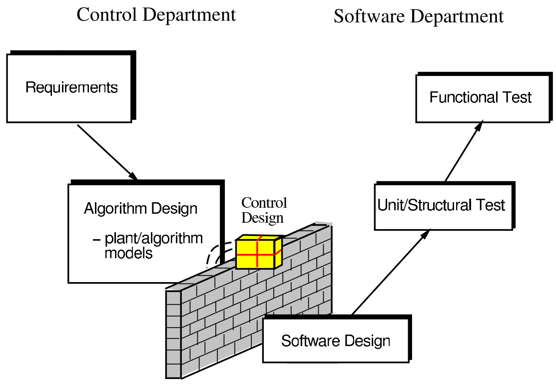
Lecture 15: Integrated Control and Scheduling

[These slides]

1. Introduction
2. Control task timing
3. Control analysis with delay and jitter
4. Control design to compensate for delay and jitter
5. Scheduling design to reduce delay and jitter
6. TrueTime: A MATLAB/Simulink-based simulator for real-time control systems

1. Introduction

Typical control system development today:



Problems

- The control engineer does not care about the implementation
  - "trivial"
  - "buy a fast computer"
- The software engineer does not understand controller timing
  - "$\tau_i = (T_i, D_i, C_i)$"
  - "hard deadlines"
- Control theory and real-time scheduling theory have evolved as separate subjects for more than thirty years

In the Beginning...

Liu and Layland (1973): "Scheduling algorithms for multiprogramming in a hard-real-time environment."

- Rate-monotonic (RM) scheduling
- Earliest-deadline-first (EDF) scheduling
- Actually motivated by process control
  - Samples "arrive" periodically
  - Control response must be computed before end of period
  - "Any control loops closed within the computer must be designed to allow at least an extra unit sample delay."

Common Assumptions about Control Tasks

In the simple task model, a task τ_i is described by

- a fixed period T_i
- a fixed, known worst-case execution time C_i
- a hard relative deadline $D_i = T_i$

Is this model suitable for control tasks?

Fixed Period?

Not necessarily:

- Some controllers are not sampled against time
  - Engine controllers
- Some controllers may switch between different modes with different sampling intervals
  - Hybrid controllers
- The sampling period could be on-line adjusted by a supervisory task ("feedback scheduling")

Fixed and Known WCET?

Not always:

- WCET analysis is a very hard problem
  - May have to use estimates or measurements
- Some controllers may switch between different modes with different execution times
  - Hybrid controllers
- Some controllers can explicitly trade off execution time for quality of control
  - “Any-time” optimization algorithms
  - Model-predictive controllers (MPC)
  - Long execution time ⇒ high quality of control

Hard Deadlines?

Most often not:

- Controller deadlines are often **firm** rather than hard
  - OK to miss a few outputs, but not too many in a row
  - Depends on what happens when a deadline is missed:
    - ∗ Task is allowed to complete late – often OK
    - ∗ Task is aborted at the deadline – worse
- At the same time, meeting all deadlines does not guarantee stability of the control loop
  - $D_i = T_i$ is motivated by runability conditions only

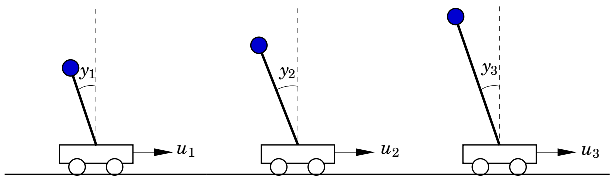
Inputs and Outputs?

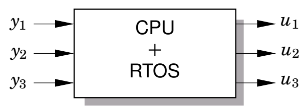
Completely missing from the simple task model:

- When are the inputs (measurement signals) read?
  - Beginning of period?
  - When the task starts?
- When are the outputs (control signals) written?
  - When the task finishes?
  - End of *Calculate Output*?
  - End of period?

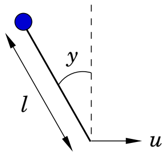
Inverted Pendulum Example

Control of three inverted pendulums using one CPU:





The Inverted Pendulum



A simple second-order model is given by

$$\frac{d^2y}{dt^2} = \omega_0^2 \sin y + u\, \omega_0^2 \cos y$$

where $\omega_0 = \sqrt{\frac{g}{l}}$ is the natural frequency of the pendulum.

Lengths $l = 1, 2, 3$ cm $\Rightarrow$ $\omega_0 = 31, 22, 18$ rad/s

Control Design

Linearization around the upright equilibrium gives the state-space model

$$\frac{dx}{dt} = \begin{pmatrix} 0 & 1 \\ \omega_0^2 & 0 \end{pmatrix} x + \begin{pmatrix} 0 \\ \omega_0^2 \end{pmatrix} u$$
$$y = \begin{pmatrix} 1 & 0 \end{pmatrix} x$$

Digital controller: state feedback from observer w. direct term

- State feedback poles specified in continuous time as

$$s^2 + 1.4\omega_c s + \omega_c^2 = 0$$

- Observer poles specified in continuous time as

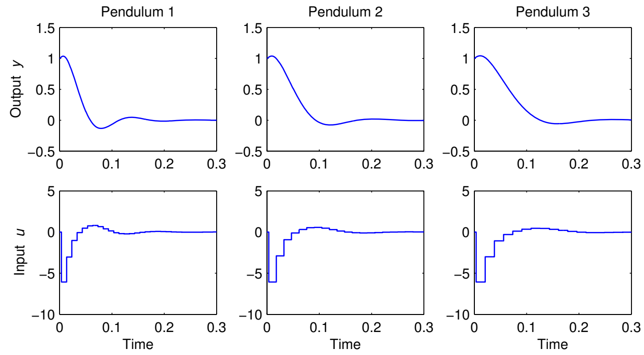
$$s^2 + 1.4\omega_o s + \omega_o^2 = 0$$

Control Design

- State feedback poles: $\omega_c = 53, 38, 31$ rad/s
- Observer poles: $\omega_o = 106, 75, 61$ rad/s
- Sampling intervals: $T = 10, 14.5, 17.5$ ms
- Sampling at the beginning of the period, actuation at the end of execution
- Assumed execution time: $C = 3.5$ ms

Simulation 1 – Ideal Case

Each control task runs on a separate CPU.



Schedulability Analysis

- Assume $D_i = T_i$
- Utilization $U = \sum_{i=1}^{3} \frac{C_i}{T_i} = 0.79$
- Schedulable under EDF?

$$U < 1 \quad \Rightarrow \quad \text{Yes}$$

- Schedulable under RM?

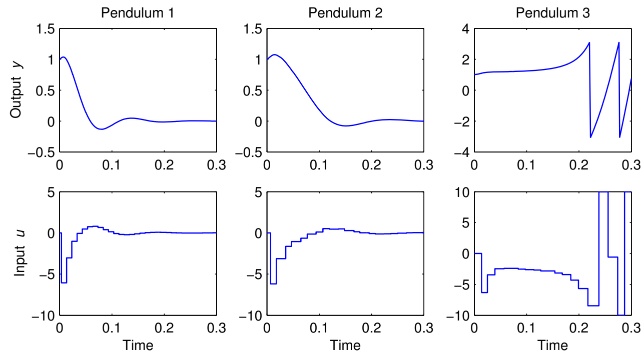
$$U > 3(2^{1/3} - 1) = 0.78 \quad \Rightarrow \quad \text{Cannot say}$$

Must compute worst-case response times R_i:

Task	T	D	C	R
1	10	10	3.5	3.5
2	14.5	14.5	3.5	7.0
3	17.5	17.5	3.5	14.0

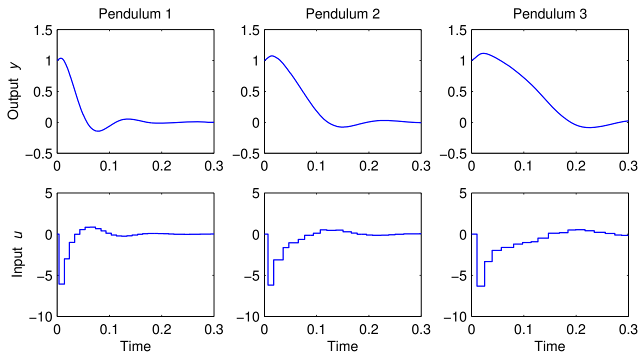
$$\forall i: \; R_i < D_i \; \Rightarrow \; \text{Yes}$$

Simulation 2 – Rate-Monotonic Scheduling



- Loop 3 becomes unstable

Simulation 3 – Earliest-Deadline-First Scheduling



- All loops are OK

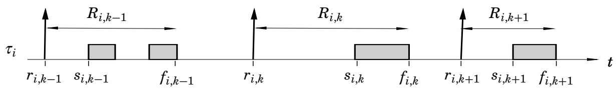
Questions

- How can a loop become unstable even though the system is schedulable?
- Why does EDF work better than RM in this example?

Need to study control loop timing

2. Control Task Timing

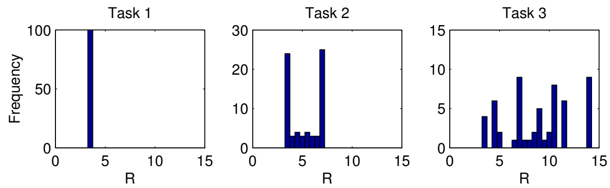
Periodic task executing in a multi-tasking system:



- $r_{i,k} = kT_i$ – release time of job k of task i
- $s_{i,k}$ – start time of job k of task i
- $f_{i,k}$ – finish time of job k of task i
- $R_{i,k}$ – response time of job k of task i
- $R_i = \max_k R_{i,k}$ – worst-case response time of task i

Response Times in the Pendulum Example

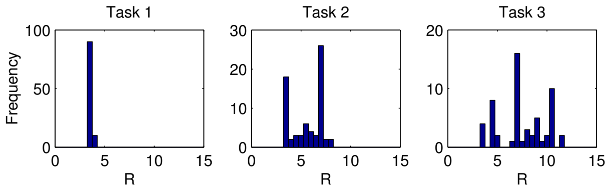
Histograms of measured response times during a 1-second simulation under rate-monotonic scheduling:



- The maximum values agree with the theoretical worst-case response times
- Under RM scheduling: low priority ⇒ large variability

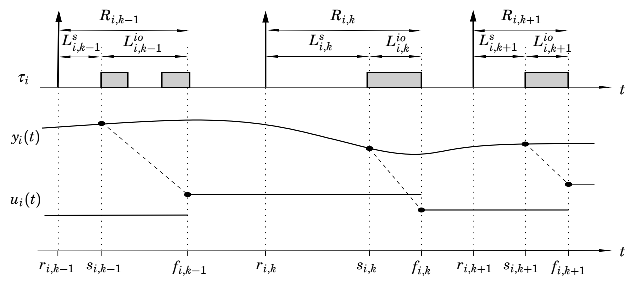
Response Times in the Pendulum Example

Histograms of measured response times under EDF scheduling:



- Smaller variability for Task 3 compared to RM scheduling

Latency and Jitter in Control Tasks



- $L^s_{i,k}$ – sampling latency of job k of task i
- $L^{io}_{i,k}$ – input-output latency of job k of task i
- $J^s_i = \max_k L^s_{i,k} - \min_k L^s_{i,k}$ – sampling jitter of task i
- $J^{io}_i = \max_k L^{io}_{i,k} - \min_k L^{io}_{i,k}$ – input-output jitter of task i

3. Control Analysis with Delay and Jitter

- Constant delay in linear systems – straightforward
- Sampling and input-output jitter – more difficult
  - Worst-case stability analysis
    - ∗ Only input-output jitter
    - ∗ Requires minimum and maximum values for the input-output latency
    - ∗ Stability theorem by Kao and Lincoln
  - Average-case, stochastic performance analysis
    - ∗ Requires a stochastic model of the latencies
    - ∗ Jitterbug toolbox
  - Simulation
    - ∗ TrueTime toolbox

Analysis of Constant Input-Output Delay

- Delay decreases the phase margin
- Definition: the **delay margin** L_m is the maximum constant delay the loop can tolerate before it goes unstable
- Continuous-time systems:

$$L_m = \varphi_m / \omega_c$$

  - This formula is only approximate for sampled control systems
  - For sampled control systems, we must compute a root locus with respect to L to find the exact value of L_m

Approximate and Exact Delay Margins in the Pendulum Example

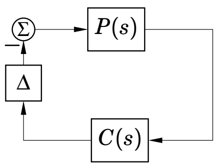
Controller	φ_m/ω_c (ms)	L_m (ms)
1	9.15	9.17
2	12.92	12.95
3	15.84	15.88

Limitations of Analysis using Delay Margin

- Only holds for linear systems
- Only holds for constant delays

Jitter Margin – Stability under Input-Output Jitter

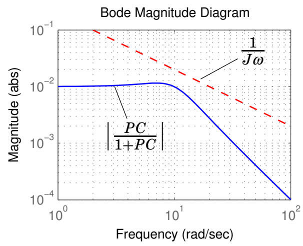
Stability theorem due to Kao and Lincoln (2004):



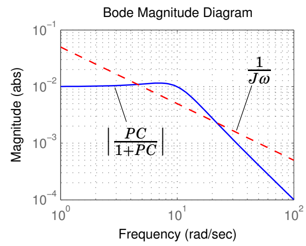
- Continuous-time plant $P(s)$
- Continuous-time controller $C(s)$
- Arbitrarily time-varying delay $\Delta \in [0,\, J]$
- Theorem: The closed-loop system is stable **if**

$$\left| \frac{P(i\omega)C(i\omega)}{1+P(i\omega)C(i\omega)} \right| < \frac{1}{J\omega} \quad \forall \omega \in [0, \infty].$$

Graphical test:



Stable

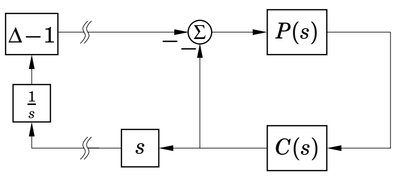


Could be unstable

(Note that the theorem gives a sufficient but not necessary condition for stability)

Proof Sketch

Uses nonlinear control theory. Rewrite the control output as one direct path and one error path:



Gain of left part: J

Gain of right part: $\max_{\omega} \left| \frac{i\omega\, P(i\omega)C(i\omega)}{1+P(i\omega)C(i\omega)} \right|$

The result follows from the Small Gain Theorem

Stability Under Jitter – Sampled Control Case

The sampled control case is more complicated.

Assume continuous-time plant $P(s)$, discrete-time controller $C(z)$ and input-output jitter $J \le h$.

The closed-loop system is stable if

$$\left| \frac{P_{\text{alias}}(\omega)C(e^{i\omega})}{1+P_{\text{ZOH}}(e^{i\omega})C(e^{i\omega})} \right| < \frac{1}{\sqrt{J}\left|e^{i\omega}-1\right|}, \quad \forall \omega \in [0, \pi]$$

where

- $P_{\text{alias}}(\omega) = \sqrt{\sum_{k=-\infty}^{\infty} \left| P\left(i(\omega + 2\pi k)\frac{1}{h}\right)\right|^2}$
- $P_{\text{ZOH}}(z)$ is the ZOH-discretization of $P(s)$

Jitter Margins in the Pendulum Example

Definition: the **jitter margin** J_m be the largest jitter such that the closed-loop system is still guaranteed to be stable.

Delay margins and jitter margins for the pendulums:

Controller	L_m (ms)	J_m (ms)
1	9.17	8.30
2	12.95	11.72
3	15.88	14.37

Limitations of Analysis using the Jitter Margin

- Only holds for linear systems
- Assumes zero sampling jitter
- Only uses knowledge of the minimum and maximum input-output latencies
- Does not exploit any statistical properties about the jitter

Jitterbug

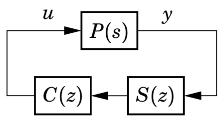
- Matlab toolbox for stochastic control analysis (Lincoln and Cervin, 2002)
- Random delays in the control loop described by probability distributions
- System disturbed by white noise
- Performance measured by quadratic cost function

$$V = \mathrm{E}\ x^T Q x$$

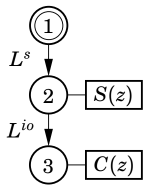
  - Small V ⇔ good performance
  - $V = \infty$ ⇔ unstable control loop

Jitterbug Model – Example

Signal model:



Timing model:



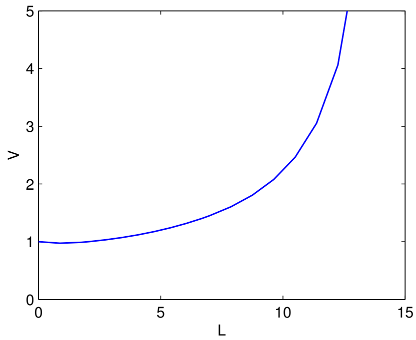
- $P(s)$ – process
- $S(z)$ – sampler, $C(z)$ – controller and actuator
- L_s, L_{io} – latency distributions (random variables)
- Cost function: $V = \mathrm{E}\ y^2$

Jitterbug – Example Script

```
dt = h/5;                  % time granularity
PLs = [0.2 0.2 0.6 0 0 0];  % distribution of Ls
PLio = [0.5 0 0 0 0 0.5];   % distribution of Lio
N = initjitterbug(dt,h);
N = addtimingnode(N,1,PLs,2);        % node 1
N = addtimingnode(N,2,PLio,3);       % node 2
N = addtimingnode(N,3);              % node 3
N = addcontsys(N,1,plant,3,Q,R1,R2); % plant
N = adddiscsys(N,2,1,1,2);           % sampler in node 2
N = adddiscsys(N,3,ctrl,2,3);        % controller in node 3
N = calcdynamics(N);
V = calccost(N)
```

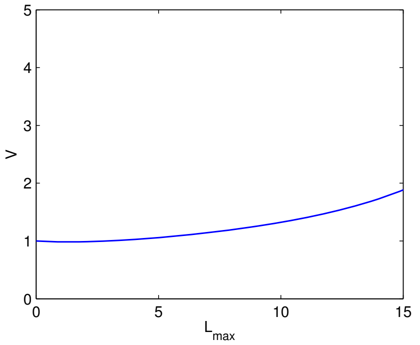

Jitterbug Example 1

Pendulum controller 3: Evaluate cost for different values of constant delay input-output delay L:



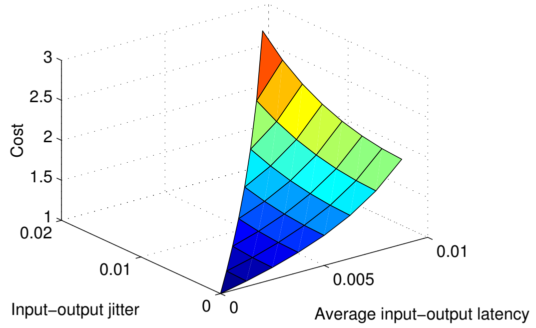
Jitterbug Example 2

Pendulum controller 3: Cost vs random delay $L \in U(0, L_{max})$:



Jitterbug Example 3

Pendulum controller 3: Cost vs average I-O delay and I-O jitter

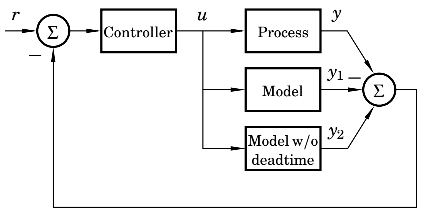


Limitations of Analysis using Jitterbug

- Only holds for linear systems
- Very simplistic stochastic model with independent random delays
- Calculates the average-case performance

4. Control Design to Compensate for Delay and Jitter

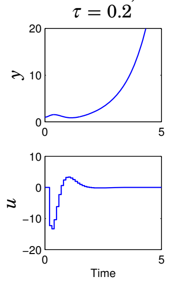
Classical Smith predictor for delay compensation:



Problem: Only works if the process model is stable

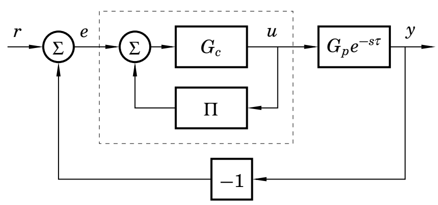
Example: Pendulum Controller with Smith Predictor

$h = 0.1$,
$\tau = 0.2$



- The controller thinks that it is doing the right thing
- Based on feedforward rather than feedback

Delay Compensation – More General



Design procedure:

- Design controller G_c for delay-free process G_p
- Add compensator

$$\Pi(s) = \tilde{G}_p(s) - G_p(s)e^{-s\tau}$$

to cope with deadtime process $G_p e^{-s\tau}$

Many Different Compensators Have Been Proposed

- Smith predictor:

$$\tilde{G}_p(s) = G_p(s)$$

- Watanabe–Ito predictor:

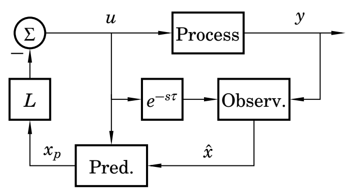
$$\tilde{G}_p(s) = Ce^{A\tau}(sI - A)^{-1}B - C\int_0^\tau e^{-As}ds B$$

  - Guarantees that the controller retains integral action
- ...

Observer–Predictor

In a state feedback–observer structure, delay control signal to the observer and compute the feedback from a predicted state:

$$\frac{d\hat{x}(t)}{dt} = A\hat{x}(t) + Bu(t-\tau) + K\Big(y(t) - C\hat{x}(t)\Big)$$
$$x_p(t) = e^{A\tau}\hat{x}(t) + \int_{t-\tau}^{t} e^{A(t-s)}Bu(s)ds$$
$$u(t) = -Lx_p(t)$$



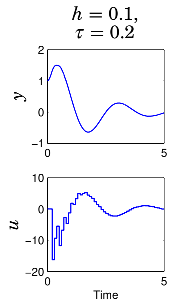
Better – Digital Design

- Include the delay in the process description
- Sample the process with the delay
- Design a controller for the sampled system
  - A simple option is to place the extra poles in the origin
    - Corresponds to the observer–predictor
    - Might be too aggressive
  - Try to respect the rule of thumb

$$\omega(h + 2\tau) = 0.1 \text{ to } 0.5$$

  where ω is the bandwidth of the closed-loop system

Pendulum Controller with Delay Compensation using Digital Design



- Shaky response, but stable
- $\omega(h + 2\tau) = 1.4$

Coping with Jitter

Three approaches

- Ignore the jitter
- Design a robust controller
- Design a controller that actively compensates for the jitter in each sample
  - Requires that the latencies are measured
  - Problem: the input-output latency in the current sample is not known when the control signal is computed

Coping with Input-Output Jitter

Sampled model with varying delay τ_k:

$$x(k+1) = \Phi x(k) + \Gamma_0(\tau_k)u(k) + \Gamma_1(\tau_k)u(k-1)$$

- Design the feedback

$$u(k) = -L\begin{pmatrix}\hat{x}(k)\\ u(k-1)\end{pmatrix}$$

  based on the average (expected) input-output delay
- Modify the observer to take into account current delay τ_k:

$$\hat{x}(k+1) = \Phi\hat{x}(k) + \Gamma_0(\tau_k)u(k) + \Gamma_1(\tau_k)u(k-1) + K(y(k) - C\hat{x}(k))$$

Similar techniques can be used for sampling jitter

5. Scheduling Design to Reduce Delay and Jitter

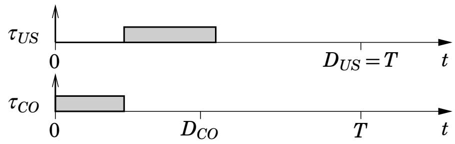
A control algorithm normally consists of two parts:

```
while (1) {
  read_input();
  calculate_output();
  write_output();
  update_state();
  ...
}
```

Idea: schedule the two parts as separate tasks

- input, calculate, output – high priority
- update – low priority

Subtask Scheduling Analysis



- Calculate Output (τ_{CO}) should have as short deadline as possible
- Update State (τ_{US}) can have deadline $D_{US} = T$.

A Deadline Assignment Algorithm

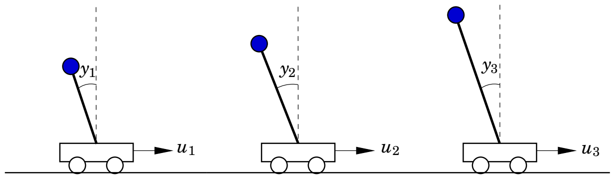
Assume we have a number of control tasks that can be divided into Calculate Output and Update State.

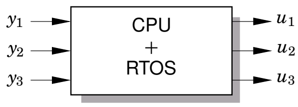
1. Start by assigning initial deadlines
   - $D_{CO} := T - C_{US}$
   - $D_{US} := T$

   for all tasks.
2. Assign deadline-monotonic priorities to all subtasks
3. Calculate the response time R of each subtask
4. Assign $D_{CO} := R_{CO}$ for all tasks
5. Repeat from 2 until no further improvement.

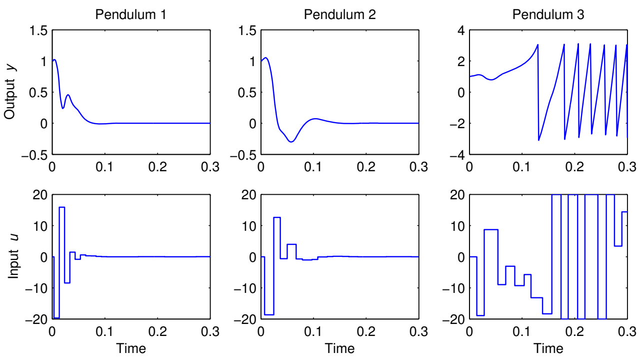
Inverted Pendulum Example (Again)

Control of three inverted pendulums using one CPU:

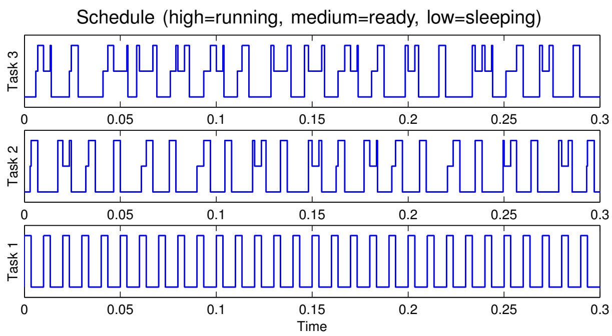




Simulation under RM scheduling



Simulation under RM scheduling



- Large delay and jitter for controller 3

Subtask Scheduling Analysis

Each pendulum controller is divided into two subtasks:

- Calculate output: $C_{CO} = 1.5$ ms
- Update state: $C_{US} = 2.0$ ms

First iteration of algorithm:

	T	D	C	R
τ_{CO1}	10.0	8.0	1.5	1.5
τ_{US1}	10.0	10.0	2.0	3.5
τ_{CO2}	14.5	12.5	1.5	5.0
τ_{US2}	14.5	14.5	2.0	7.0
τ_{CO3}	17.5	15.5	1.5	8.5
τ_{US3}	17.5	17.5	2.0	14.0

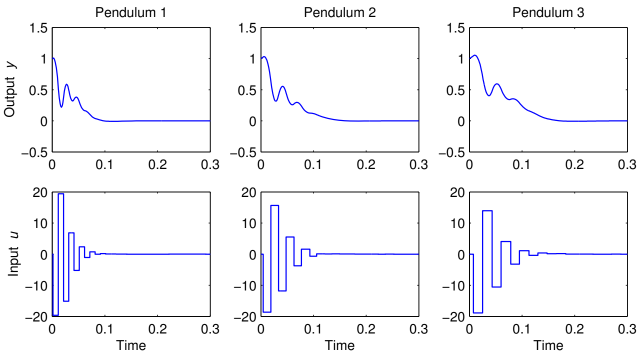
Subtask Scheduling Analysis

Third iteration (converged):

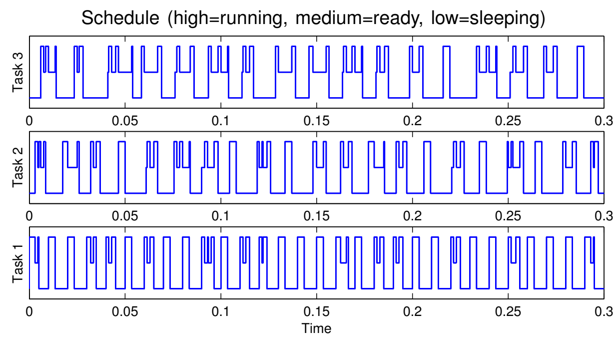
	T	D	C	R
τ_{CO1}	10.0	1.5	1.5	1.5
τ_{US1}	10.0	10.0	2.0	6.5
τ_{CO2}	14.5	3.0	1.5	3.0
τ_{US2}	14.5	14.5	2.0	8.5
τ_{CO3}	17.5	4.5	1.5	4.5
τ_{US3}	17.5	17.5	2.0	14.0

New worst-case input-output latencies: 1.5, 3.0, 4.5 ms.

Simulation under Subtask Scheduling



Simulation under Subtask Scheduling



- More context switches

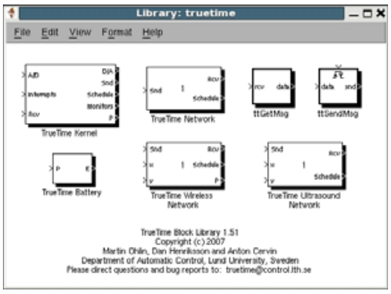
6. The TrueTime Simulator

- MATLAB/Simulink toolbox by Henriksson, Cervin, Ohlin, Eker (1999–2008)
- TrueTime supports co-simulation of control task execution, network communication, and plant dynamics
  - Simulink blocks model real-time kernels and communication networks
  - The kernels execute user code (tasks and interrupt handlers) written in C++ or MATLAB code
  - The simulated application is programmed in much the same way as a real application

Why Co-Simulation?

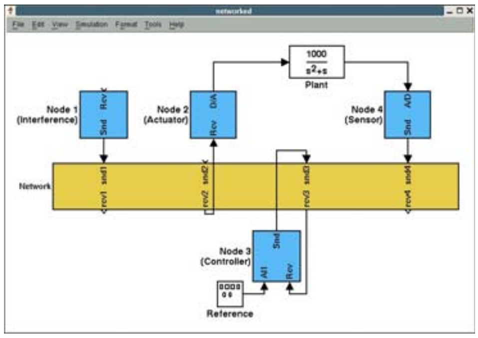
- Networked embedded systems are very complex systems
- Nonlinear system dynamics
- Temporal nondeterminism
  - preemption by higher-priority tasks, blocking, varying computation times, kernel overhead, . . .
  - network interface delays, queuing delays, transmission and retransmission delays, lost packets, . . .

The TrueTime Block Library

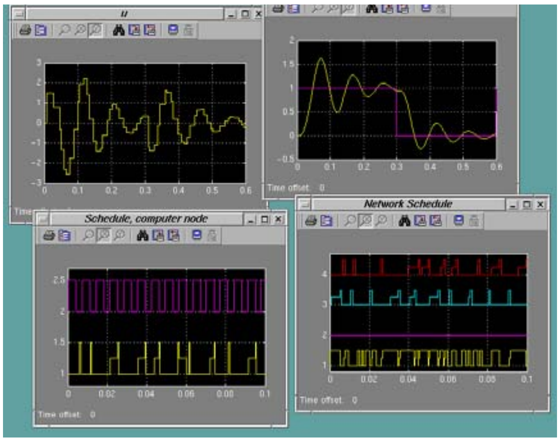


- A Kernel block, three Network blocks, and a Battery block
  - Simulink S-functions written in C++
  - Event-based execution using zero-crossing functions
  - Portable to other simulation environments

Example – Networked Control Loop

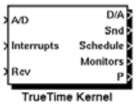


Example – Networked Control Loop



The Kernel Block

- Simulates a generic real-time kernel with A/D-D/A and network interfaces
- Executes user-defined tasks and interrupt handlers
- Supports various scheduling policies
- Supports all common real-time primitives (timers, monitors, semaphores, mailboxes, dynamic task attributes, . . .)
- More features: context switch overheads, overrun handlers, data logging, . . .



Example of Kernel Initialization Script

```
nbrInputs = 3;
nbrOutputs = 3;
ttInitKernel(nbrInputs, nbrOutputs, 'prioFP');
periods = [0.01 0.02 0.04];
code = 'my_ctrl';
for k = 1:3
   data.u = 0;
   taskname = ['Task ' num2str(k)];
   offset = 0;
   period = periods(k);
   prio = k;
   ttCreatePeriodicTask(taskname,offset,period,prio,code,d
ata);
end
```


Code Functions

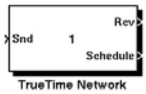
- Each task or interrupt handler in the user application must be implemented in a code function
- The code function is called repeatedly by the kernel during the simulation
  - The simulated execution time is returned by the function
- Three options for the implementation:
  - C++ code (fast)
  - MATLAB code (medium)
  - Simulink block diagram (slow)

Example of a MATLAB Code Function

```
function [exectime,data] = my_ctrl(segment,data)
switch segment,
  case 1,
    data.y = ttAnalogIn(1);
    data.u = calculate_output(data.x,data.y);
    exectime = 0.002;
  case 2,
    ttAnalogOut(1,data.u);
    data.x = update_state(data.x,data.y);
    exectime = 0.004;
   case 3,
    exectime = -1;
end
```

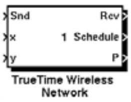

The Wired Network Block

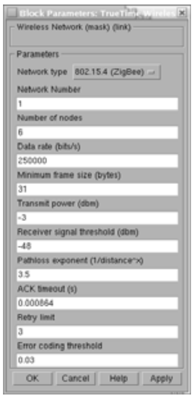
- Supports six common MAC layer policies:
  - CSMA/CD (Ethernet)
  - CSMA/AMP (CAN)
  - Round Robin (Token bus)
  - FDMA
  - TDMA
  - Switched Ethernet
- Policy-dependent network parameters
- Generates a transmission schedule



The Wireless Network Block

- Used in basically the same way as the wired network block
- Supports two common MAC layer policies:
  - 802.11b/g (WLAN)
  - 802.15.4 (ZigBee)
- Variable network parameters
- x and y inputs for node locations
- Generates a transmission schedule





TrueTime Demo: Robot Soccer

- $5+5$ mobile robots communicating over a wireless network