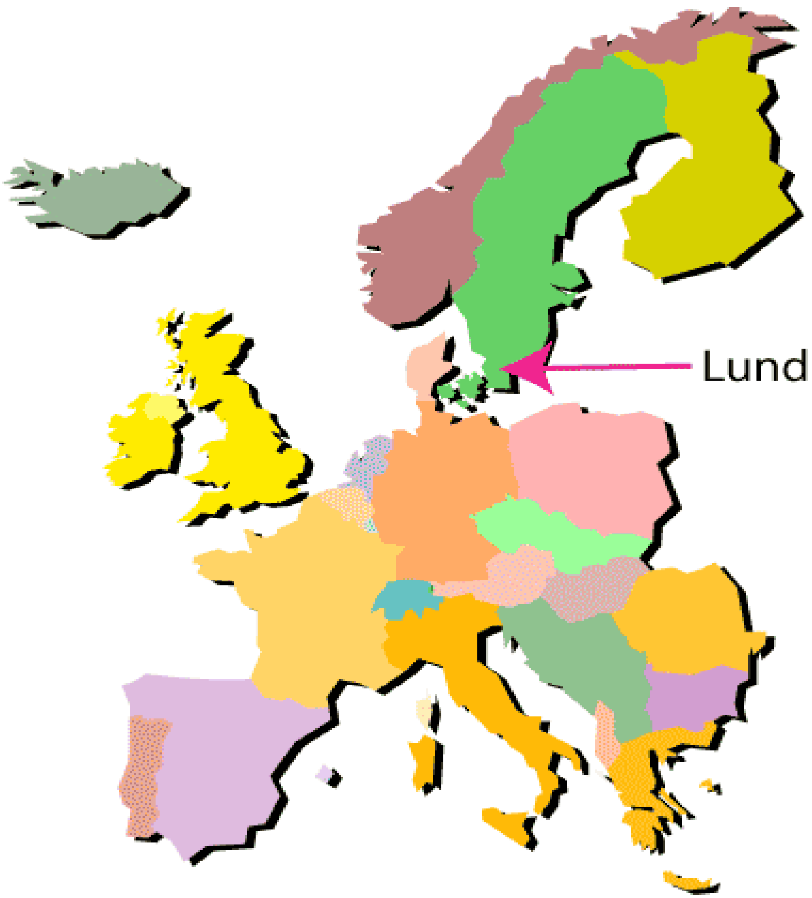
Analysis and Simulation of Embedded Control Performance using Jitterbug and TrueTime

Karl-Erik Årzén
Dept of Automatic Control
Lund University

Joint work with
Anton Cervin, Bo Lincoln, Dan Henriksson

LUND UNIVERSITY

Lund University

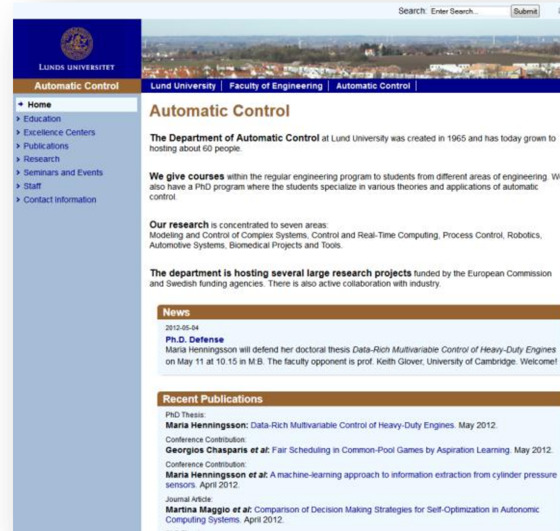


Founded in 1666

One of the largest institutes for higher education and research in Scandinavia with about 35 000 students

Department of Automatic Control

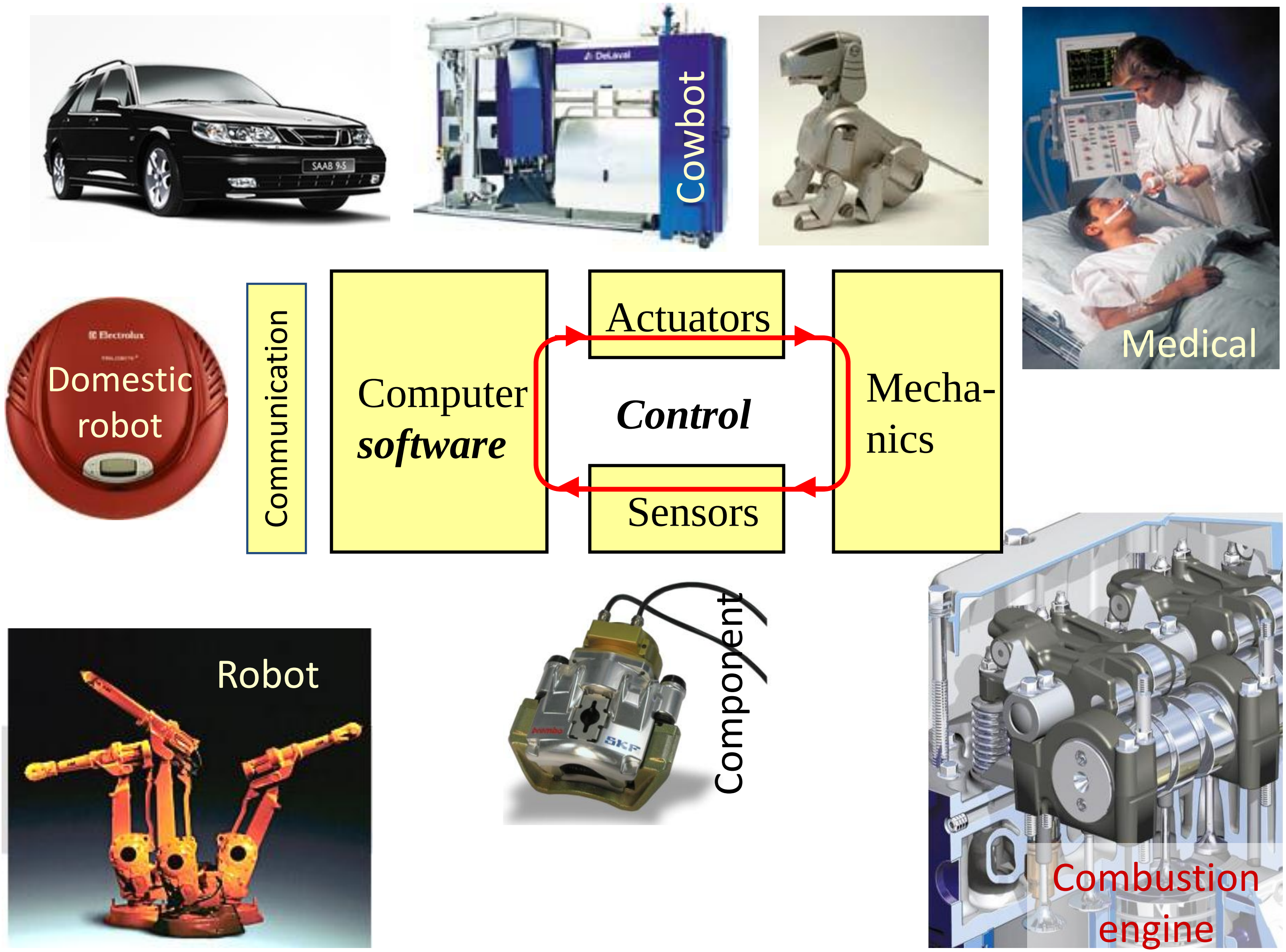
- One of the largest control departments in Europe
- Founded by Prof Karl Johan Åström
- Currently:
  - 10 Faculty
  - 34 PhD students
- Research areas:
  - Modeling and Control of Complex Systems
  - Control and Real-Time Computing
  - Process Control
  - Robotics
  - Automotive Systems
  - Biomedical Projects
  - Tools
- www.control.lth.se



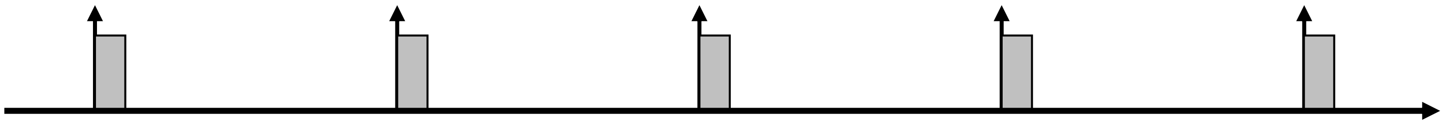
Contents

- **Networked Embedded Control**
- Stability Margin
- Average-case stochastic performance analysis using Jitterbug
- Simulation using TrueTime
- TrueTime in Modelica

Products relying on embedded control

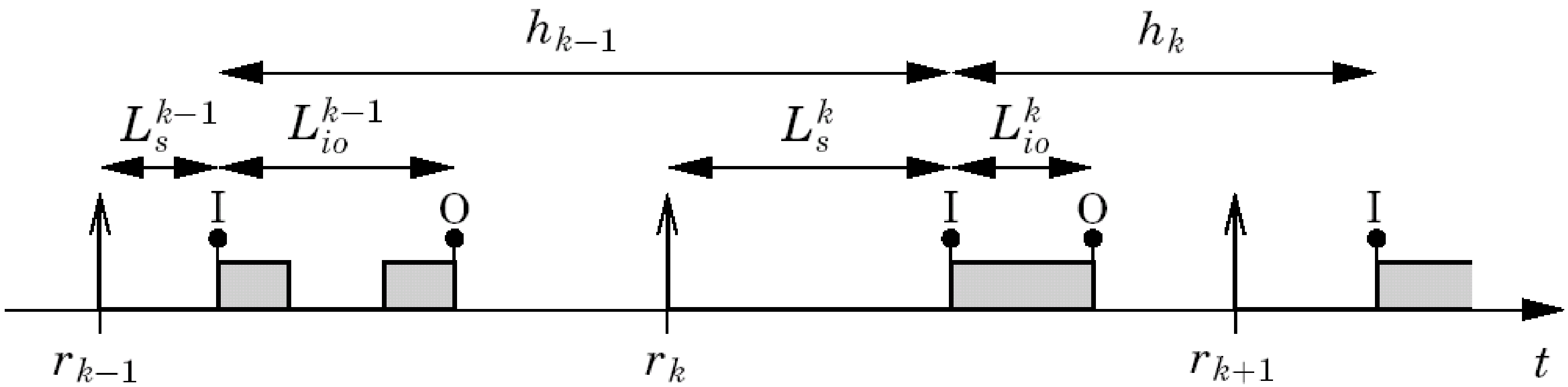


Control Loop Timing



- Classical control assumes deterministic sampling
  - in most cases periodic
  - too long sampling interval or too much jitter give poor performance or instability
- Classical control assumes negligible or constant input-output latencies
  - if the latency is small compared to the sampling interval it can be ignored
  - if the latency is constant it can be included in the control design
  - too long latency or too much jitter give poor performance or instability

Networked Embedded Control Timing



- Embedded control often implies temporal non-determinism
  - [illegible]
  - [illegible] on-deterministic kernel primitives, ...

How does this effect performance?

- Networked control often implies temporal non-determinism
  - network interface delay, queuing delay, transmission delay, propagation delay, link layer resending delay, transport layer ACK delay, ...
  - lost packets

Analysis of Control Performance

- Constant delays in linear systems --- straightforward
- Sampling jitter and input-output jitter -- more difficult
  - Worst-case stability analysis
    - Requires minimum and maximum values for the jitter
    - Stability margin theorems by Kao & Lincoln and by Cervin
  - Average-case stochastic performance analysis
    - Requires a stichastic model of latencies
    - Jitterbug toolbox
  - Simulation
    - TrueTime toolbox

Analysis of Control Performance

The control performance depends on a number of issues:

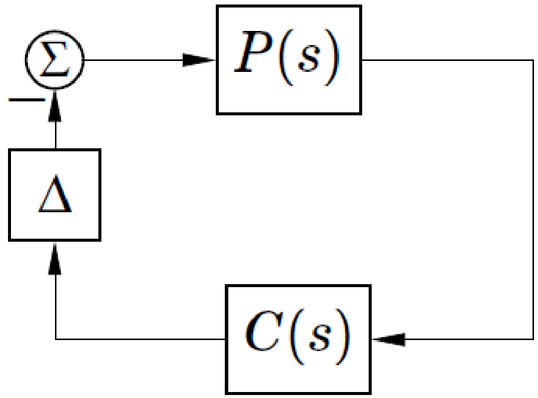
- The dynamics of the plant that is controlled
- The controller type
- The design specifications for the controller
- The nature of the disturbances
- The delay distribution
- What type of control performance that we are interested in

Contents

- Networked Embedded Control
- **Stability Margin**
- Average-case stochastic performance analysis using Jitterbug
- Simulation using TrueTime
- TrueTime in Modelica

Jitter Margin – Stability under Input-Output Jitter

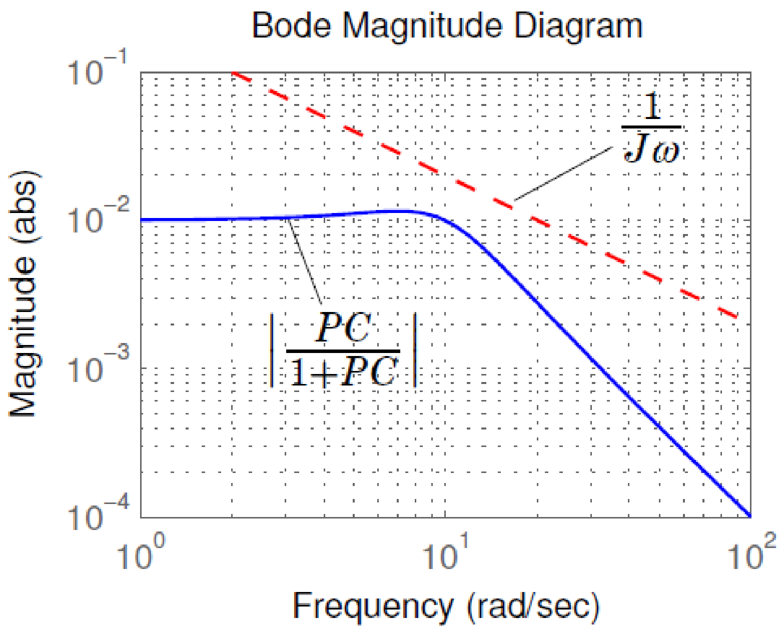
Stability theorem due to Kao and Lincoln (2004):



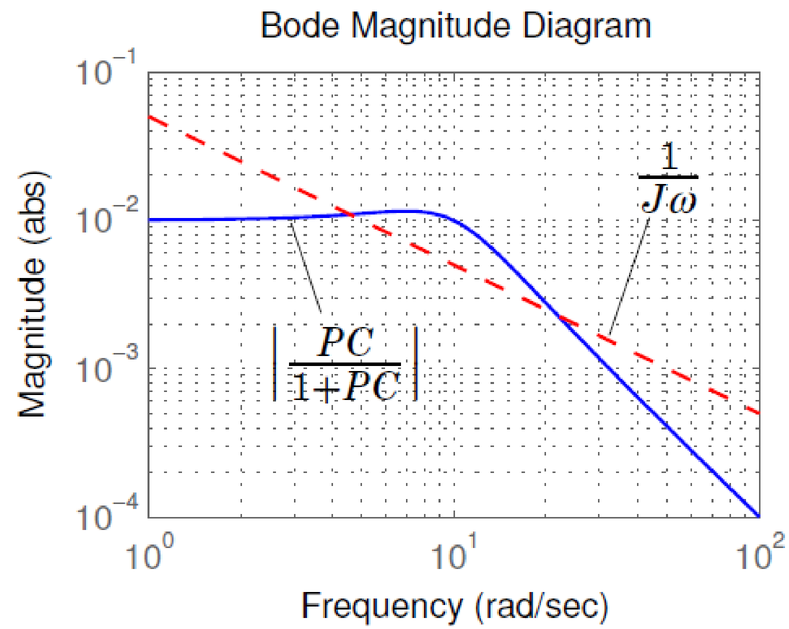
- Continuous-time plant $P(s)$
- Continuous-time controller $C(s)$
- Arbitrarily time-varying delay $\Delta \in [0, J]$
- Theorem: The closed-loop system is stable **if**

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

Graphical test:



Stable



Could be unstable

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 $J \leq h$.

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

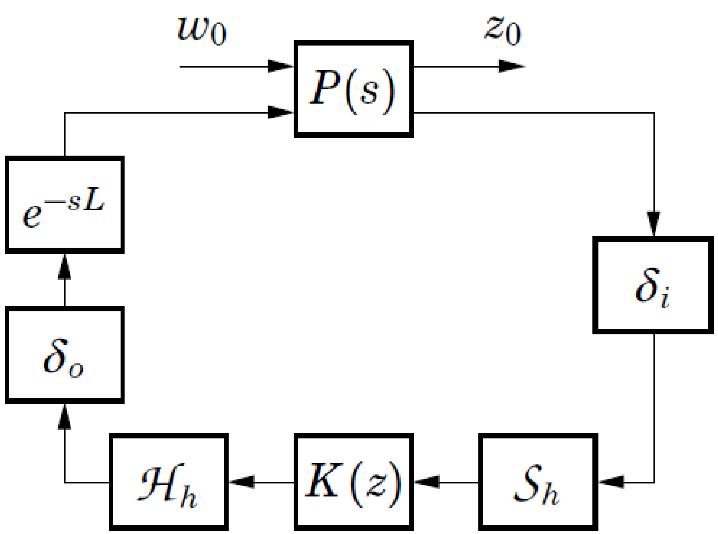
- $P_{\text{alias}}(\omega) = \sqrt{\sum_{k=-\infty}^{\infty} \left| P\left(i(\omega + 2\pi k) \frac{1}{h} \right) \right|^2}$
- $P_{\text{ZOH}}(z)$ is the ZOH-discretization of $P(s)$

Jitter Margin Limitations

- 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

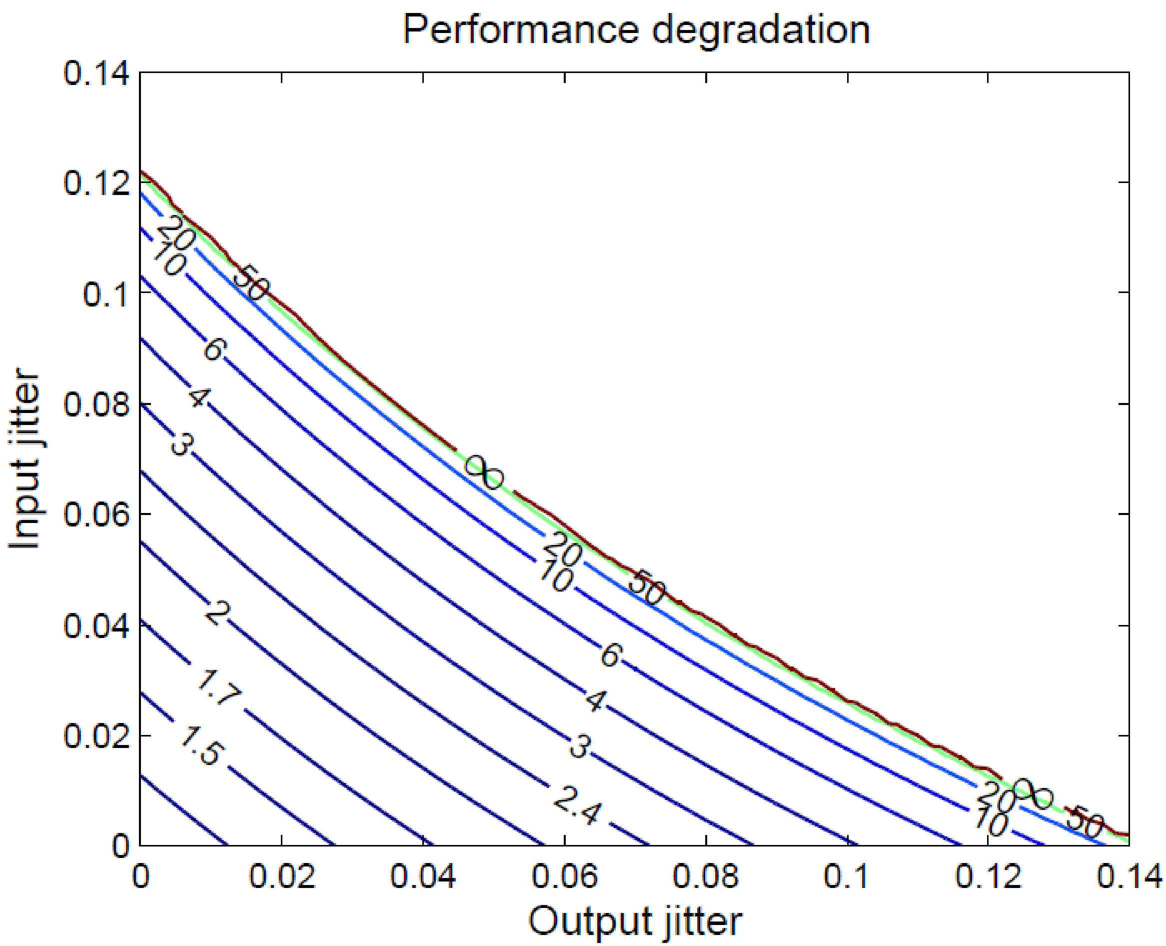
Jitter Margin for Input and Output Jitter

- Cervin (ACC 2012)



- Nominal input–output delay L
- Time-varying input delay $\delta_i(t) \in [-\frac{J_i}{2}, \frac{J_i}{2}]$
- Time-varying output delay $\delta_o(t) \in [-\frac{J_o}{2}, \frac{J_o}{2}]$
- H_∞-type performance metric: $\gamma = \sup \dfrac{\|z_0\|}{\|w_0\|}$

$$P(s) = \frac{1}{s^2 - 0.01}, \quad K(z) = \frac{-12.95z^2 + 10z}{z^2 - 0.2181z + 0.1081}, \quad h = 0.2, \quad L = 0.08$$



Contents

- Networked Embedded Control
- Stability Margin
- **Average-case stochastic performance analysis using Jitterbug**
- Simulation using TrueTime
- TrueTime in Modelica

Jitterbug

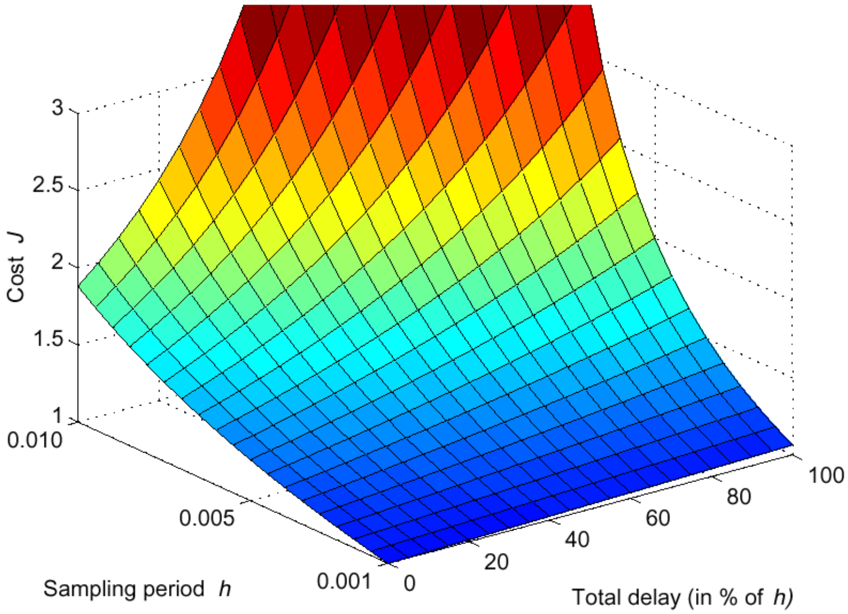
- Matlab toolbox for stochastic control analysis (Lincoln and Cervin, 2002)
- Random delays in the control loop described by probability distributions
- System disturbed by white noise
- Performance measured by quadratic cost function

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

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

Jitterbug

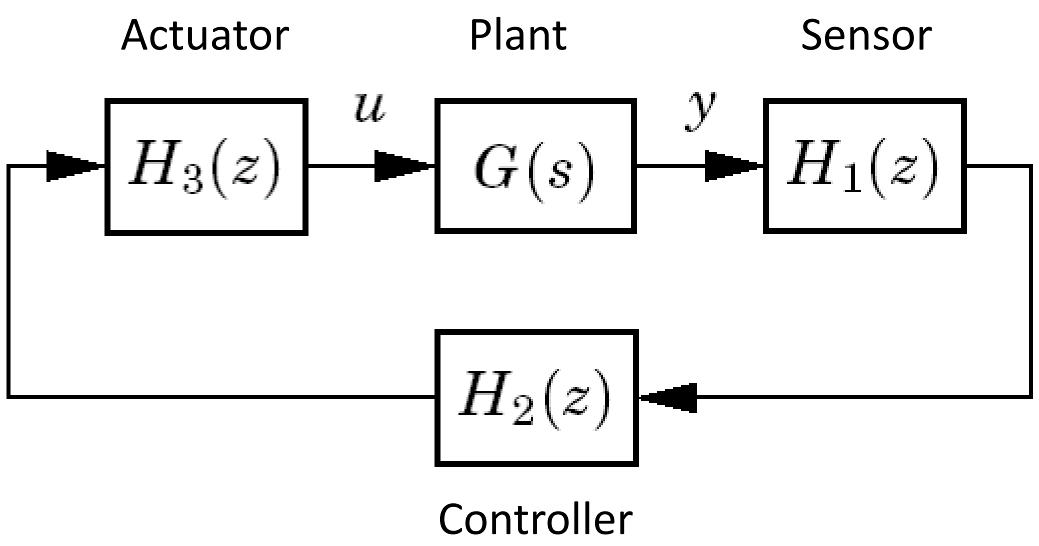
- Matlab-based toolbox for analysis of real-time control performance
- Evaluate effects of latencies, jitter, lost samples, aborted computations, etc on control performance
- Analyze jitter-compensating controllers, aperiodic controllers, multi-rate controllers
- Calculation of a quadratic performance criterion function
- Packaging of existing theory for linear quadratic Gaussian systems and jump-linear systems



Jitterbug Analysis

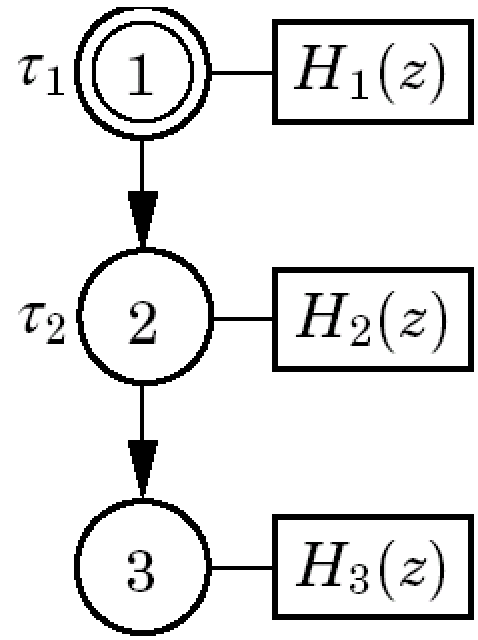
- System described using a number of connected continuous-time and discrete-time transfer function blocks driven by white noise

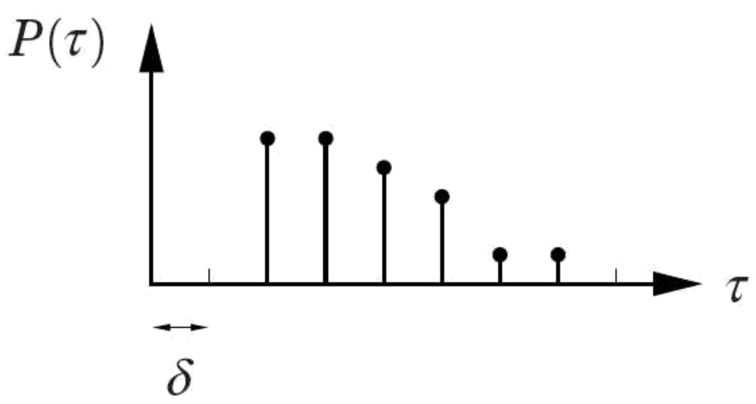
Distributed Control Loop:



Jitterbug Analysis

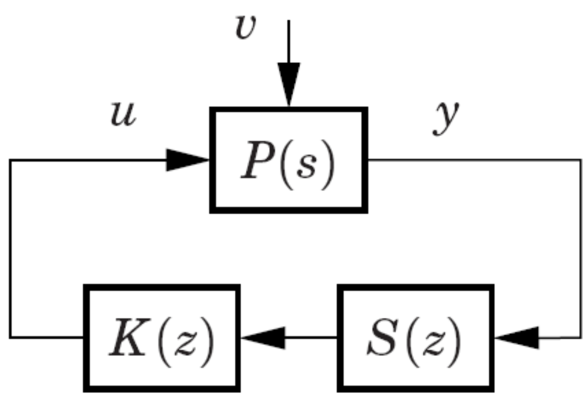
- The execution of the blocks is described by a stochastic timing model expressed as an automaton
- Each state can trigger one or more discrete systems
- Time intervals are represented by discrete probability distributions



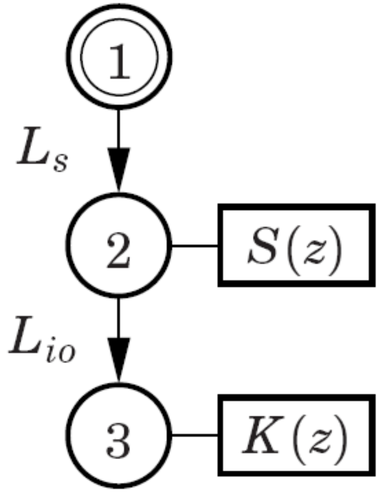


Jitterbug Model - Example

Signal model:



Timing model:



- $P(s)$ – process
- $S(z)$ – sampler
- $K(z)$ – controller/actuator

Jitterbug Example Script

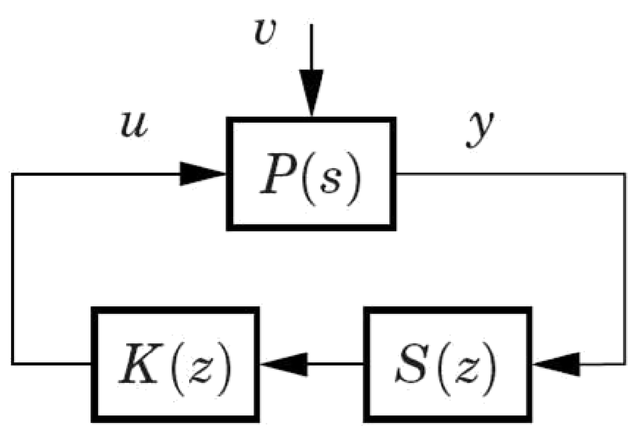
```
Ptau1 = 1; % Corresponds to zero delay
Ptau2 = [zeros(1,round(L/dt)) 1];
N = initjitterbug(dt,h);
  % timegrain dt, periodic system with period h
N = addtimingnode(N,1,Ptau1,2);
  % add timing node with delay distributin to next node
N = addtimingnode(N,2,Ptau2,3);
N = addtimingnode(N,3);
  % add timing node with no next node
```

Jitterbug example script

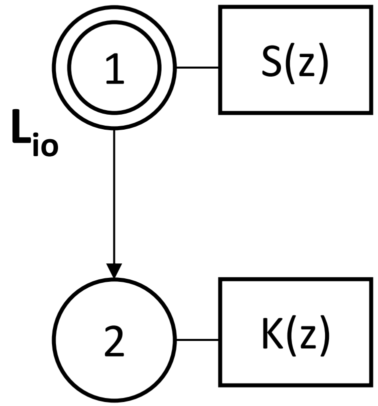
```
N = addcontsys(N,1,plant,3,Q,R1,R2);
  % add cont-time LTI system taking its input from syst 3
N = adddiscsys(N,2,1,1,2);
  % add disc-time LTI system (sampler) taking its input
  % from system 1 and executing in timing node 2
N = adddiscsys(N,3,ctrl,2,3);
  % add disc-time LTI system (controller) taking its
  % input from system 2 and executing in timing node 3
N = calcdynamics(N);
  % Calculate internal dynamics
J = calccost(N)
  % Calculate (and display) cost
```

Simple Example

Signal model:



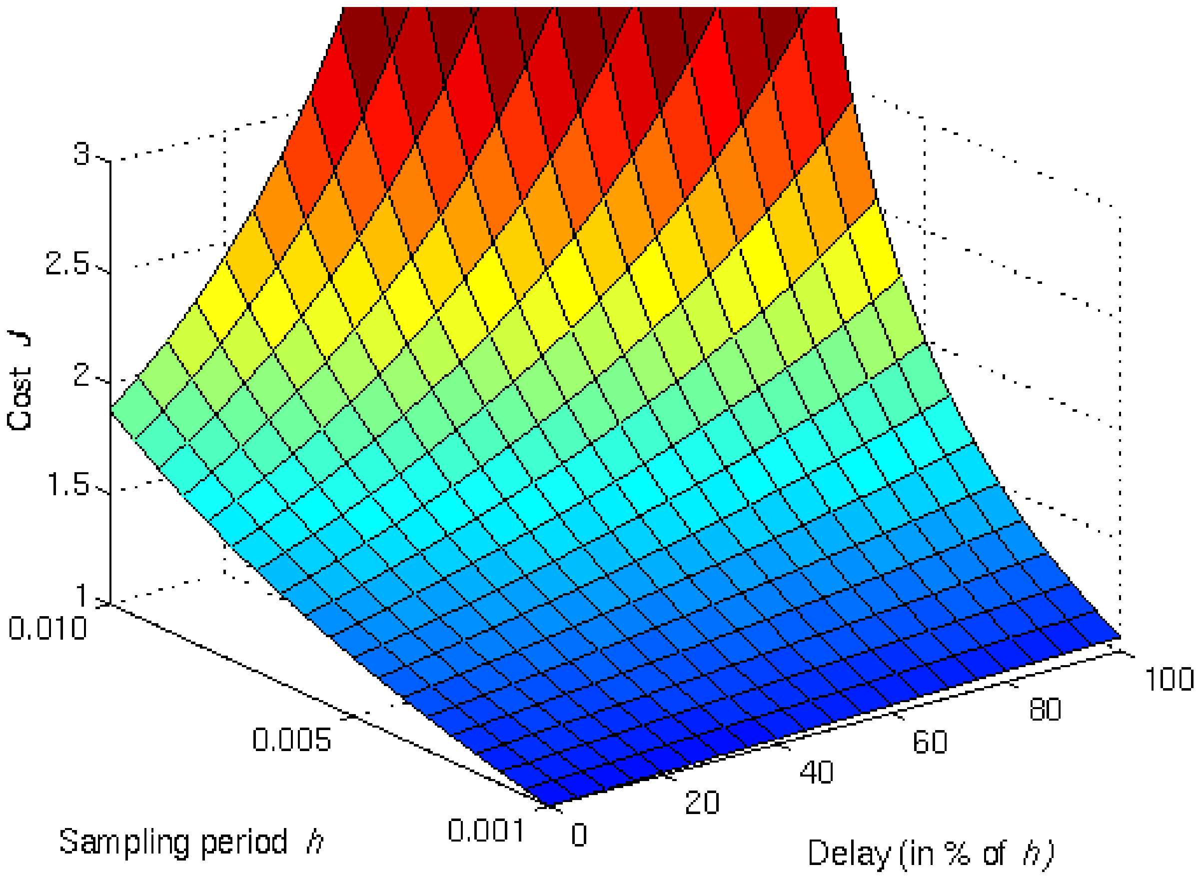
Timing model:



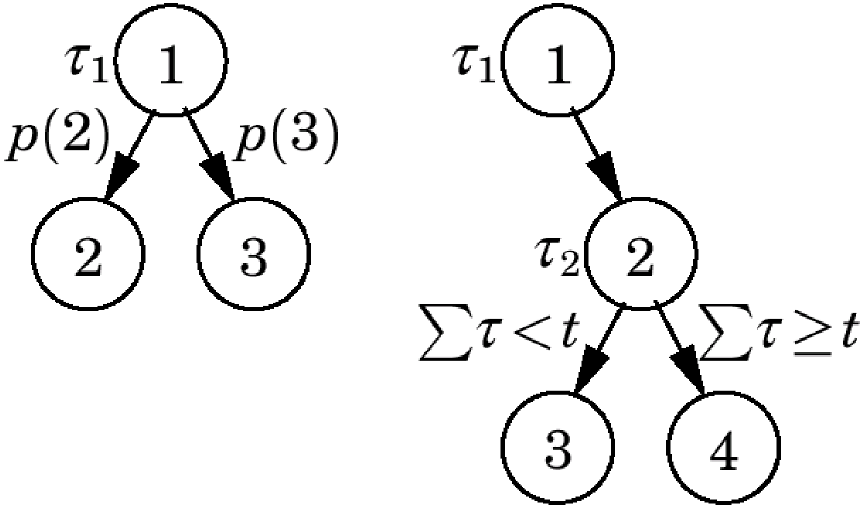
- P(s) – Process (Inverted pendulum)
- S(z) – Sampler (perfect sampling)
- K(z) – Controller + actuator
- L_{io} – input output latency

Demo

Results


3
2.5
2
1.5
1
Cost J
0.010
0.005
0.001
Sampling period h
0
20
40
60
80
100
Delay (in % of h)

More complicated cases



- random choice of path
- choice of path depending on delay
- different update equations in different nodes
- aperiodic systems
- ...

Aperiodic Systems

- Jitterbug supports both periodic and aperiodic systems
- Periodic:
  - `>calccost`
  - Analytical solution, reasonably fast
- Aperiodic:
  - `>calccostiter`
  - Iterative computation with possibly very slow convergence

Internal Workings

1. Sample the continuous-time system, the noise, and the cost function with the time-grain δ
2. Translate the timing model into a Markov chain
3. Formulate the closed-loop system as a discrete-time jump linear system

$$x(k+1) = \Phi_i(k)x(k) + e(k), \qquad \mathrm{E}\{e(k)e^T(k)\} = R_i(k)$$

   where $\Phi_i(k)$ and $R_i(k)$ depends on the Markov state i

4. Compute the stationary covariance $P = \mathrm{E}\{xx^T\}$ from

$$P_i(k+1) = \mathrm{E}\{\Phi_i(k)P_i(k)\Phi_i(k)^T + R_i(k)\}$$

Computational Complexity

- A continuous-time system of order n requires n internal states
- A discrete system of order n requires $n + 1$ internal states (one extra for the output)
- The stationary covariance P can be found directly by solving a linear system of equations of dimension n^2
  - n – total number of internal states
- The amount of memory required is $n^4 2m(p + 1)$
  - m – number of timing nodes
  - p – number of time-steps per period ($= \frac{h}{\delta}$)

Pros and cons

Pros:

- Analytical performance computation
- Fast to evaluate cost for a wide range of parameters
- Guarantees stability (in mean-square sense) if cost is finite

Cons:

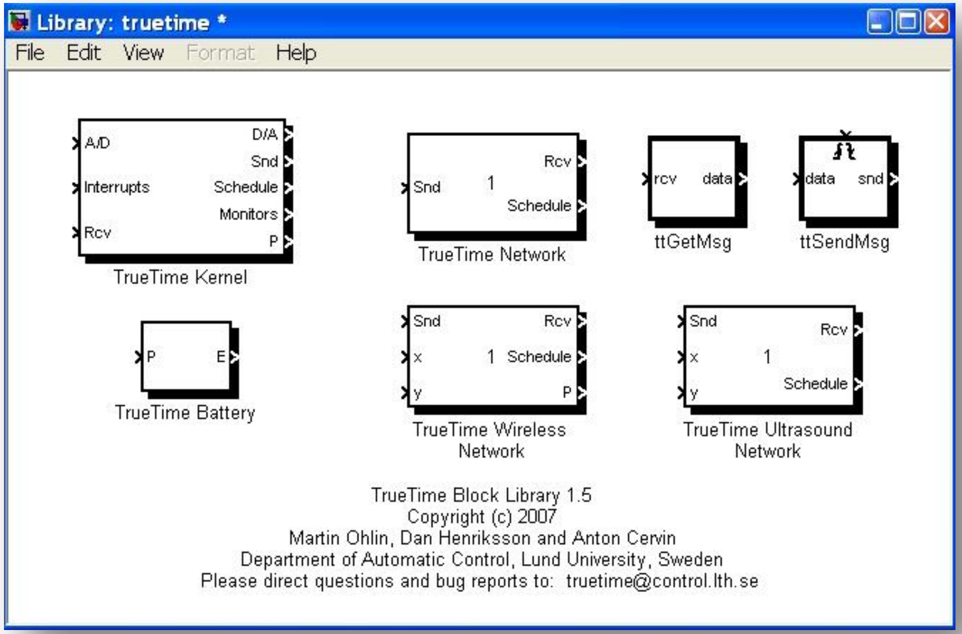
- Simplistic timing models
  - Indepedent delays
  - Delay distributions may not change over time
- Only linear systems and quadratic costs
- Requires knowledge about latency distributions
  - Where do we get this from?
  - Existing scheduling theory can at best give worst-and best-case values
- Statistical analysis
  - The calculated cost is an expected value
  - All results only hold in a mean-value sense
    - Not suitable as a basis for formal verification
  - Timing scenarios with probability zero are disregarded by the analysis
    - E.g. switching-induced instability

Contents

- Networked Embedded Control
- Stability Margin
- Average-case stochastic performance analysis using Jitterbug
- **Simulation using TrueTime**
- TrueTime in Modelica

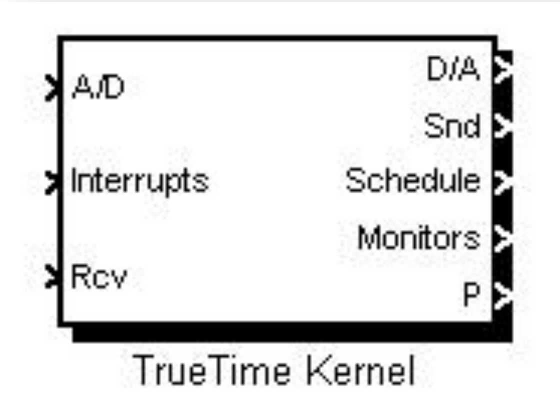
TrueTime

- Simulator for the cyber parts of CPS
- Embedded in physical system simulators (Simulink, Modelica)
- Simulation of
  - Real-time kernels
  - Wired and wireless networks
- Developed in Lund since 1999
  - Version 2.0
  - Large userbase
  - GPL



Modeling of Computations

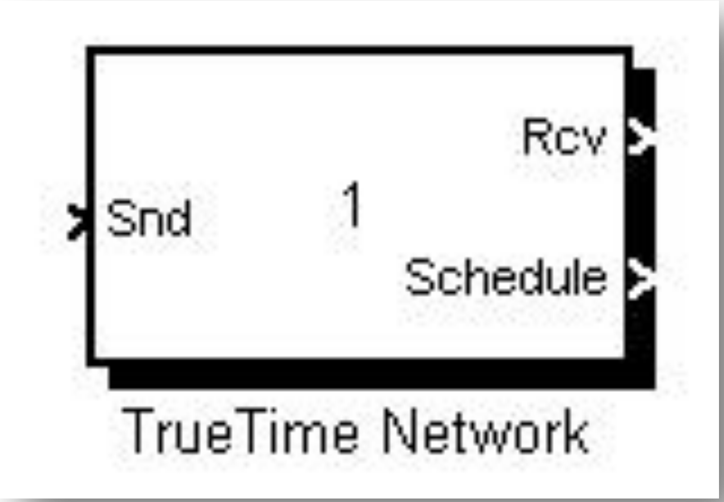
- Simulates an event-based real-time kernel
- Executes user-defined tasks and interrupt handlers
  - C/C++ or M-files
- Arbitrary user-defined scheduling policies
- Real-time primitives
- Code structured into code segments
  - emulate multithreading




```
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
```

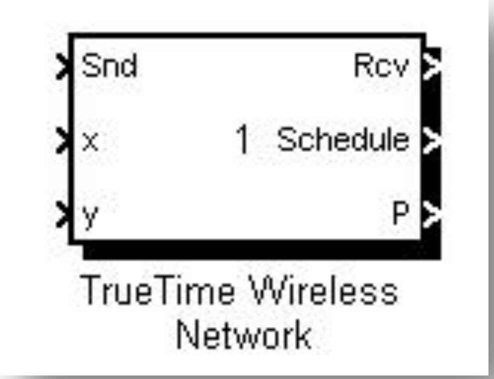
Modeling of Wired Networks

- Models the medium access delay and the transmission delay
- A number of pre-defined data-link layer protocols
  - Switched Ethernet
  - CAN
  - Round Robin
  - FDMA
  - TDMA
  - CSMA/CD (Shared Ethernet)
  - Flexray
  - PROFINET IO

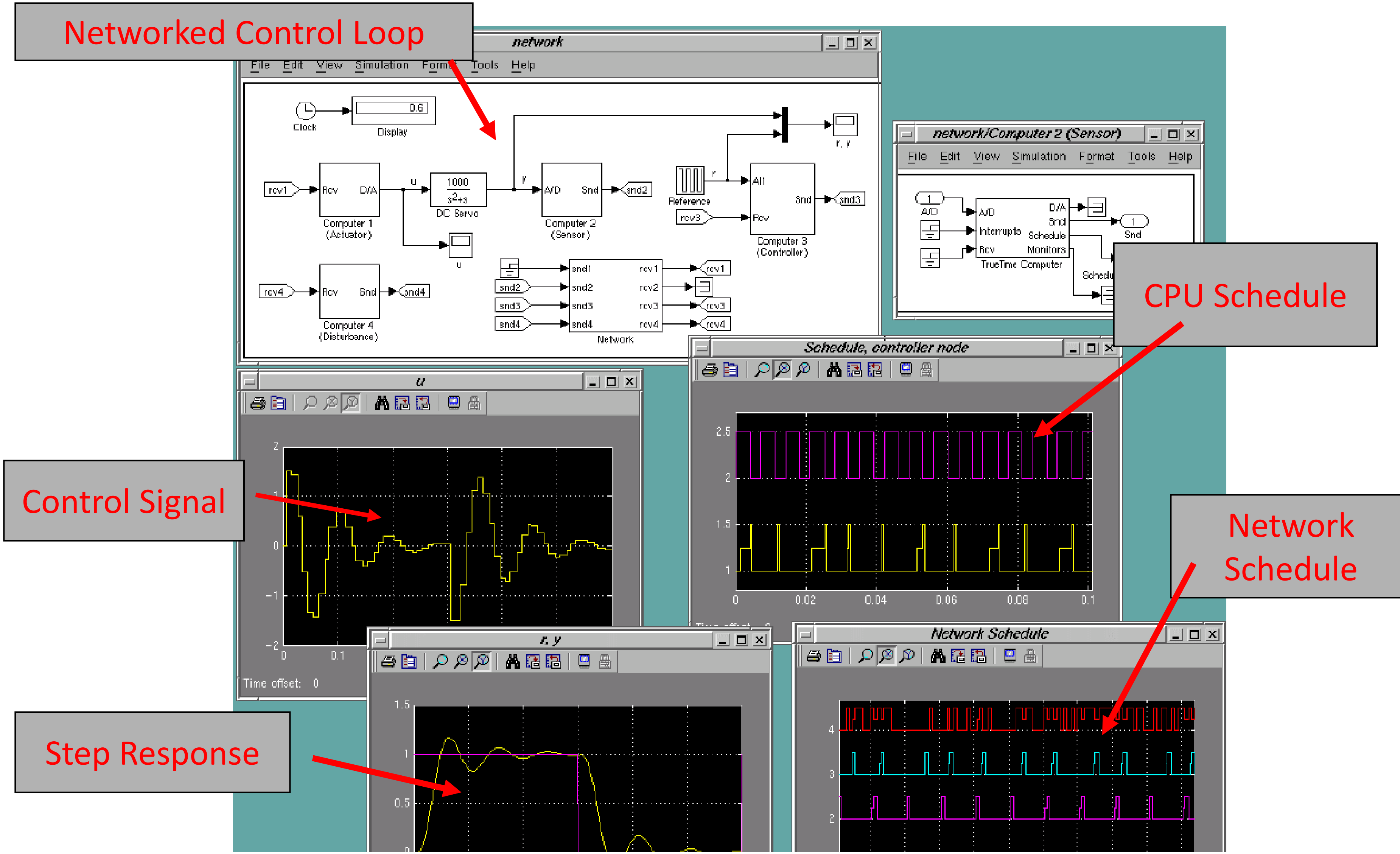


Modeling of Wireless Networks

- Supports two common MAC layer policies:
  - IEEE 802.11 b/g (WLAN)
  - IEEE 802.15.4 (“ZigBee”)
  - (Wireless HART - implemented by ABB)
- x and y inputs for node locations (2D)
- Radio models:
  - Exponential path loss (default)
  - User-defined models to model multi-path propagation, fading etc

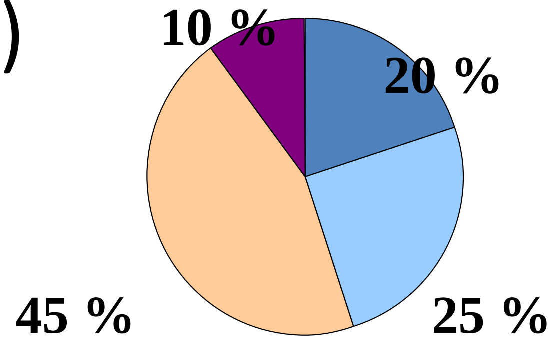


TrueTime: Networked Embedded Control

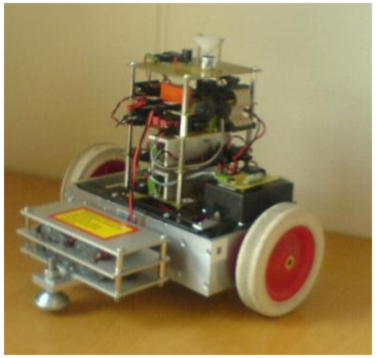


New Features

- Multicore kernels
  - Each TrueTime kernel may have multiple cores
  - Partitioned scheduling
  - `ttSetNumberOfCPUs(no)`
  - `ttSetCPUAffinity(task,cpu)`
- Constant bandwidth servers (CBS)
  - Virtual processors
  - Temporal isolation
  - `ttCreateCBS(budget,period)`
  - `ttAttachCBS(task,CBS)`
  - `ttSetCBSParameters(budget, period)`



TrueTime: Mobile Robotics



- Tunnel road safety scenario in RUNES
  - EU FP6 IP (2004-2007)
  - Coordinated by Ericsson
- Stationary sensor network in a road tunnel

- Mobile robots as mobile gateways for restoring connectivity among isolated subislands of the network

Localization

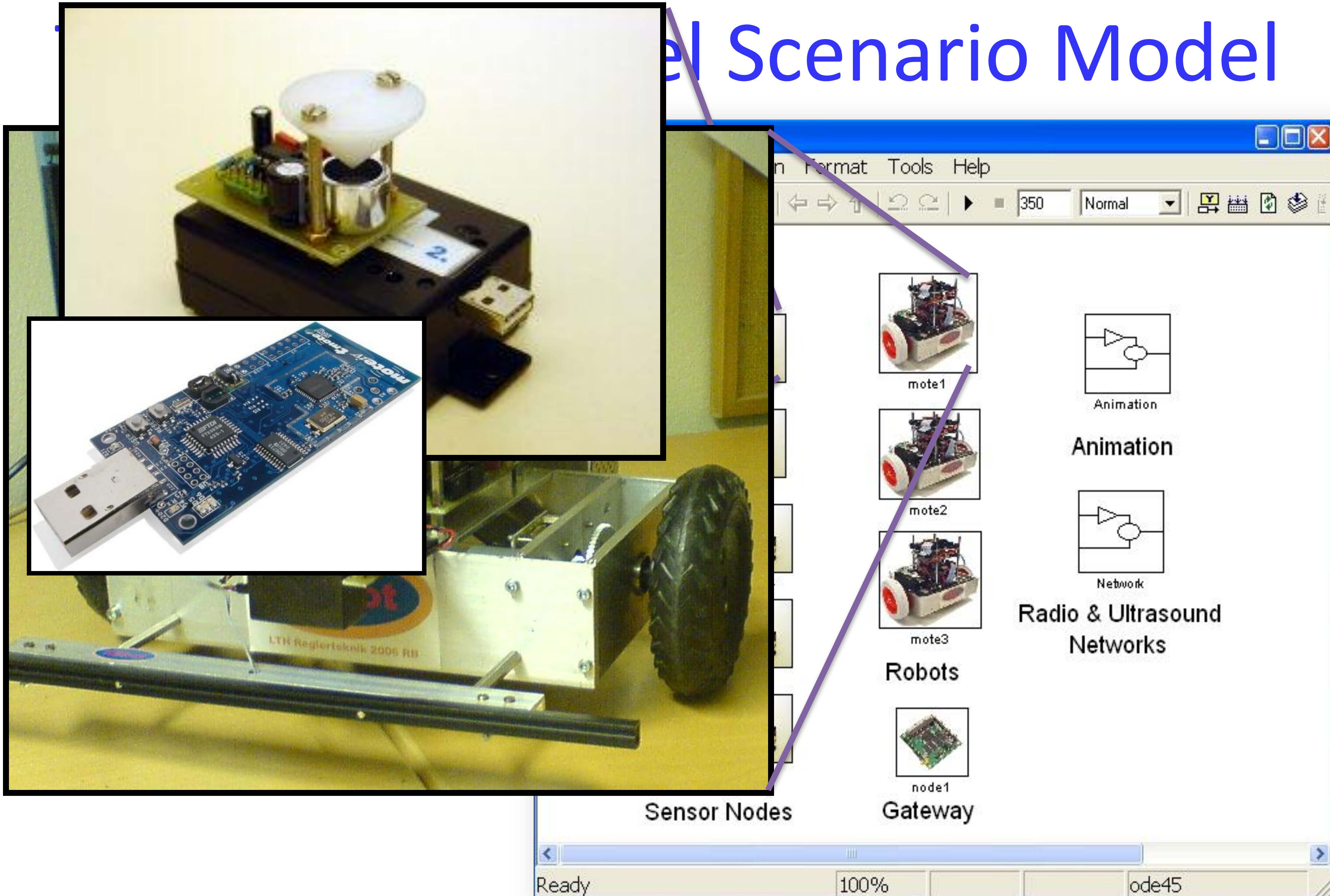
- Ultrasound-based
  - Active mobile robots
  - Passive stationary nodes
- Robot broadcasts radio packet and ultrasound pulse ”simultaneously”
- Difference in time-of-arrival allows each reachable node to calculate its distance to the robot
- Each node sends its distance measurement back to the robot
- Extended Kalman Filter fuses distance measurements with wheel encoders

Verification Problem

- Robot with several microprocessors, I2C bus communication
- Sensor network radio communication
  - IEEE 802.11 b/g (WLAN)
  - AODV routing protocol
- Ultrasound localization
- IR-based obstacle avoidance
- Control and estimation

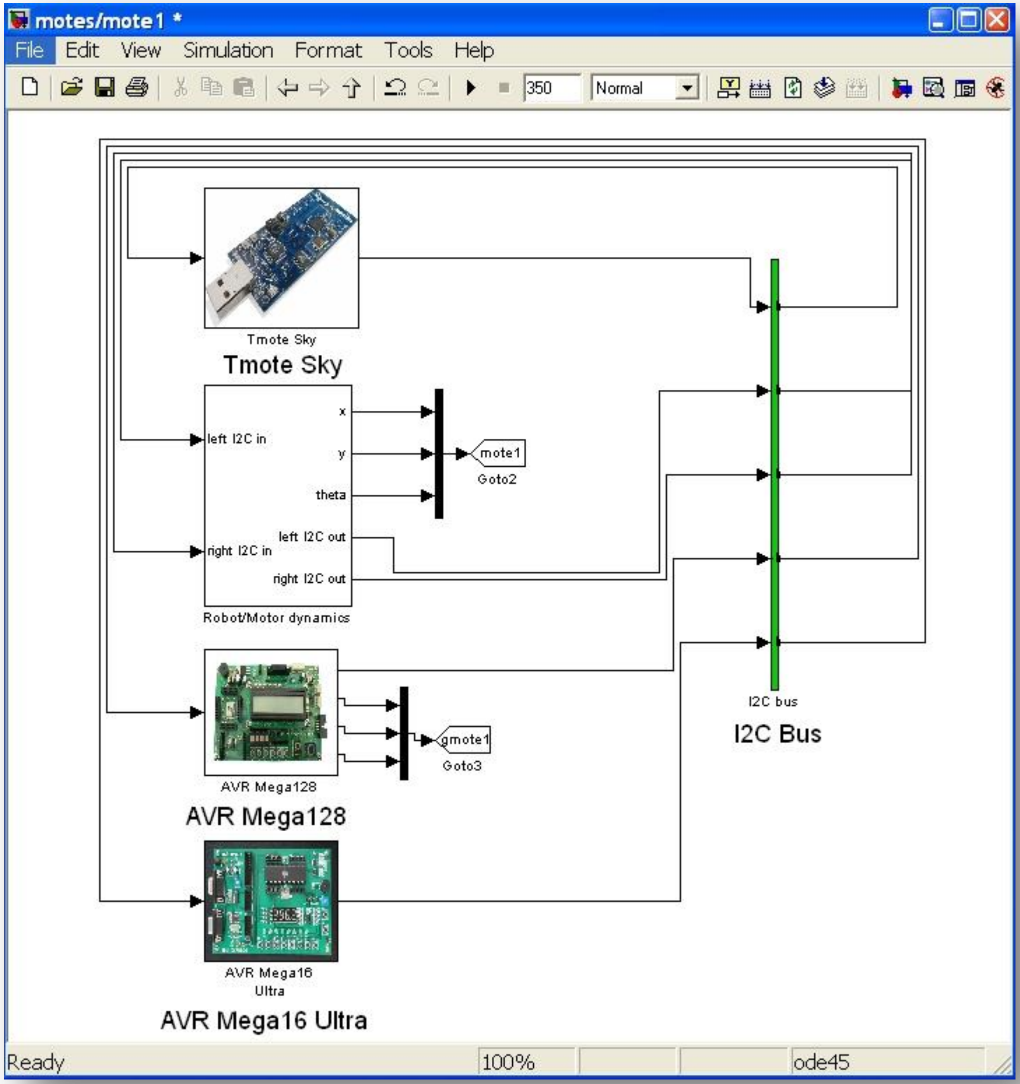
➡ How verify the functionality and timeliness of this??
  - TrueTime used for developing a simulator in parallel with the real physical implementation
  - Proof of concept and verification

el Scenario Model

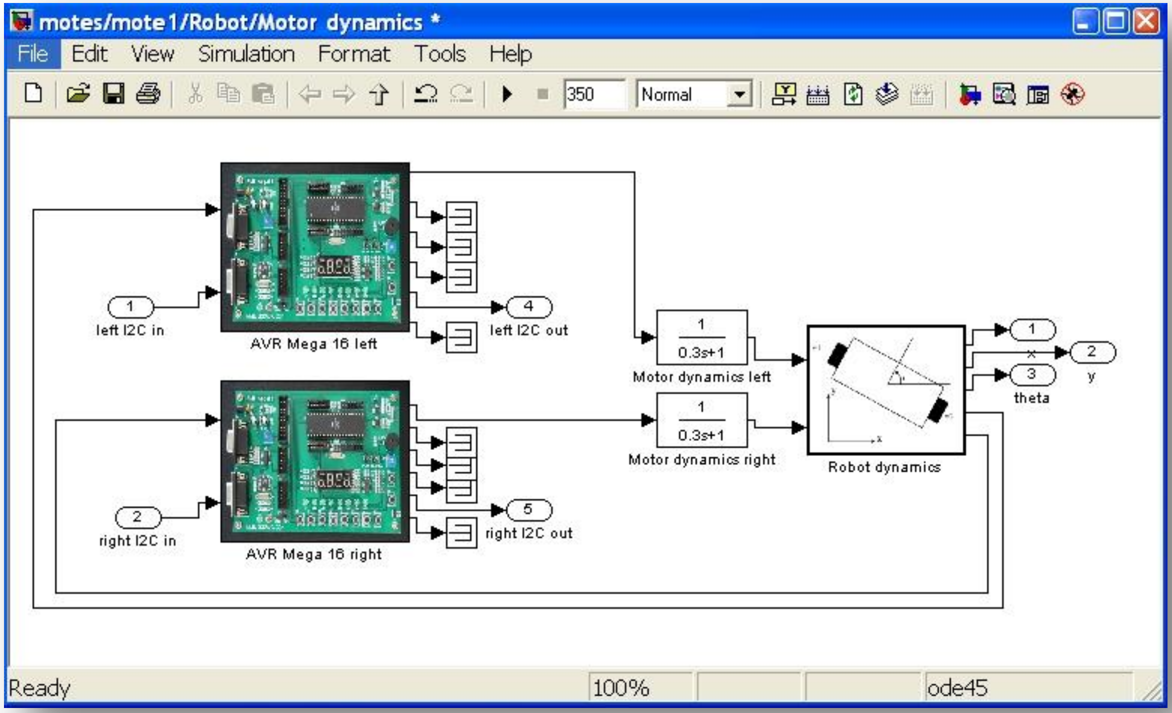


Robot Submodel

- Tmote Sky
  - Radio interface & bus master
  - Robot controller
- AVR Mega128
  - Compute engine
  - IR interface
  - EKF, navigation, and obstacle avoidance
- AVR Mega16
  - Ultrasound interface
- I2C bus
- Wheel and motor submodel

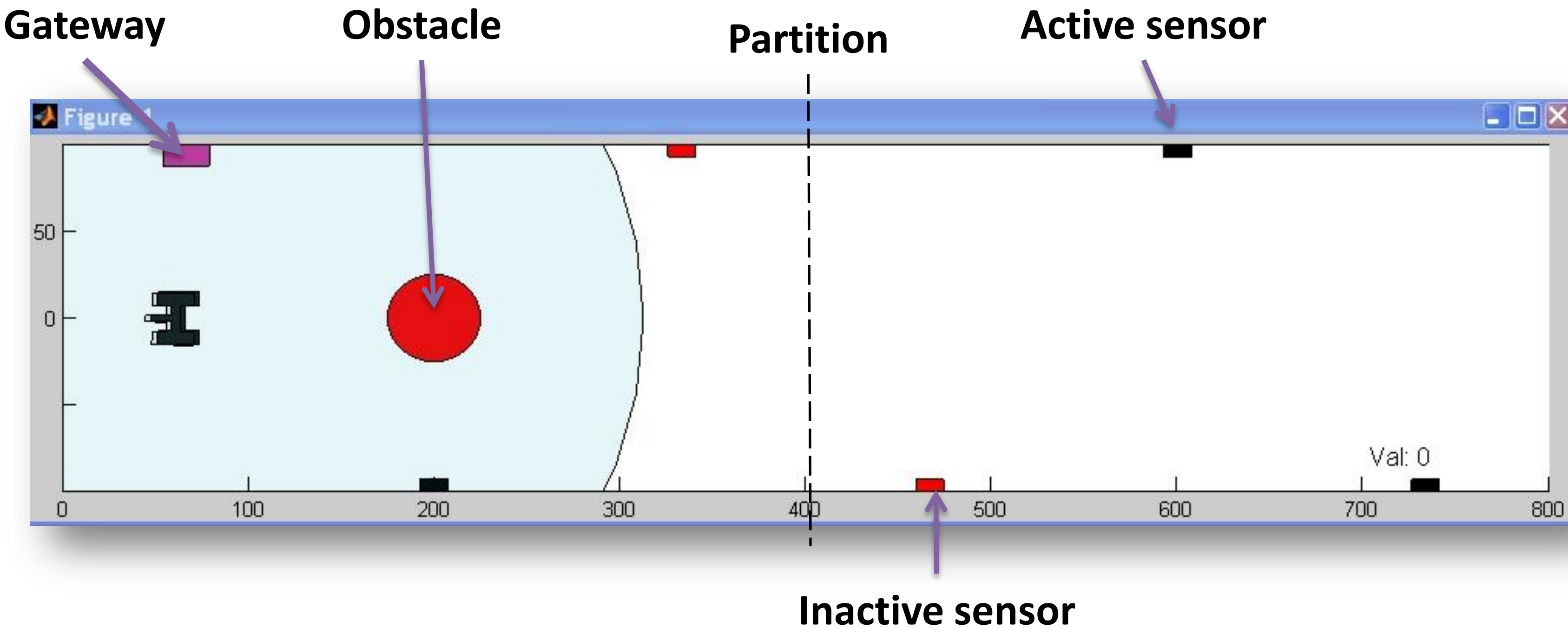


Wheel and Motor Submodel



- One AVR Mega16 for each wheel/motor
- Simple motor models
- Dual-drive unicycle robot dynamics model

Animation



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

Demo

Video Demo

Contents

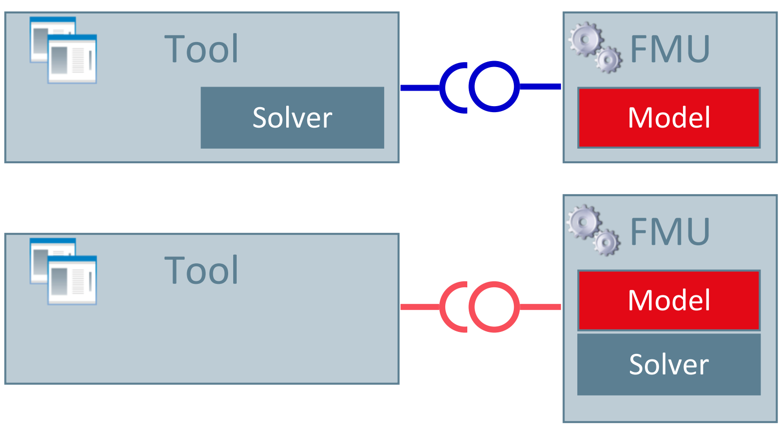
- Networked Embedded Control
- Stability Margin
- Average-case stochastic performance analysis using Jitterbug
- Simulation using TrueTime
- **TrueTime in Modelica**

TrueTime for Simulink

- S-function interface
  - Kernels
  - Networks
- Task code
  - C/C++
  - M-file script language

TrueTime for Modelica

- Network part
  - Native Modelica version available
  - External C code version for Dymola available
- Full TrueTime
  - Flexible Mockup Interface (FMI)
    - Open source non-proprietary model exchange format
    - Model Exchange
    - Co-Simulation



TrueTime for FMI

- Kernels and Networks are Flexible Mockup Units (FMUs)
  - Modelica simulation tools:
    - Dymola
    - Open-source tools: OpenModelica, JModelica
  - Non-Modelica tools that embrace FMI
- Task code written in C
- Work in progress
  - Vanderbilt University
  - DARPA Adaptive Vehicle Make (AVM) programme
  - TrueTime a part of the Meta toolchain for CPS

Conclusions

- Networked embedded control often implies temporal nondeterminism
- New tools are needed to simplify the design space exploration
- Three examples:
  - Jitter margin – worst-case stability results
  - Jitterbug – average-case stochastic performance analysis
  - TrueTime – simulation of real-time kernels and networks
- Available through www.control.lth.se