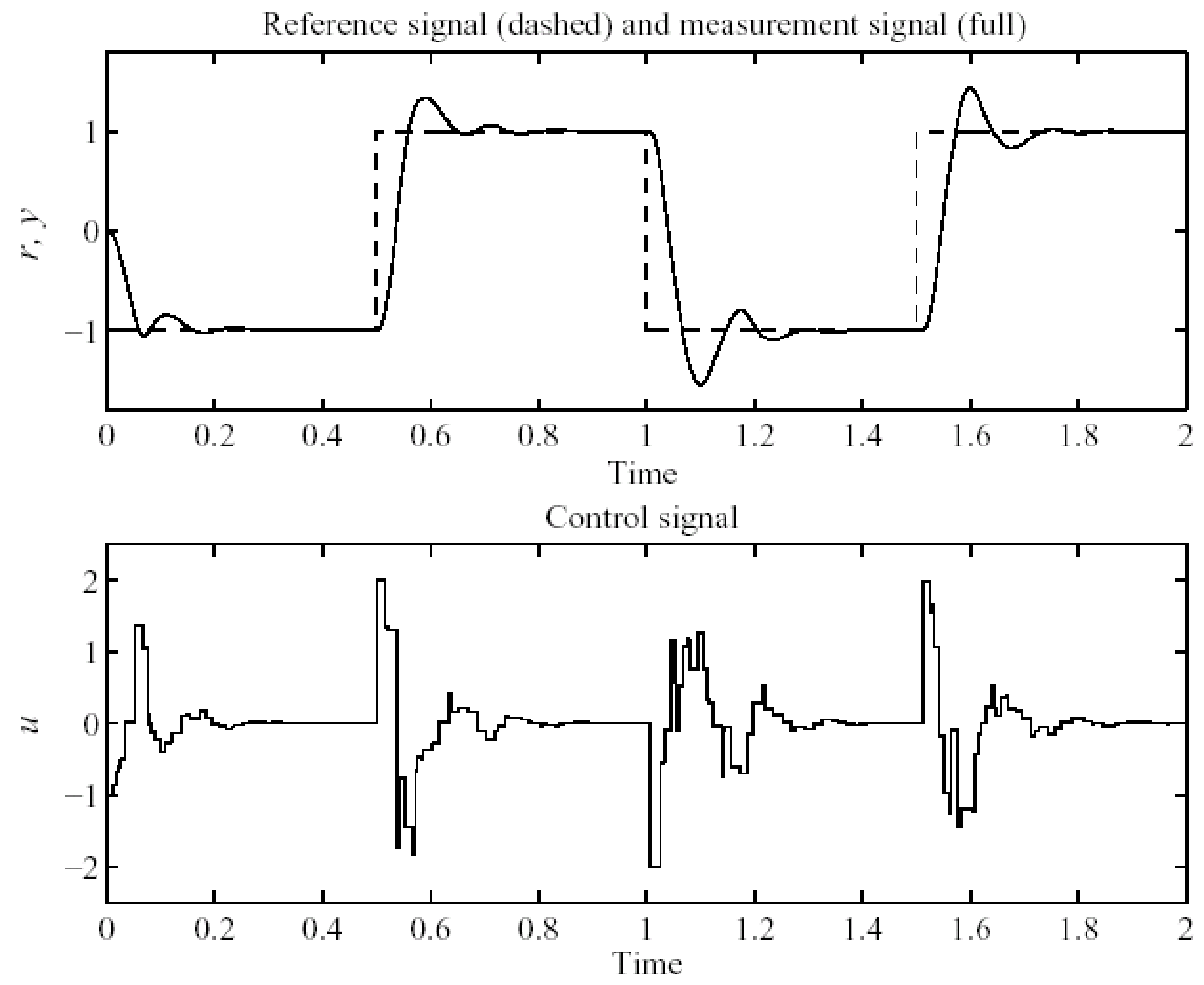


**Figure 7.** Sampling jitter causes the control performance to degrade.

Finally, in a third simulation, the controller is redesigned to compensate for the varying sampling interval. This is done by measuring the actual sampling interval and recalculating the controller parameters $a_d$ and $b_d$ at each sample. Fig. 8 shows that this version of the controller handles the sampling jitter well. The performance is almost as good as in Fig. 6 where there was no jitter at all.

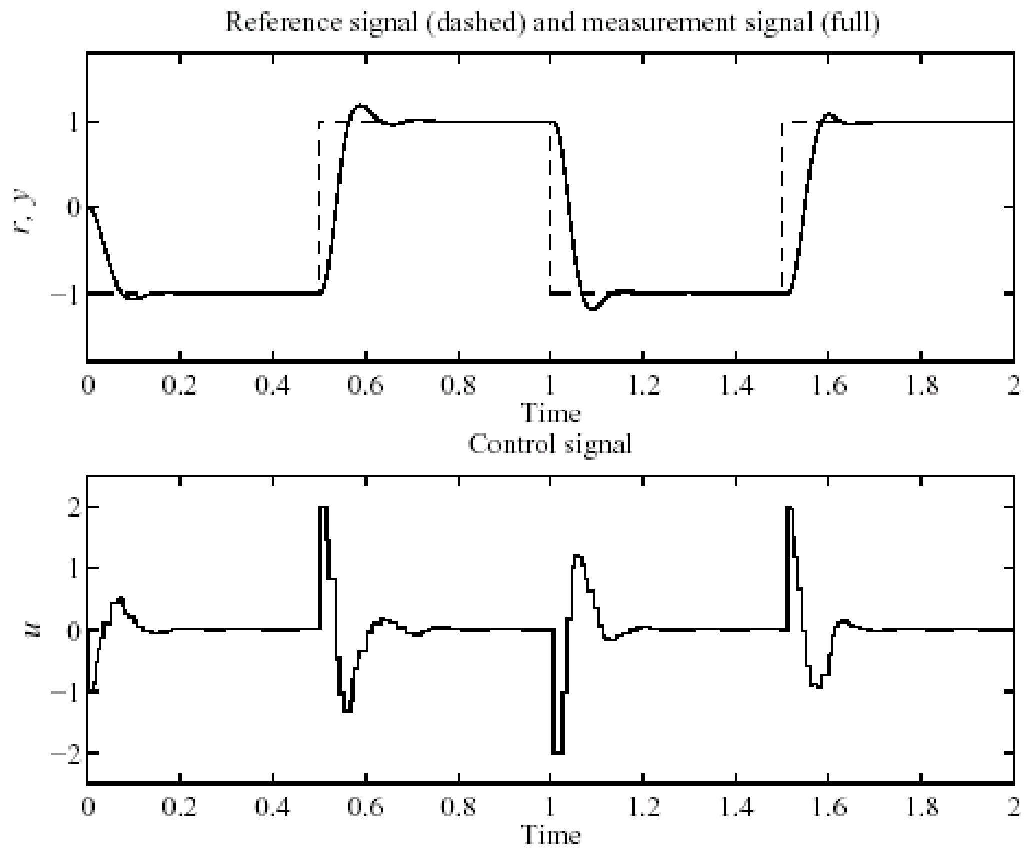


**Figure 8.** When compensating for the sampling jitter, the control performance is good again.

## Input-Output Latency

a. Division of Algorithm into two parts:
    1. Caculate Output
    2. Update State
b. Try to ensure that the delay is constant, i.e., Jitter Free
c. Design the controller to be robust against jitter
d. Compensate the delay at each sample changing the Controller parameters

# b. Constant Delay

The sampled system will now be on the form (assuming that the

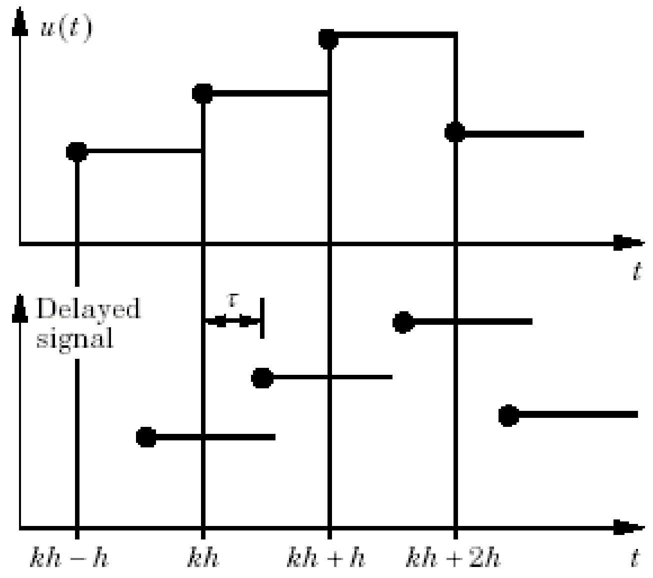


**Figure 9.** Relationship among $u(t)$ and the delayed signal $u(t-\tau)$, and the sampling instances.

$$\begin{aligned}\frac{dx}{dt} &= Ax(t) + Bu(t-\tau)\\ y(t) &= Cx(t) + Du(t)\end{aligned}$$

$$x(kh+h) = \Phi x(kh) + \Gamma_0 u(kh) + \Gamma_1 u(kh-h),$$

where

$$\begin{aligned}\Phi &= e^{Ah}\\ \Gamma_0 &= \int_0^{h-\tau} e^{As}\,ds\,B\\ \Gamma_1 &= \int_{h-\tau}^{h} e^{As}\,ds\,B\end{aligned}$$

Example 2—Input-Output Latency Jitter

Again, consider PD control of the DC servo from Example 1. (The sampling jitter is assumed to be zero.) A delay is now introduced from the sampling to the output action. First, the input-output latency is constant and equal to 0.007 ms. The controller is retuned assuming this delay, and the resulting parameters are $K = 1$, $T_d = 0.045$, and $N = 100$. The simulation result is shown in Fig. 11. It is not possible to get as good performance as in the previous example due to the time delay.

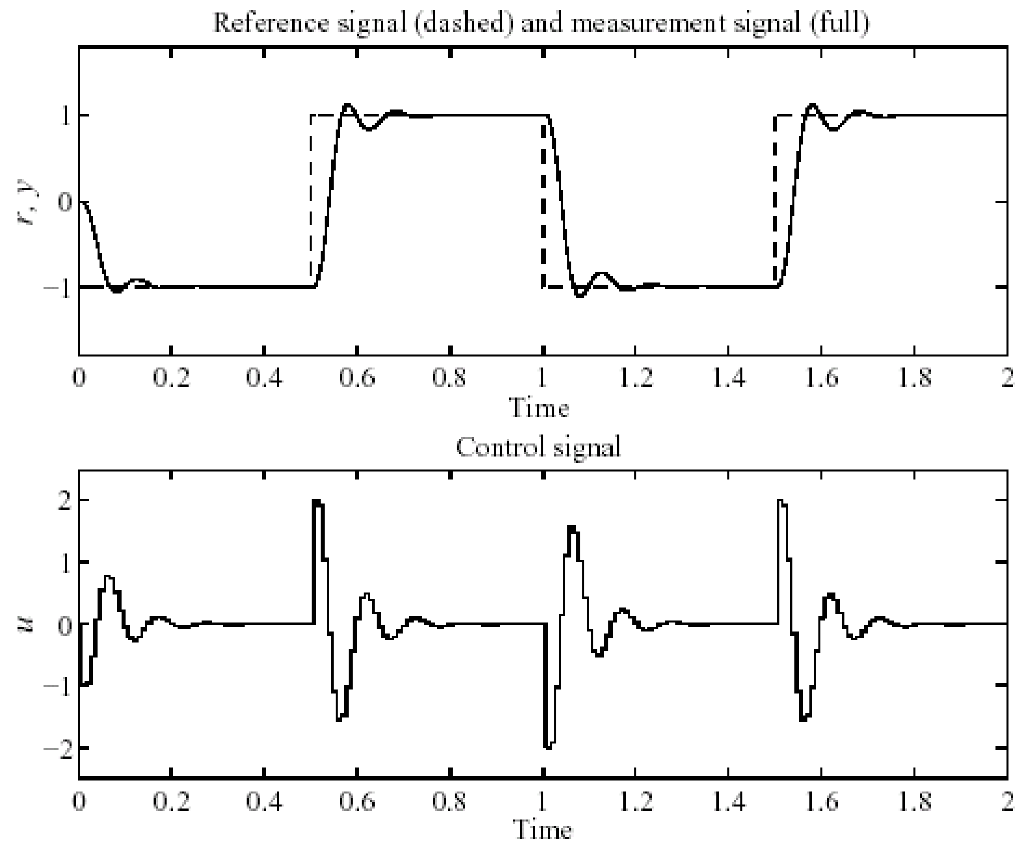


**Figure 11.** The controller can be tuned to handle a constant input-output latency.

Next, the input-output latency is randomly varying between 0.002 ms and 0.012 ms. Even though the average delay is the same as before, the performance is now worse, as shown in Fig. 12.

$\square$

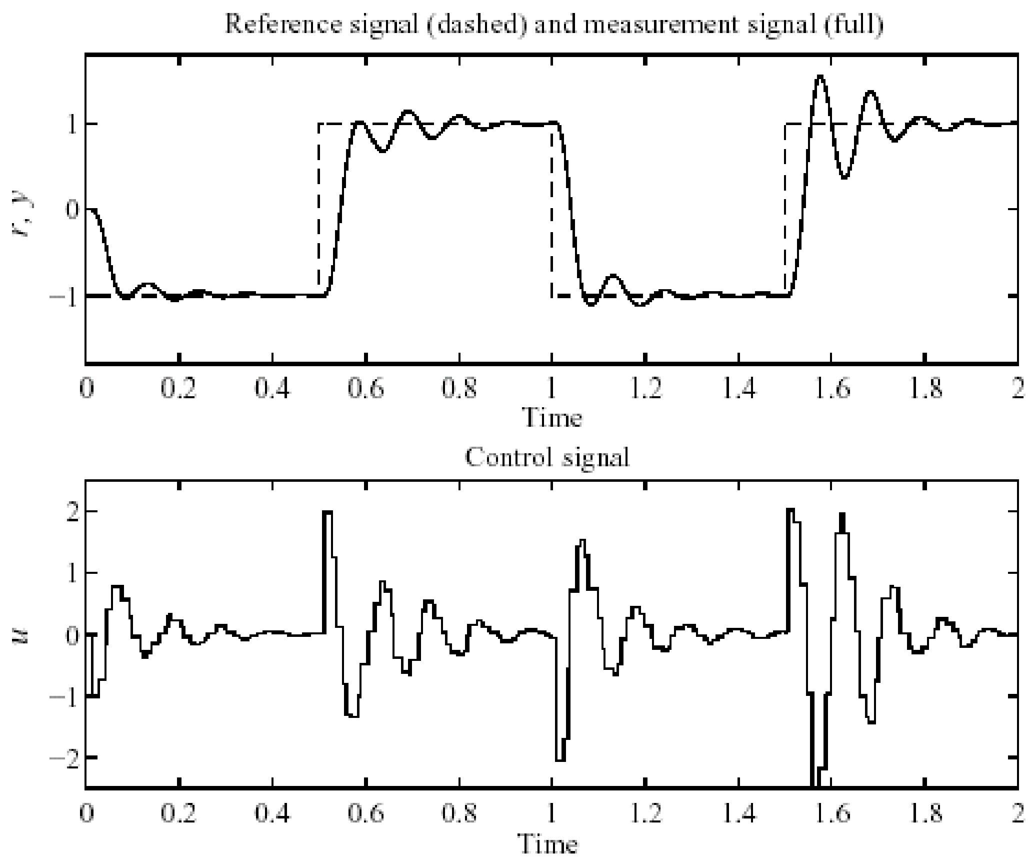


**Figure 12.** Variable input-output latency degrades the control performance.

## Control Loop Timing and Scheduling

Scheduling theory can be used to analyze the time variations and delays in control loops when they are implemented as real-time tasks. Understanding the control requirements, the implementation could be made such that the resulting delay and the jitter are small.

The following example shows that a simple-minded implementation of control loops can introduce a lot of jitter and delays:

EXAMPLE 3
Three control loops with different sampling periods are implemented in a priority-preemptive real-time OS. The task code for each control loop looks like this (this would be a good implementation in a single-task system):

```
t = CurrentTime;
LOOP
  AD-Conversion;
  ControlAlgorithm;
  DA-Conversion;
  t := t + h;
  WaitUntil(t);
END
```

Assume that the execution time is 2 ms for all three tasks, and that the sampling periods are $T_1 = 12$ ms, $T_2 = 8$ ms, and $T_3 = 5$ ms. Fixed priorities are assigned to the tasks according to the rate-monotonic theory. Figure 2 shows the execution graph of the three control tasks when released at time zero. Task 3 has the shortest period, thus the highest priority, and executes with perfect periodicity. Tasks 1 and 2, on the other hand, are frequently preempted. The preemption causes variations in both the sampling period and in the input-output latency.

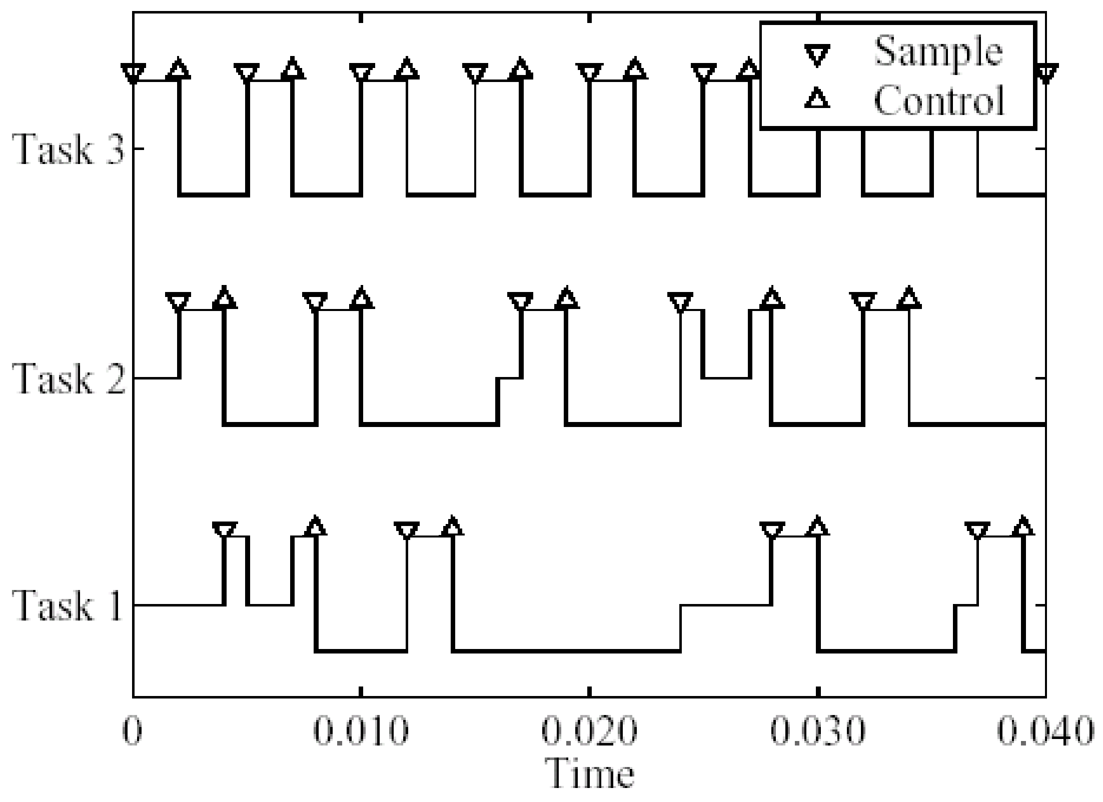


**Figure 13.** The activation graph (high=running, medium=preempted, low=sleeping) of the three control tasks in Example 3

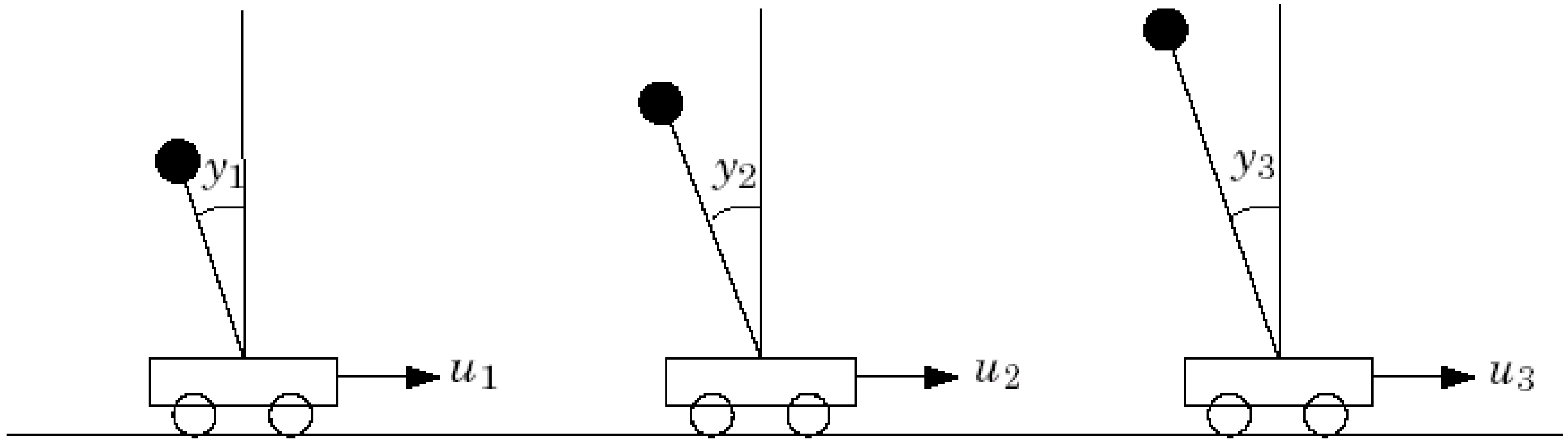


**Figure 1.2** Three inverted pendulums should be stabilized using one computer.

h1=20ms, h2=29ms, h3=35ms

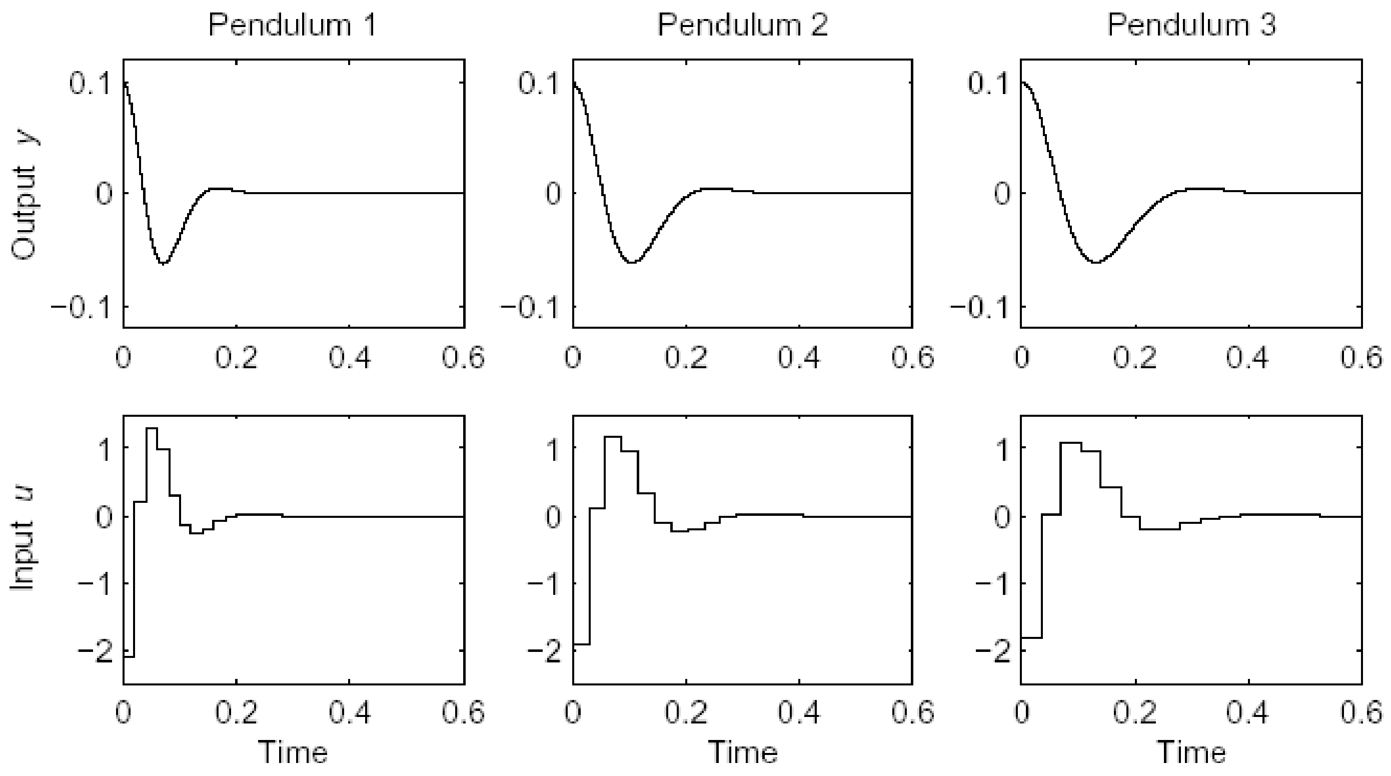


**Figure 1.3** Ideal simulation of the inverted pendulum system.

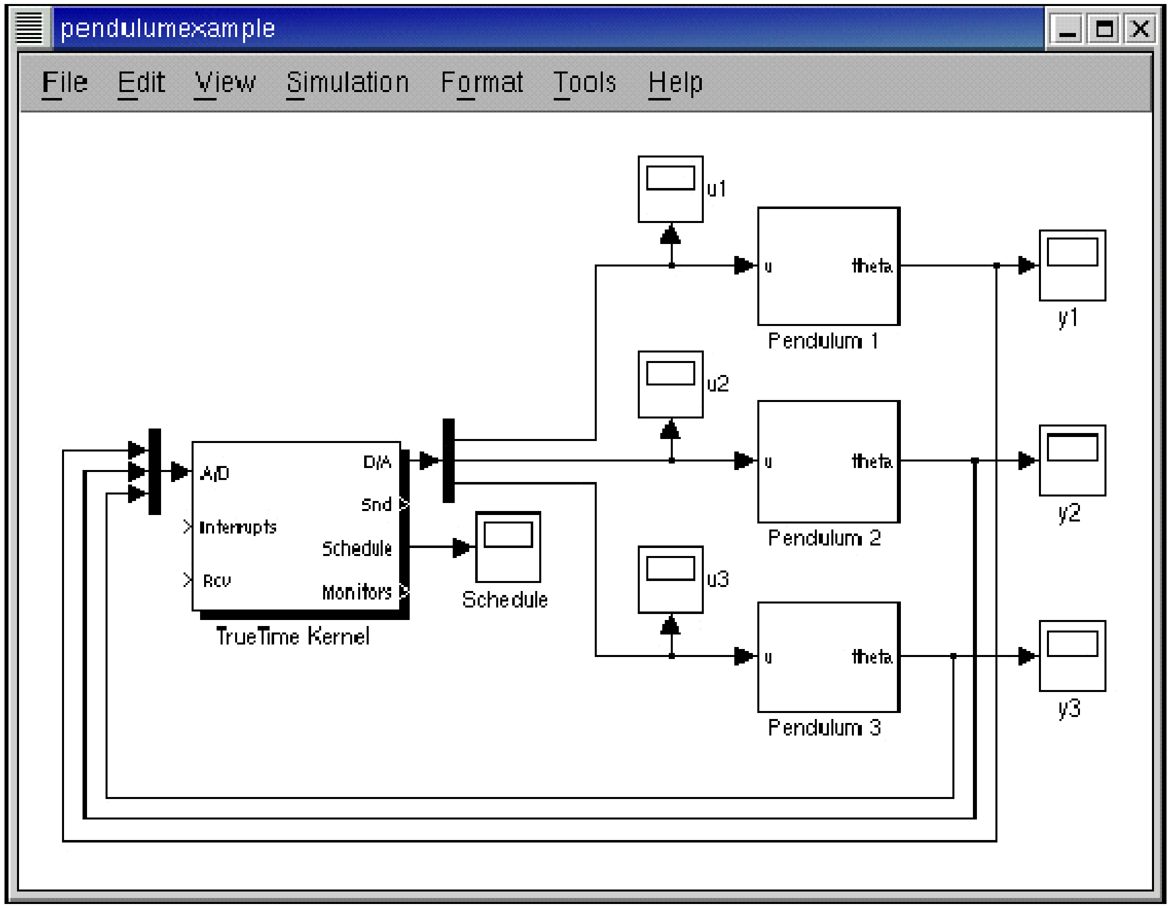
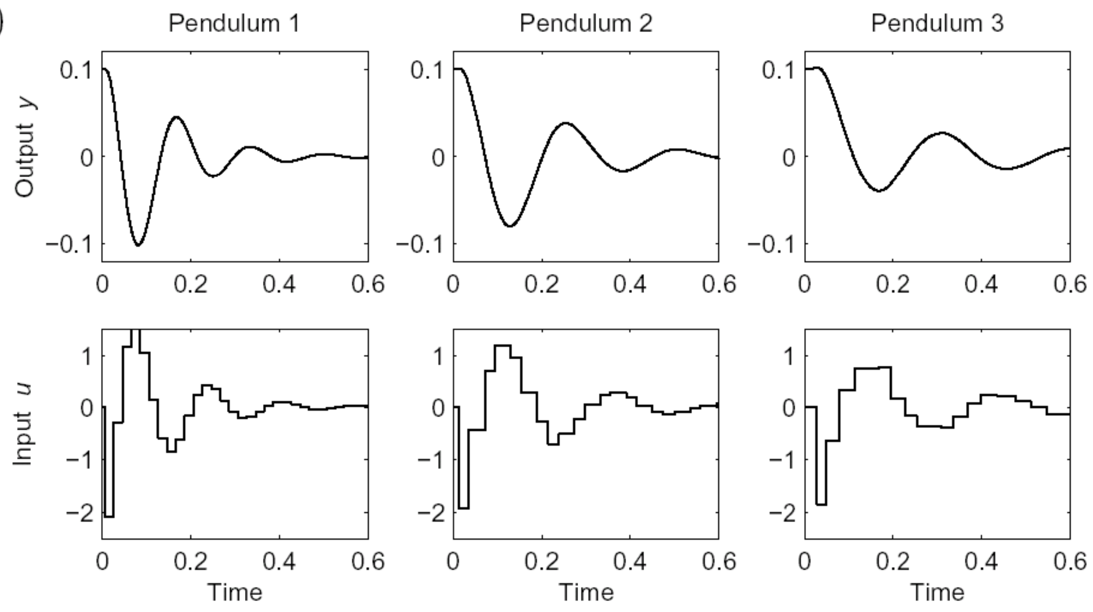


**Figure 1.4** TrueTime simulation model of the inverted pendulum system. The TrueTime Kernel block simulates a real-time operating system that executes user-defined tasks.

## Rate Monotonic Scheduling

- Execution Time C=7ms
- Task with the highest rate has higher priority
- Utilization test U=0.79

(a)



(b)

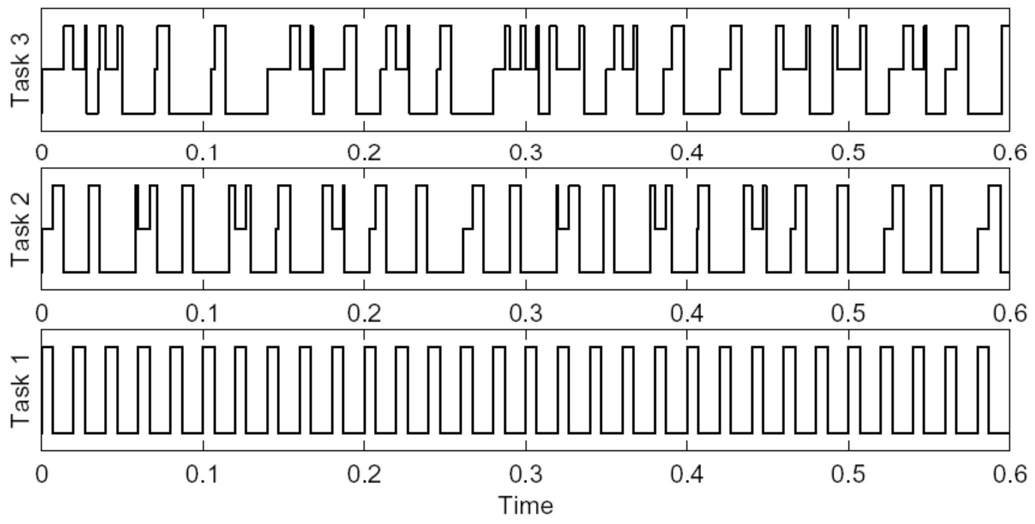


**Figure 1.5** TrueTime simulation of the inverted pendulum system under rate-monotonic scheduling: (a) control performance, and (b) task schedule. The scheduling-induced latencies in the control loops deteriorate the performance. (In the schedule plots, a high level means that the task is running, a medium level that it is preempted, and a low level that it is sleeping.)

# Statistics

- Ls= Sampling latency, time from the release of the task to the actual start of the task
- Lio= input-output latency, time from the sampling operation to the actuation operation
- h=sampling interval

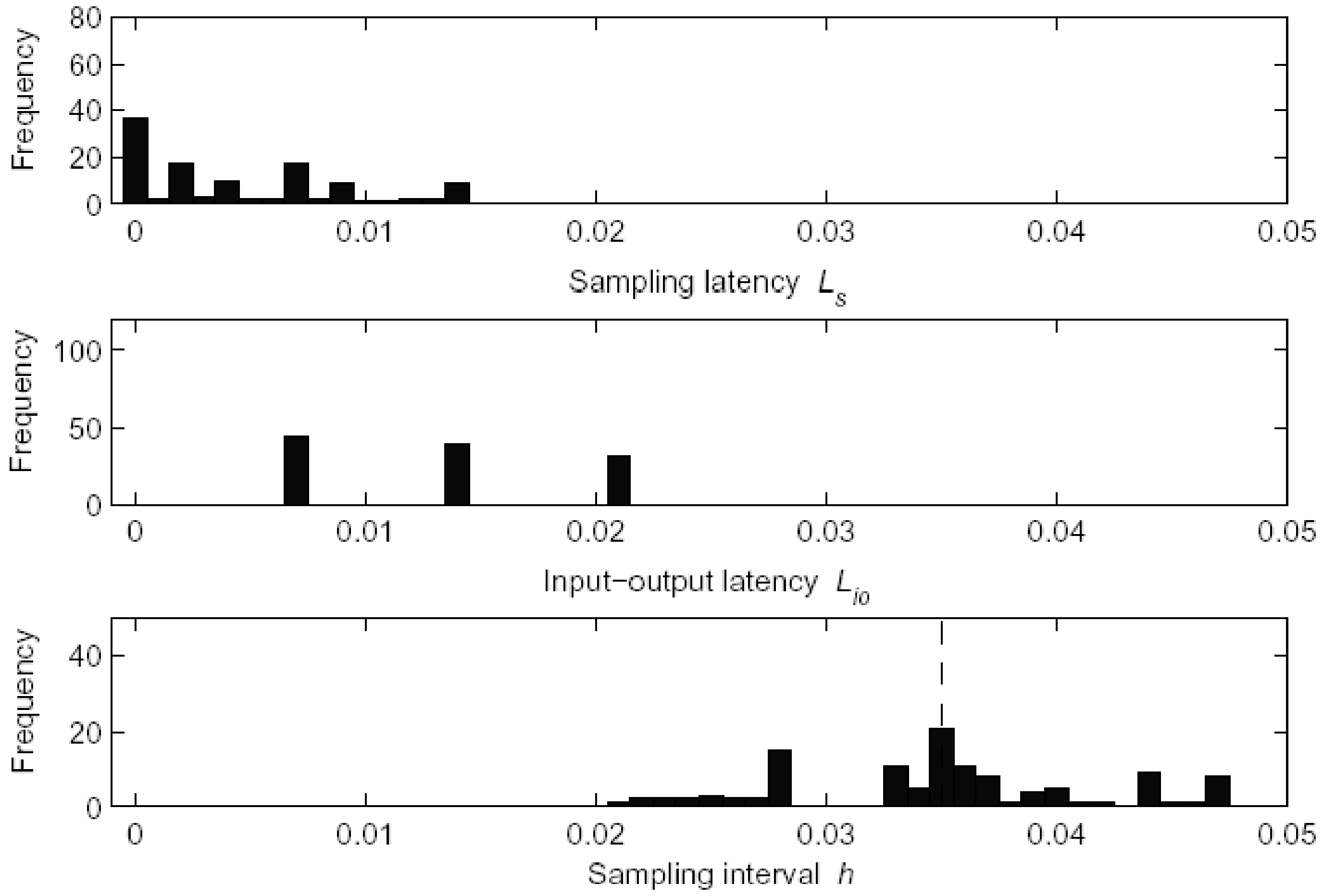
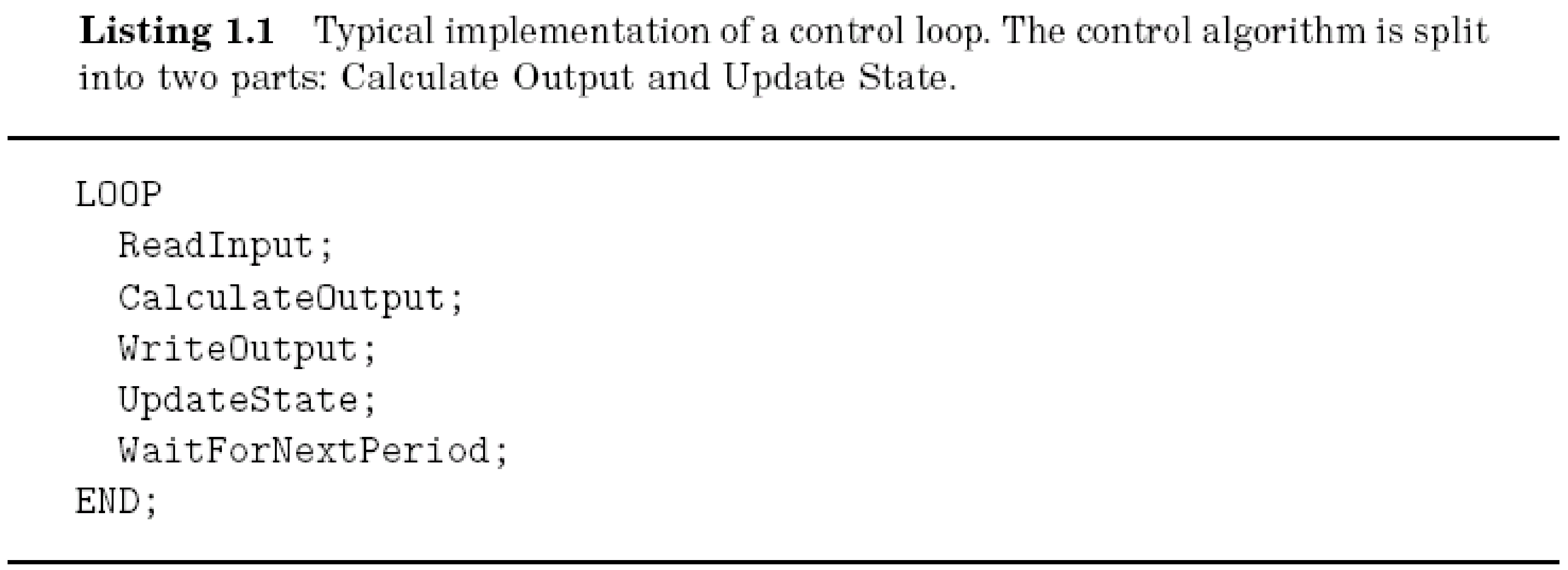


**Figure 1.6** Distribution of the sampling latency, the input-output latency, and the sampling interval for Task 3 under rate-monotonic scheduling. The controller is designed for the sampling interval $h = 0.035$.

## Sub-task scheduling



**Listing 1.1** Typical implementation of a control loop. The control algorithm is split into two parts: Calculate Output and Update State.

```
LOOP
  ReadInput;
  CalculateOutput;
  WriteOutput;
  UpdateState;
  WaitForNextPeriod;
END;
```

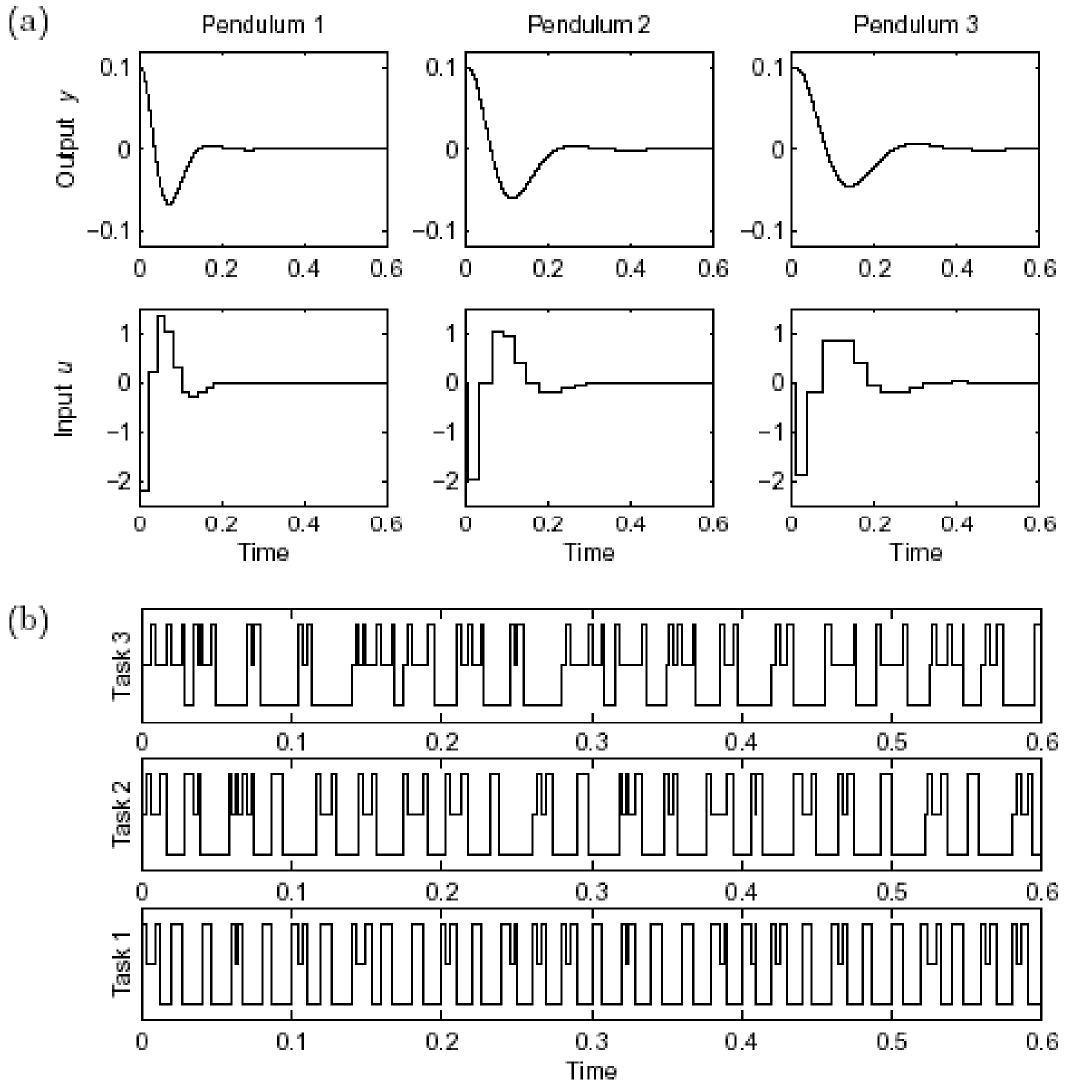


**Figure 1.10** TrueTime simulation of the inverted pendulum system under subtask scheduling: (a) control performance, and (b) task schedule. The performance is very close to the ideal case (see Figure 1.3). The scheduling strategy increases the number of context switches compared to rate-monotonic scheduling (see Figure 1.5).

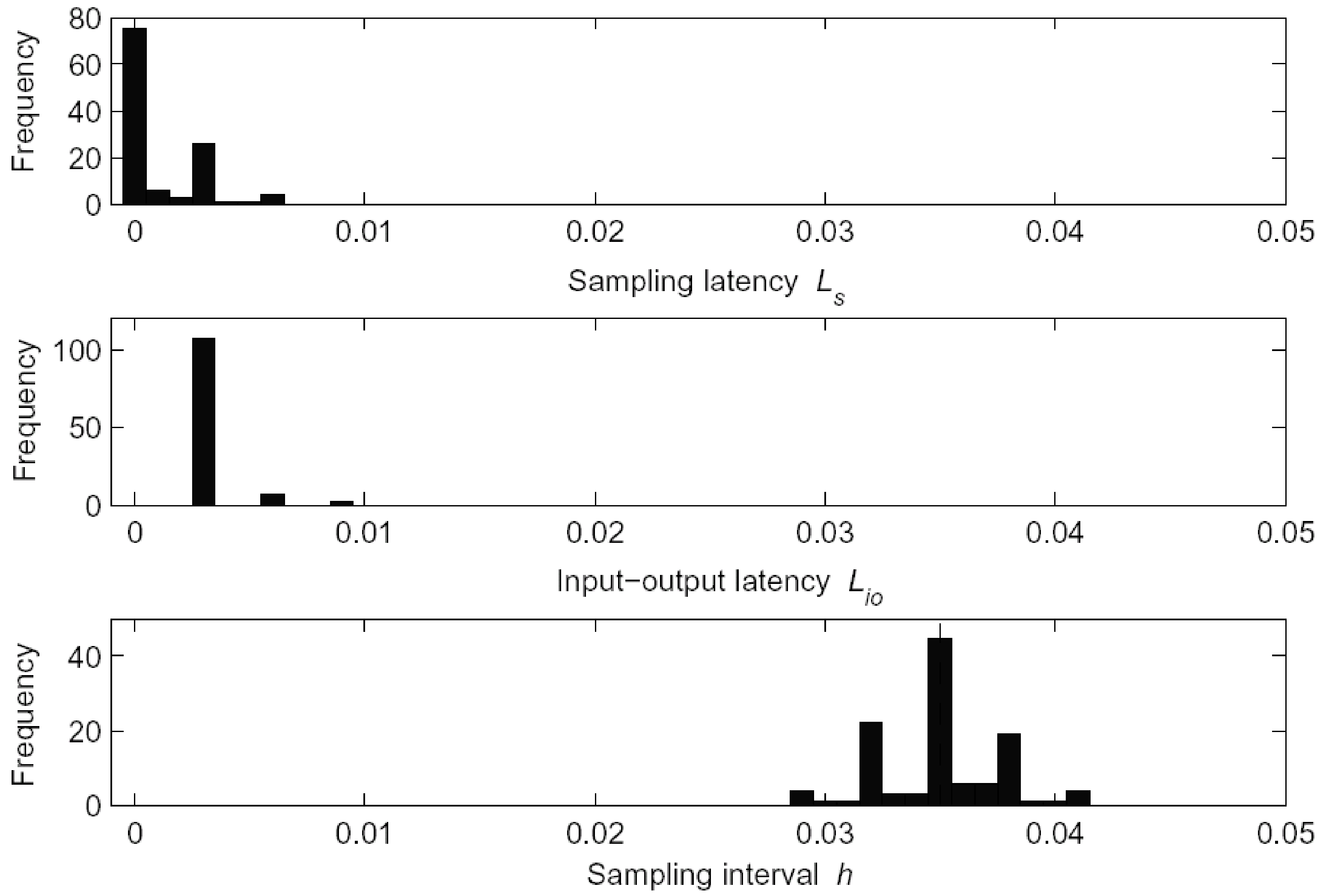
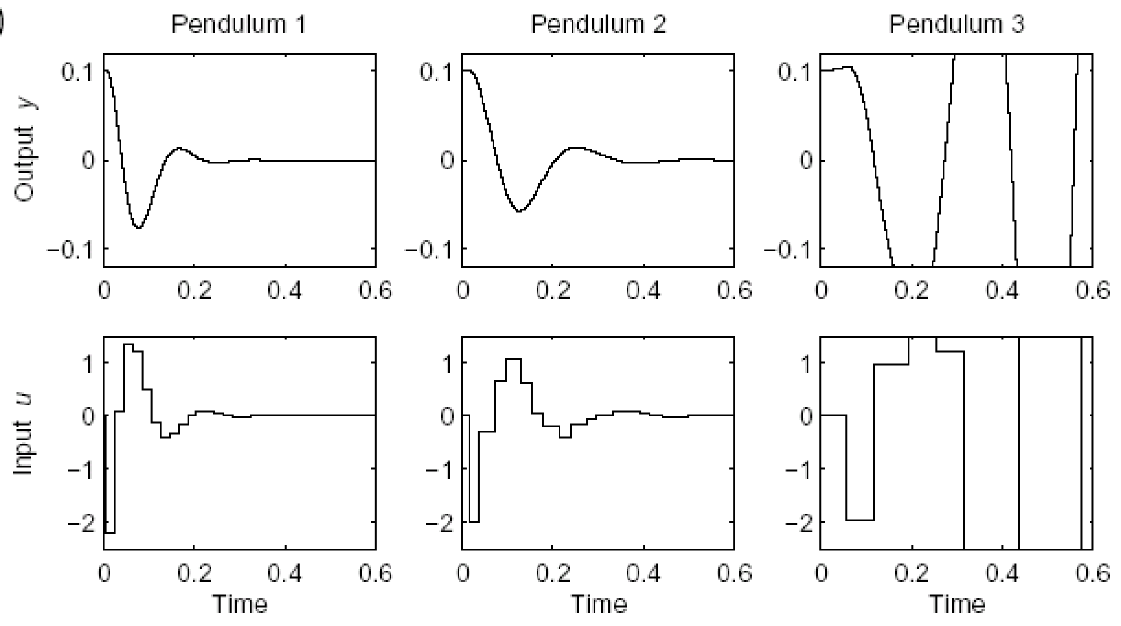


**Figure 1.11** Distribution of the sampling latency, the input-output latency, and the sampling interval for Task 3 under subtask scheduling. Compared with the rate-monotonic scheduling case (Figure 1.6), the latencies are shorter and the sampling interval is more closely centered around the nominal interval $h = 0.035$.

## Rate monotonic scheduling overloaded CPU

- Designed for C=7ms
- Actual C=10ms
- Utilization test U=1.13

(a)



(b)

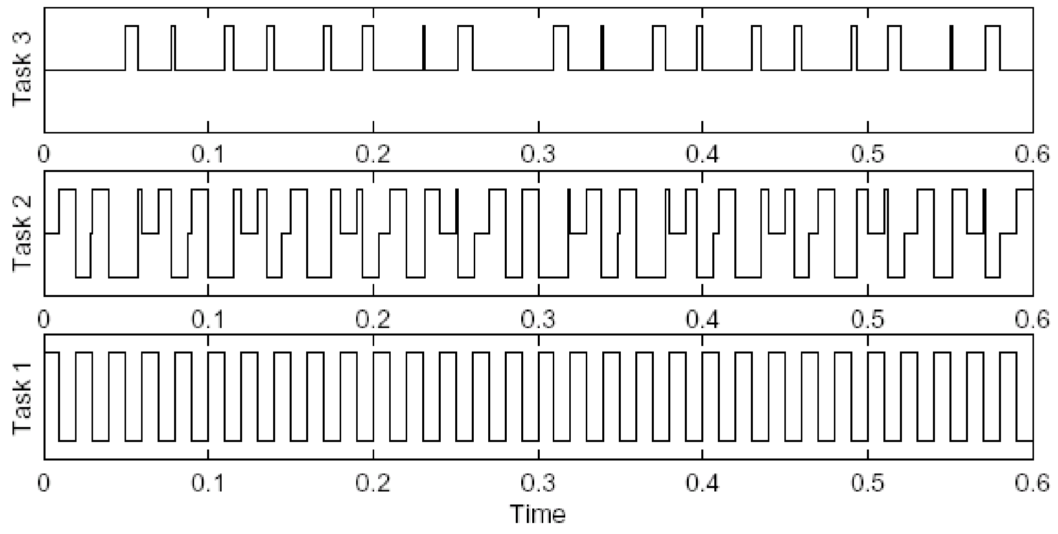
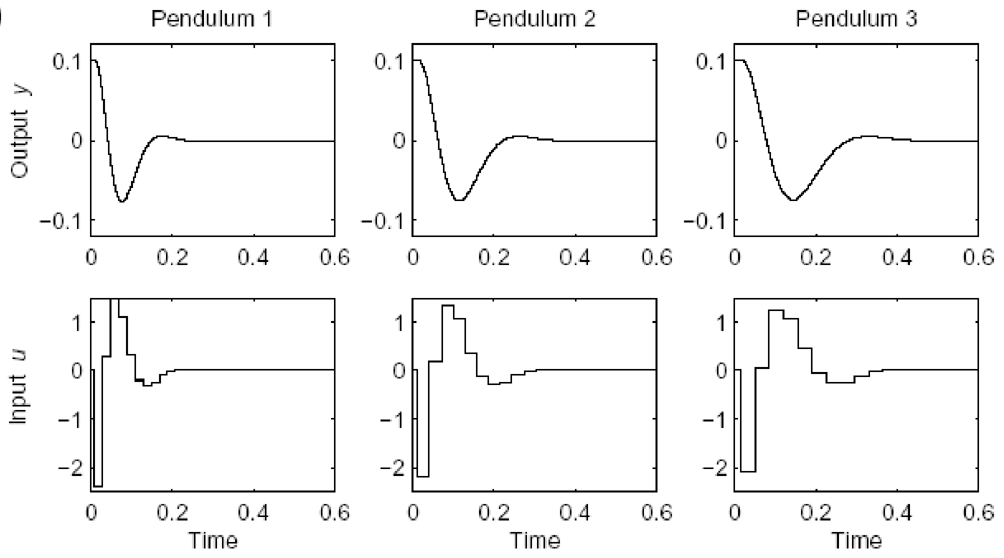


**Figure 1.12** TrueTime simulation of the inverted pendulum system under rate-monotonic scheduling and an overloaded CPU: (a) control performance, and (b) task schedule. Task 3 is preempted most of the time, causing the control loop to be destabilized.

## Control Server Model

(a)



(b)

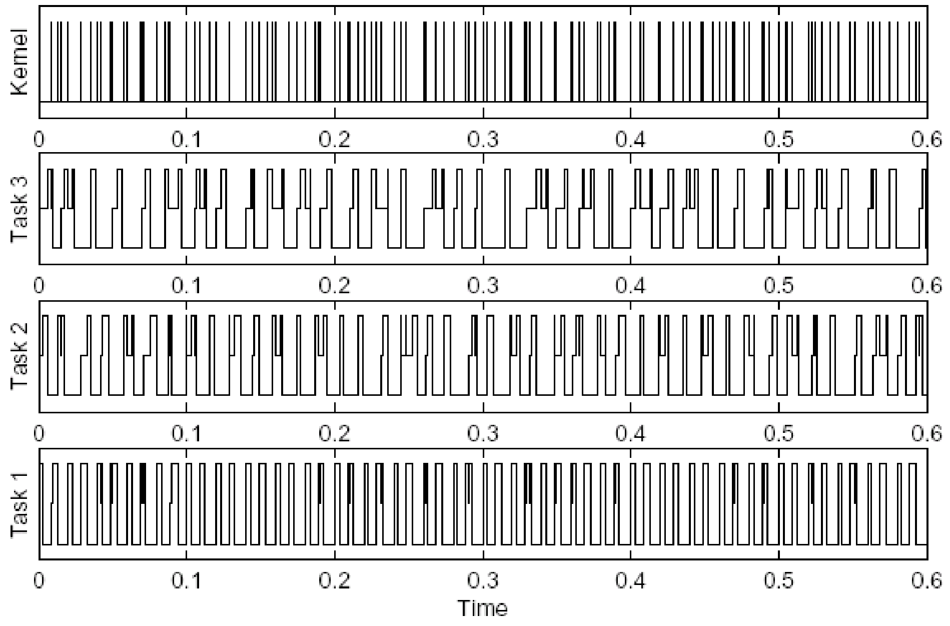


**Figure 1.14** TrueTime simulation of the inverted pendulum system under the Control Server model: (a) control performance, and (b) task schedule. The performance is very close to the ideal case (see Figure 1.3). The kernel handles the I/O operations of all the control tasks.