TrueTime: Simulation of Networked and Embedded Control Systems

Anton Cervin

Department of Automatic Control
Lund University
Sweden

Contributions from Dan Henriksson, Martin Ohlin, and Karl-Erik Årzén

Outline of Lecture

1. Simulation of networked control systems
2. A large-scale simulation example
3. TrueTime tutorial, part I

Why Simulation?

- Simulation is a crucial tool in control system development
  - Analysis
  - Design
  - Verification
- Networked embedded control systems are very complex systems
- An purely analytical approach is often not possible

Dynamics in networked embedded systems

Networked embedded control systems are **hybrid** dynamical systems:

- Continuous-time plant dynamics
- Discrete-time controller dynamics
- Discrete events, e.g.
  - Sampling a measurement signal
  - Starting a computation task
  - Sending a data packet
  - Detecting a network collision
  - Updating a zero-order hold circuit

Control System Simulators

Examples of tools:

- MATLAB/Simulink (Mathworks)
- Scilab/Scicos (originally from INRIA)
- Modelica-based simulators, e.g. Dymola (Dynasim)

Typical features:

- Focus on simulation of continuous dynamics
- Support for general user-defined discrete events
  - Zero-crossing functions
  - `if/then/else` constructs or `when` clauses

Network Simulators

Examples of tools:

- ns-2/ns-3 (originally from UC Berkeley)
- OPNET Modeler (OPNET, originally from MIT)

Typical features:

- Discrete event simulation
- Focus on wired/wireless networks, devices, protocols
- Some support for continuous dynamics, e.g. node mobility models

Hybrid System Simulators

Examples of tools:

- MATLAB/Simulink + Stateflow / SimEvents (Mathworks)
- Ptolemy II (UC Berkeley)
- Chi (TU Eindhoven)

Typical features:

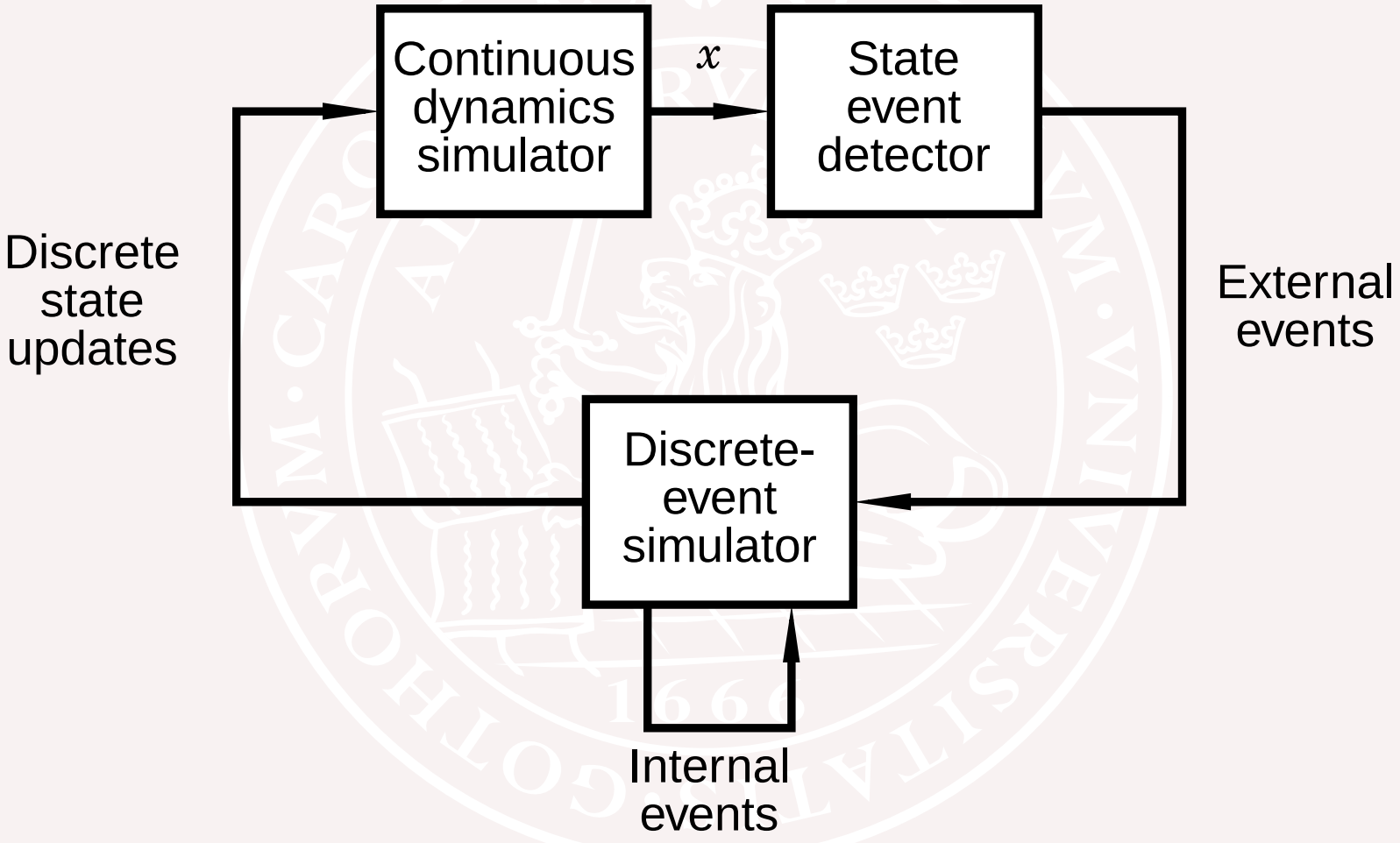
- High level of abstraction
- Large modeling effort to simulate NCS

Control–Network Co-Simulators

Examples of tools:

- RTNS (SSSA, Pisa)
  - Octave-based continuous dynamics simulation from within ns-2
- PiccSIM (Helsinki University of Technology)
  - Co-simulation of Simulink and ns-2
- TrueTime (Lund University)
  - custom-built discrete-event simulators inside MATLAB/Simulink

Co-Simulation of Hybrid Systems



Discrete-Event Simulator

```
LOOP
  Process all events at current time;
  Compute time of next internal event;
  Advance time to next internal or external event;
END
```

Continuous Dynamics Simulator

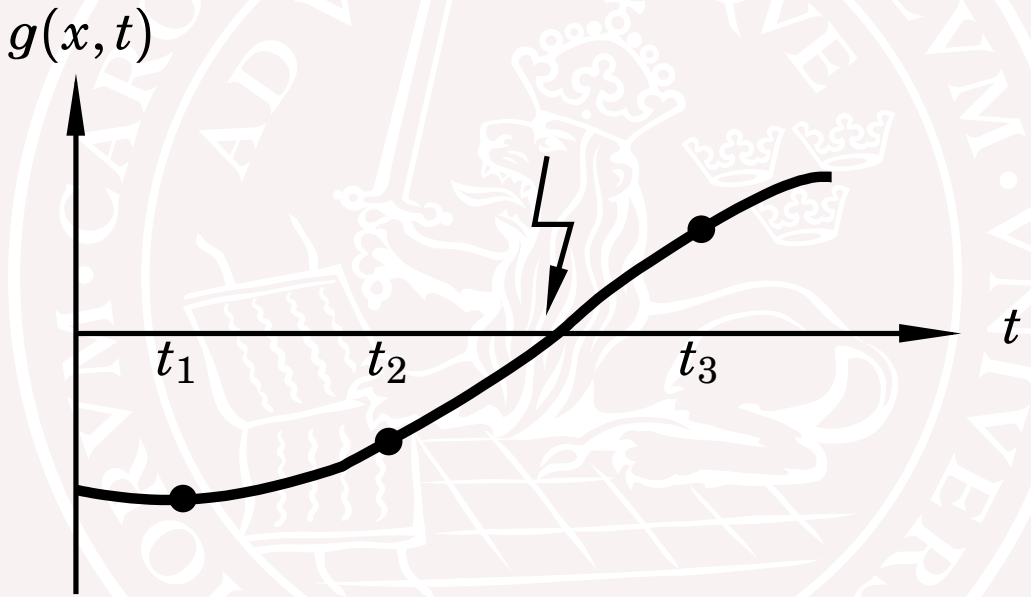
```
LOOP
  Determine suitable stepsize;
  Integrate until now+stepsize or until update event;
  IF state event detected
    Locate event and reverse time to event;
  END
END
```

- Variable-step ODE solver (integrator)
- External event detection implemented using zero-crossing functions

Zero-Crossing Functions

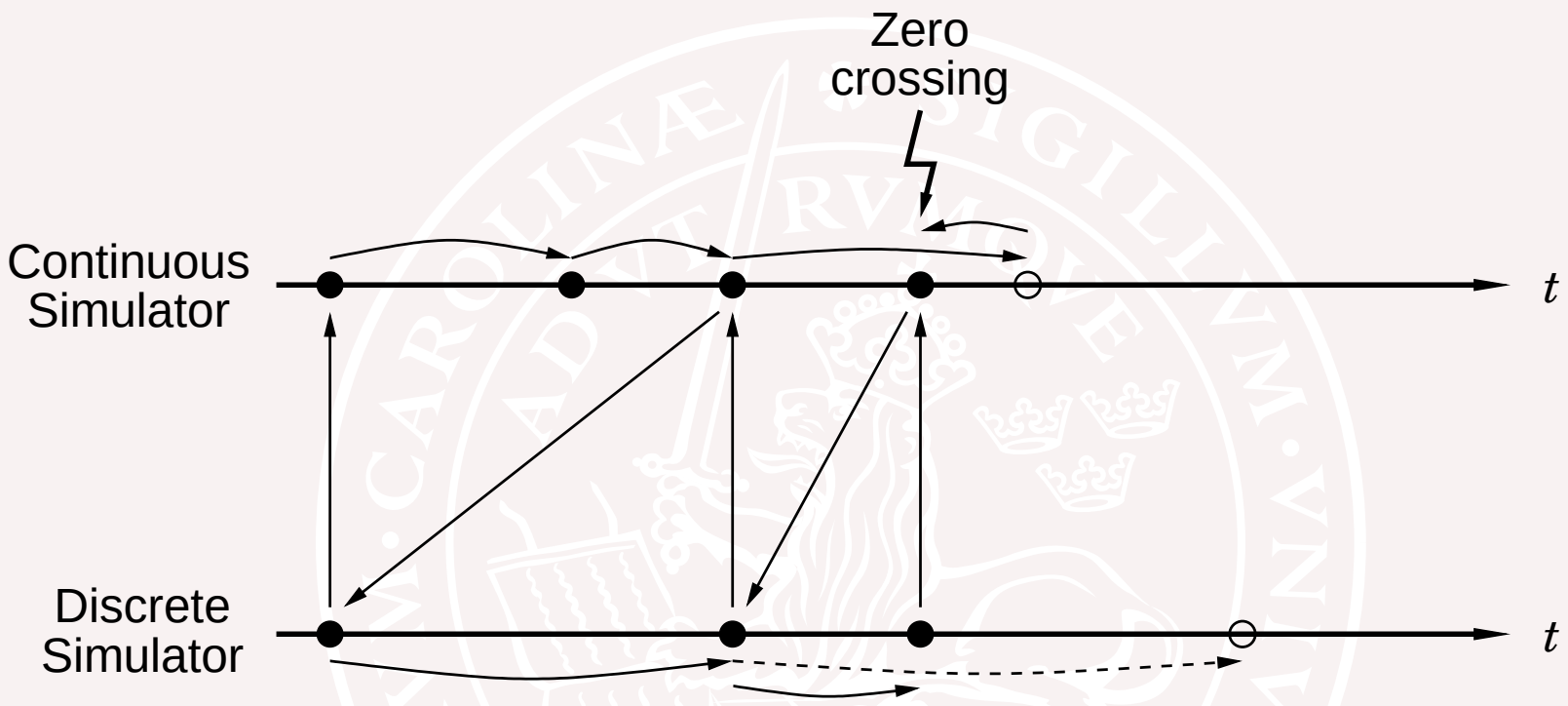
State events defined by e.g.

$$g(x,t) = 0$$



- After detection, the simulator evaluates $g(x,t)$ at several different points to locate the event with some precision

Synchronization of Simulators



- DS calculates next internal event t_i, yields to CS
- CS tries to integrate up to t_i, reports actual stop time t_s to DS
- DS advances time to t_s

Outline of Lecture

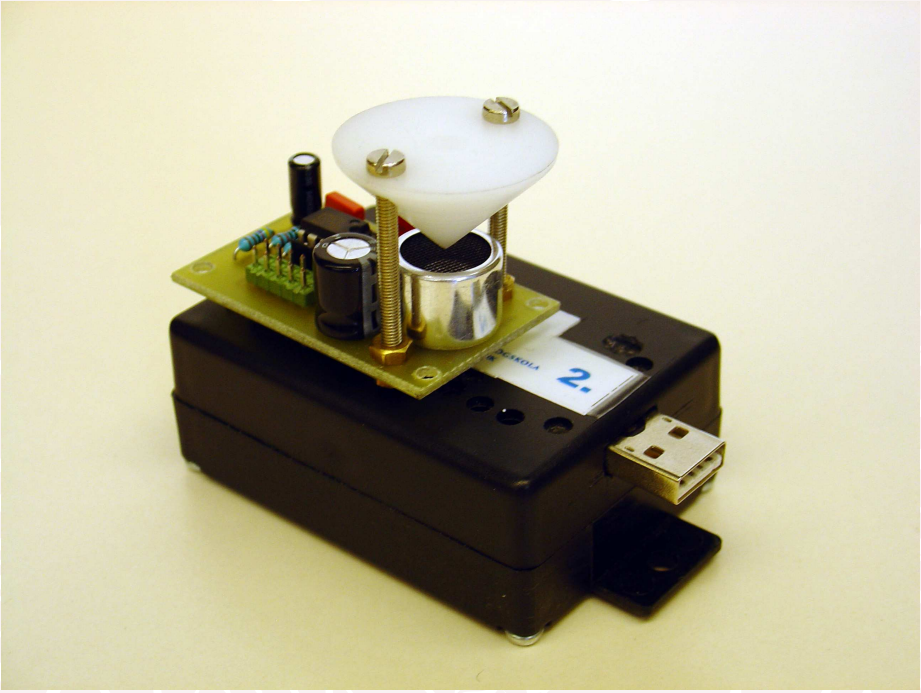
1. Simulation of networked control systems
2. A large-scale simulation example
3. TrueTime tutorial, part I

A Large-Scale Simulation Example

Sensor network & mobile ad-hoc network simulation

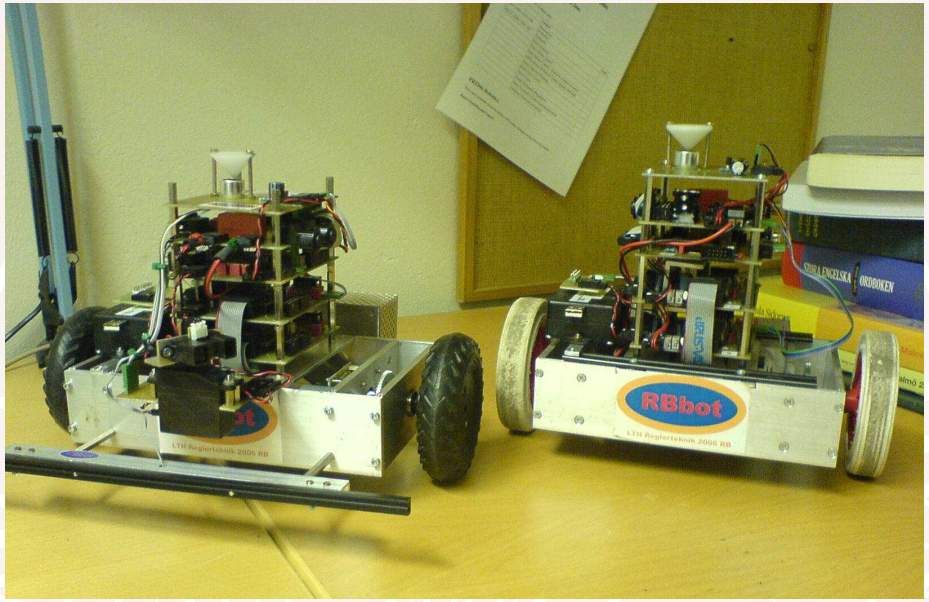
- Tunnel road safety scenario in EU/FP6 project RUNES
- Stationary sensor network in a road tunnel
- Mobile robots as mobile gateways for restoring connectivity among isolated subislands of network
- Mobile robot tasks:
  - Localization, navigation, collision and obstacle avoidance, power control
- TrueTime used for developing a simulation demo in parallel with the real physical demo (@ Ericsson, Kista, July 2007)

Stationary Sensor Nodes



- T-Mote Sky with wireless communication (IEEE 802.15.4)
- Ultrasound receiver
- AODV (Ad-hoc On-Demand Distance Vector) routing

Mobile Robots



- T-Mote Sky for wireless communication
- Ultrasound transmitter – ATmega16 AVR
- Compute engine – ATmega128 AVR
  - Extended Kalman Filter
- 2 wheel controllers – ATmega8 AVR
- Communication over I2C bus

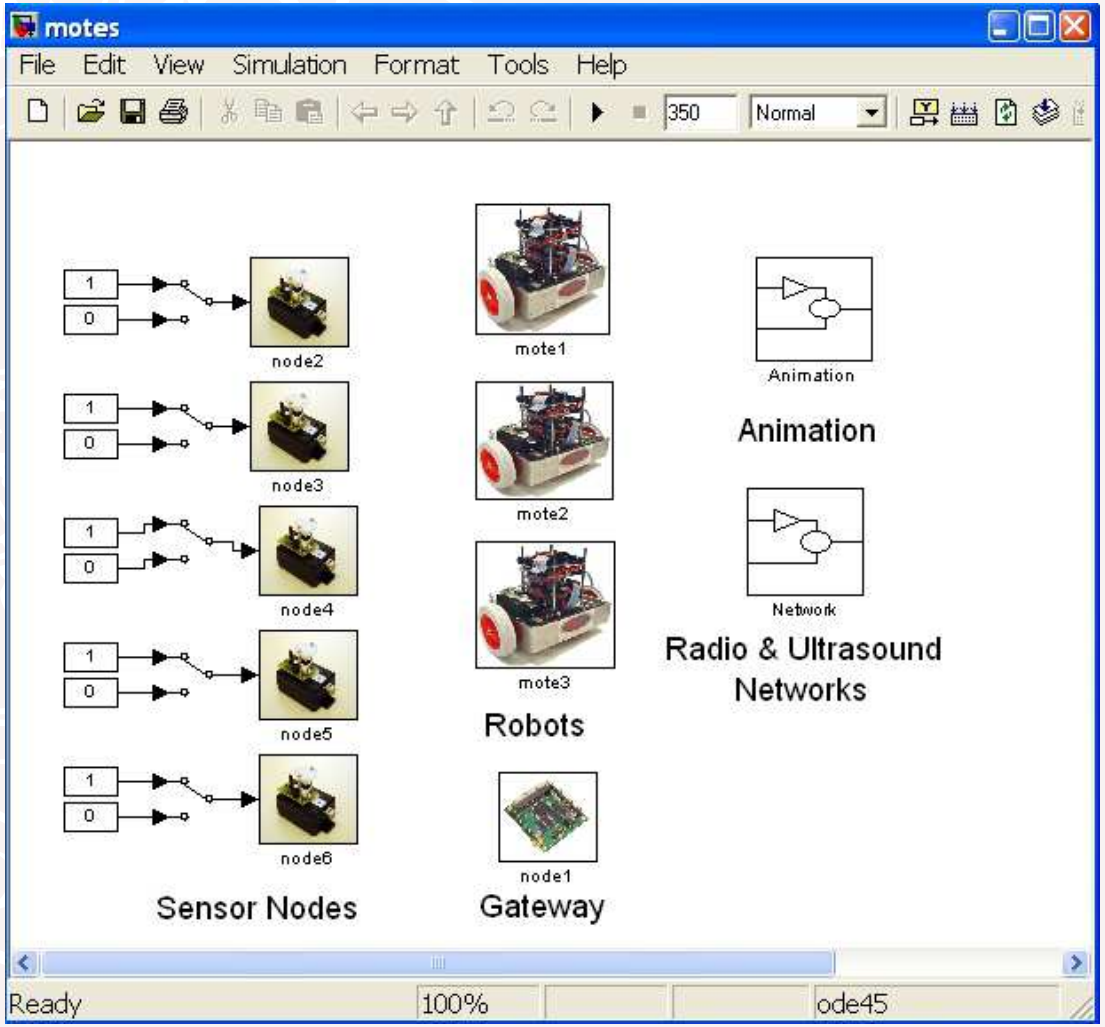
Localization Overview

- Ultrasound-based trilateration
- Robot broadcasts radio packet and ultrasound pulse "simultaneously"
- Difference in time-of-arrival allows each reachable node to calculate its distance to the robot
- After a predefined interval each node sends its distance measurement back to the robot
- Extended Kalman Filter (EKF)
  - Distance measurements used in the update part
  - Measured wheel positions (dead reckoning) used in the predictor part
  - Estimates of the x & y positions, orientation

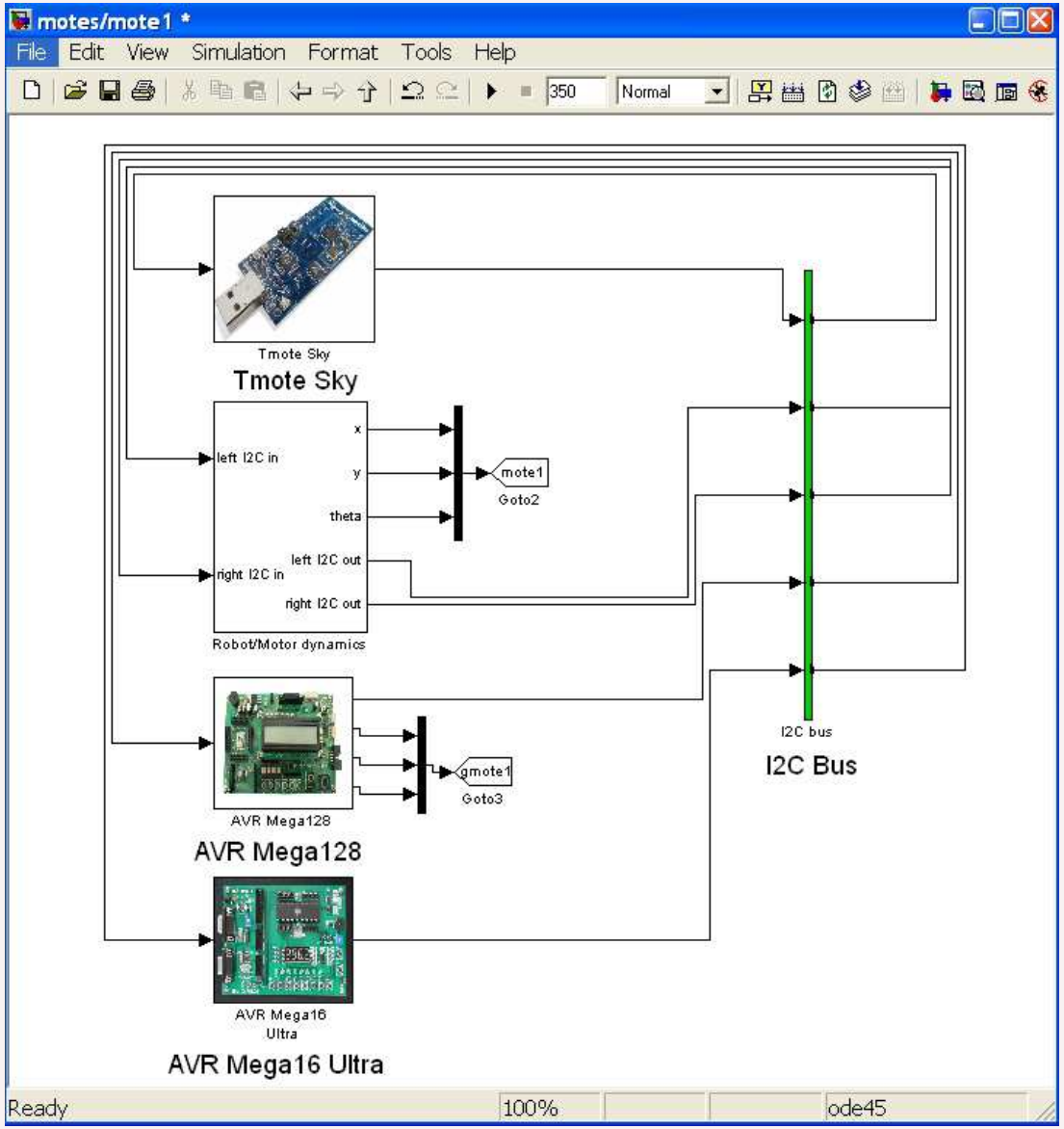
Movie from the final project review

TrueTime Simulation Model

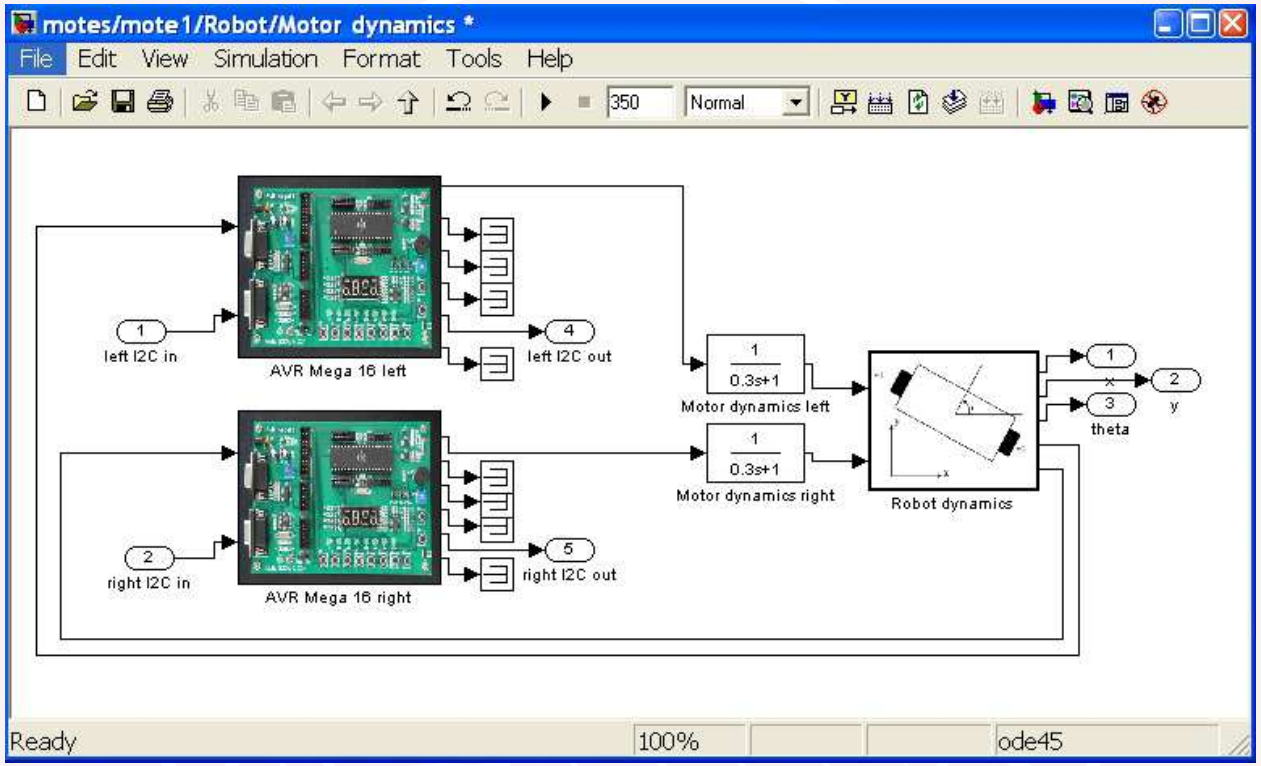
- Six sensor nodes
  - one gateway
  - turned on/off
- Up to three robots
- Radio & Ultrasound networks
- Animation



Robot Submodel

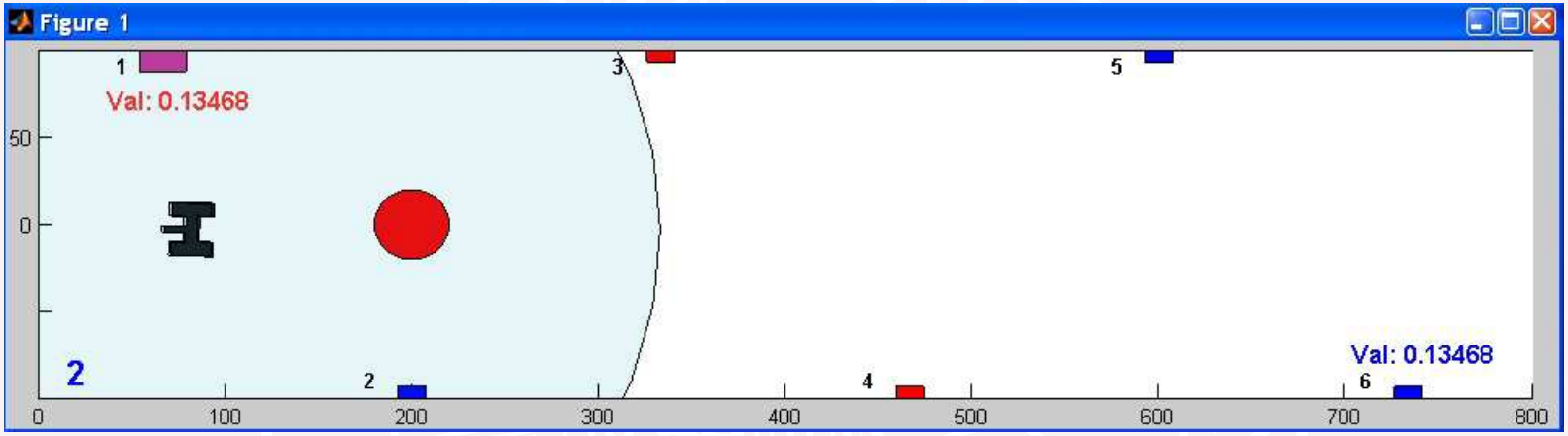


Wheel and Motor Submodels



- Simple motor models
- Dual-drive unicycle robot dynamics model

Animation



- Both the true position of the robots and their internal position estimate are shown
- A sensor node that is turned off (red) will not participate in the routing and localization

Outline of Lecture

1. Simulation of networked control systems
2. A large-scale simulation example
3. TrueTime tutorial, part I

TrueTime Overview

Co-simulation of control task execution,
network communication, and plant dynamics.

- Simulink blocks that 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
- Freeware: `http://www.control.lth.se/truetime`

TrueTime Possibilities

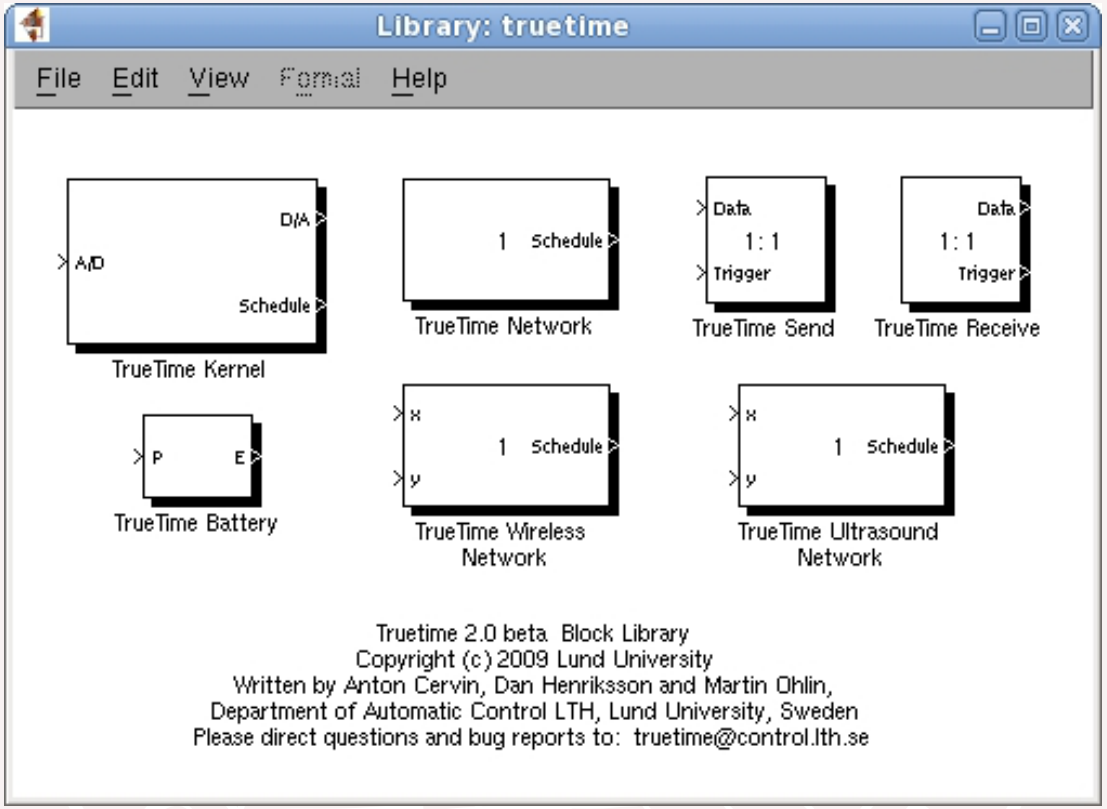
- Investigate the true, timely behaviour of time- or event-triggered control loops, subject to delays, jitter, and lost samples, caused by real-time scheduling and networking
- Experiment with novel real-time scheduling techniques, including feedback scheduling
- Investigate the performance of various wired or wireless MAC protocols
- Simulate complex scenarios involving battery-powered mobile robots communicating over a wireless network

A Very Brief History

- 1999 – first prototype without networking
- 2002 – version 1.0
- 2005 – version 1.3
  - Wireless networks
  - Dynamic voltage scaling
  - Local clocks
- 2006 – version 1.4
  - User-defined wireless pathloss function
  - AODV routing
- 2007 – version 1.5
  - Stand-alone network interface blocks
- 2009 – version 2.0 (beta)

Contributors: A. Cervin, J. Eker, D. Henriksson, M. Ohlin, K.-E. Årzén

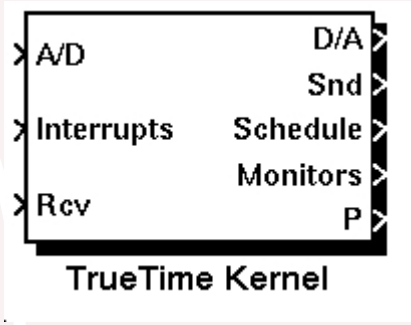
TrueTime 2.0 Block Library



- 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

The Kernel Block

- Simulates an event-based real-time kernel with A/D-D/A and network interfaces
- Executes user-defined tasks and interrupt handlers in response to timers and external events (network and hardware interrupts)
- Supports various scheduling policies
- Supports task communication/ synchronization using monitors, events, semaphores and mailboxes
- More features: context switch overheads, overrun handlers, data logging, . . .



TrueTime Code

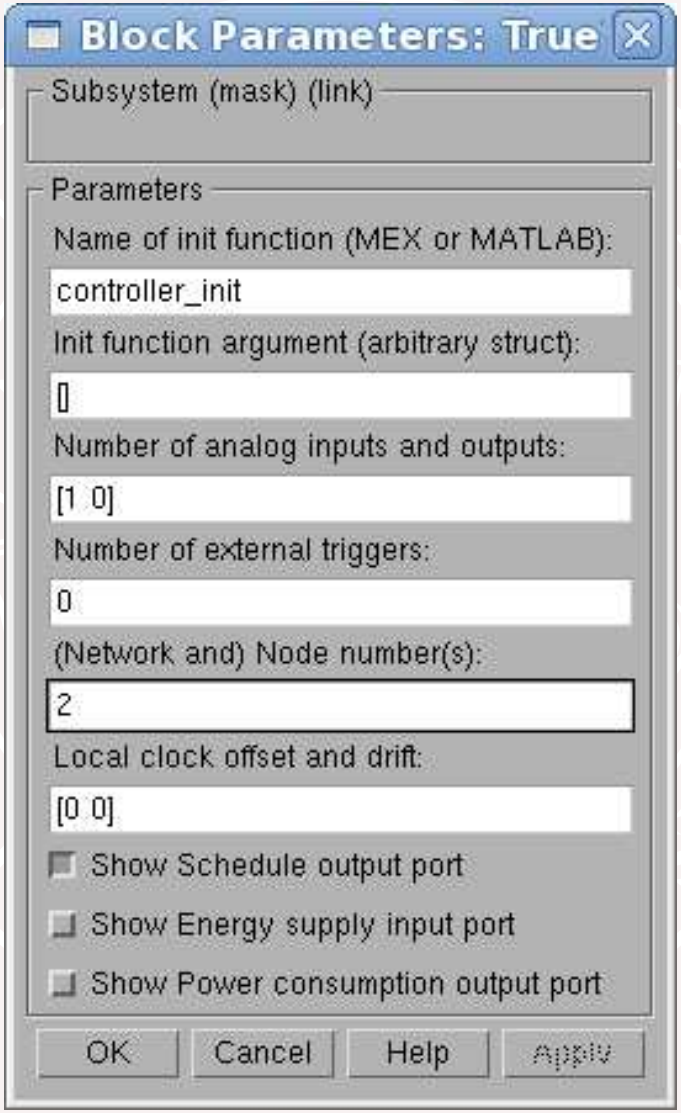
Three choices:

- C++ code (fast)
- MATLAB code (medium)
- Simulink block diagram (slow)

Kernel Implementation Details

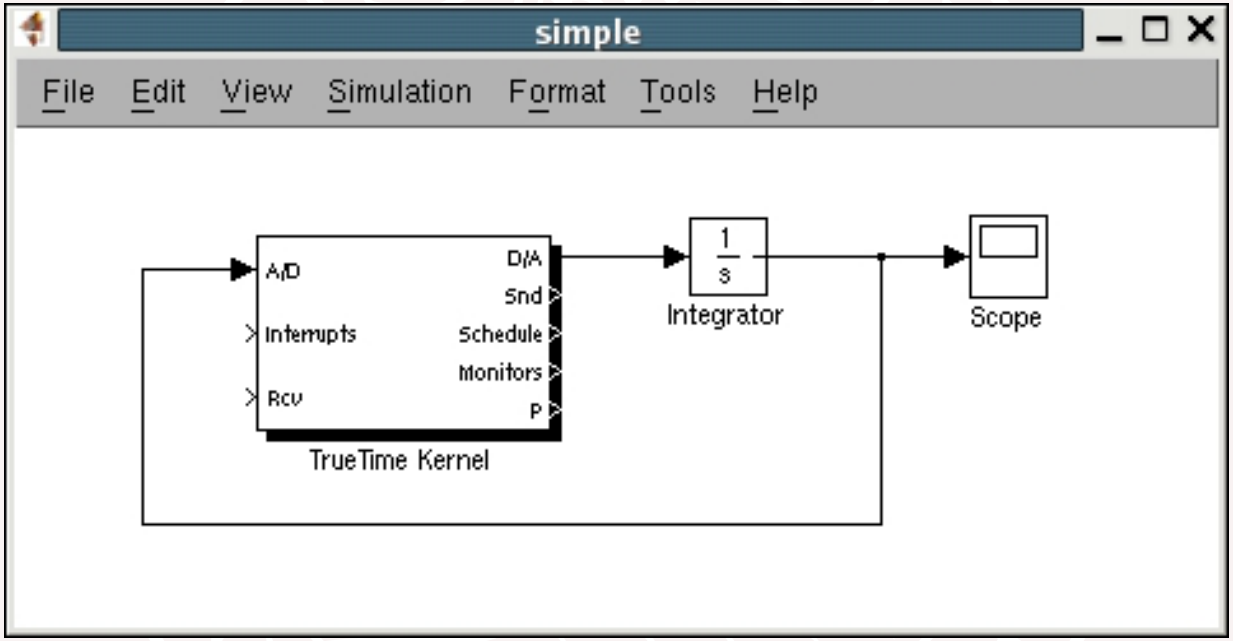
- TrueTime implements a complete real-time kernel with
  - A ready queue for tasks ready to execute
  - A time queue for tasks waiting to be released
  - Waiting queues for monitors, events, mailboxes and semaphores
- Queues are manipulated by the kernel or by calls to kernel primitives
- The simulated kernel is ideal
  - no interrupt latency and no execution time associated with real-time primitives
  - Possible to specify a constant context switch overhead

Configuration of Kernel Block



A Very Simple Example

P-control of an integrator:



- Initialization
- Task code

A Very Simple Example

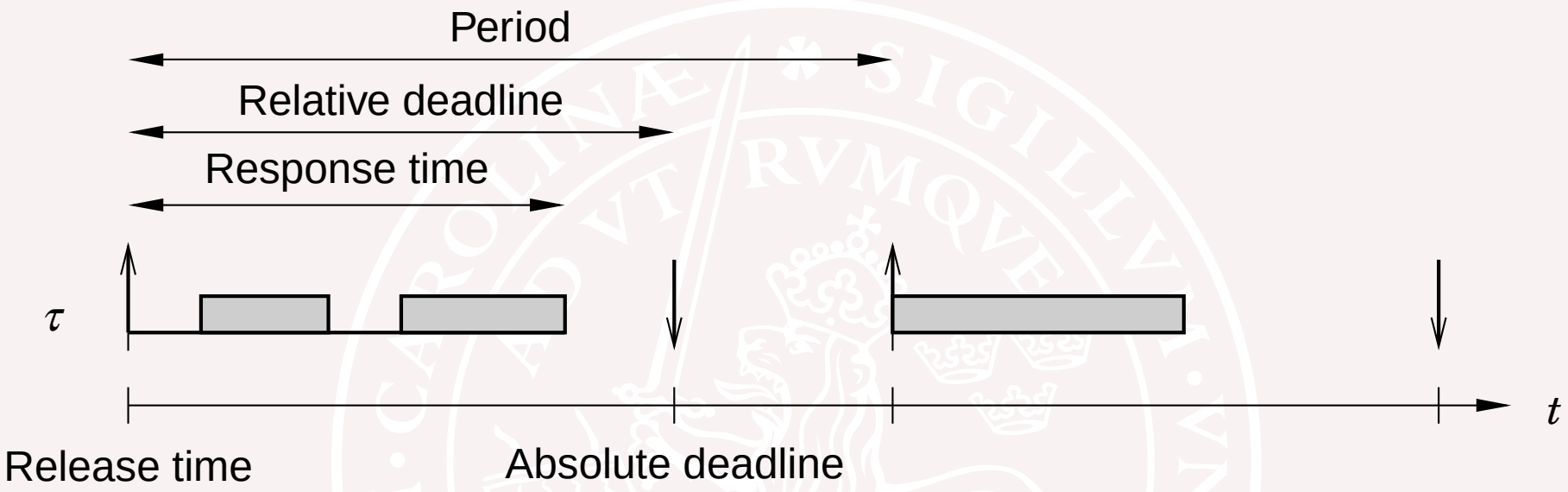
```
function verysimple_init
ttInitKernel('prioFP')
ttCreatePeriodicTask('task1', 0, 0.010, 'code', [])


function [exectime,data] = code(seg,data)
switch seg,
 case 1,
  y = ttAnalogIn(1);
  data.u = -0.5*y;
  exectime = 0.005;
 case 2,
  ttAnalogOut(1,data.u);
  exectime = -1;
end
```

Tasks

- Tasks are used to model the execution of user code
- The release of task instances (*jobs*) may be periodic or aperiodic
- For periodic tasks, the jobs are created by an internal periodic timer
- For aperiodic tasks, the jobs must be created by the user (e.g., in response to interrupts)
- In the case of multiple jobs of the same task, pending jobs are queued

```
ttCreatePeriodicTask(name, starttime, period, codeFcn, data)
ttCreateTask(name, deadline, codeFcn, data)
ttCreateJob(taskname, time)
ttKillJob(taskname)
```

Real-Time Terminology



- Each job has an execution-time budget that is set equal to the declared WCET of the task when the job is released and then decreases at unit rate
- By default, the relative deadline and the WCET are equal to the period

Task Attributes

- Dynamic attributes are updated by the kernel as the simulation progresses
  - Release time, absolute deadline, execution time, . . .
- Static attributes are kept constant unless explicitly changed by the user
  - Period, priority, relative deadline, . . .

```
ttSetAbsDeadline(taskname, value)
ttSetPeriod(taskname, value)
...
ttGetAbsDeadline(taskname)
ttGetPeriod(taskname)
...
```

Task Code

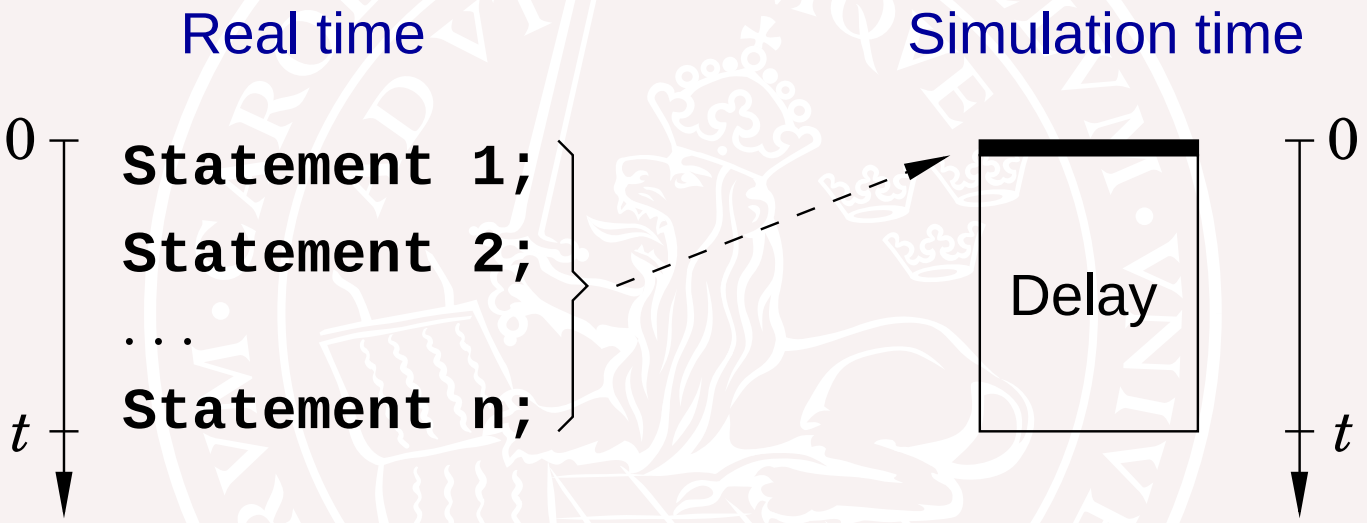
Task code is represented by a *code function* in the format

```
[exectime,data] = function mycode(segment,data)
```

- The `data` input/output argument represents the local memory of the task
- The `segment` input argument represents the program counter
- The `exectime` output argument represents the execution time of the current code segment

Code Segments

A code segment models a number of statements that are executed sequentially



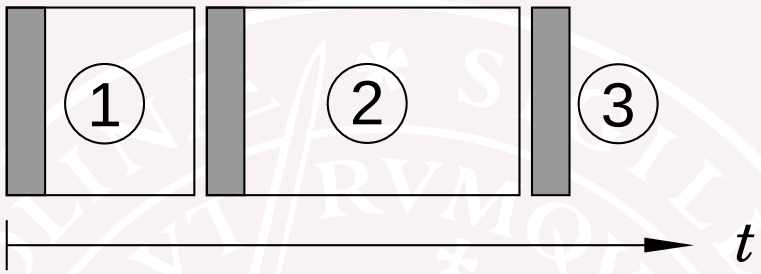
- The execution time t must be supplied by the programmer
  - Can be constant, random, or data-dependent
  - A return value of -1 means that the job has finished

Code Segments, cont'd

- All statements in a segment are executed sequentially, non-preemptively, and in zero simulation time
- Only the delay can be preempted by other tasks
- No local variables are saved between segments

(All of this is needed because MATLAB functions cannot be preempted/resumed. . .)

Multiple Code Segments



Multiple code segments are needed to simulate

- input-output delays
- self-suspensions
- waiting for events, monitors, mailboxes or semaphores
- loops or branches

```
ttSetNextSegment(nbr)
```

Example of a Code Function

```
function [exectime, data] = Event_P_Ctrl(segment, data)
switch segment,
  case 1,
     ttWait('event');    % blocking
     exectime = 0;
  case 2,
     r = ttAnalogIn(1);
     y = ttAnalogIn(2);
     data.u = data.K * (r-y);
     exectime = 0.002 + 0.001*rand;
   case 3,
     ttAnalogOut(1, data.u);
     ttSetNextSegment(1);
     exectime = 0.001;
end
```

When to use the C++ API?

- When simulation takes too long time using MATLAB code
- When you want to define your own priority functions
- When you want to define your own kernel hooks

You must use a C++ compiler supported by the MEX facility of the MATLAB version that you are running

- Microsoft C++ Compiler (Visual Studio .NET)
- GNU compiler gcc on Linux/Mac OS

Example: PID-control of a DC-servo

- Consists of a single controller task implementing a standard PID-controller
- Continuous-time process dynamics

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

- Can evaluate the effect of sampling period and input-output latency on control performance
- Four different ways to implement periodic tasks are shown
- Both C++ function and M-file implementations will be demonstrated

Computer exercise 1a

Go to the directory `examples/servo/matlab` and open the model `servo.mdl`

- Study the implementation carefully
- Investigate the influence of the PID parameters K, T_i, T_d on the control performance – what happens if you double/half the values?
- Investigate the effect of the sampling interval h and the computational delay on the control performance

Computer exercise 1b

Go to the directory `examples/threeservos/matlab` and open the model `threeservos.mdl`

- The CPU is overloaded
- Study the various scheduling methods and the resulting control performance