Networked Control Systems

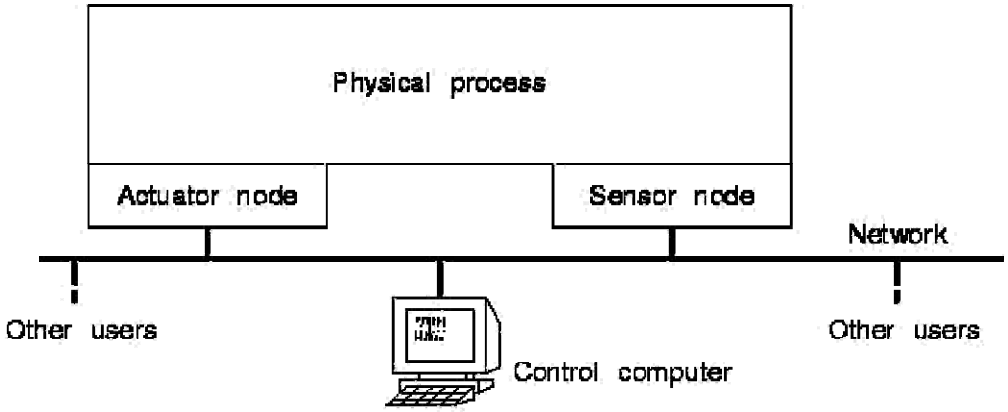
Anton Cervin and Karl-Erik Årzén

Lund University

Outline

- **Overview**
- Analysis and Design
  - Constant network delays
  - Varying network delays
- Jitterbug
  - Stochastic analysis of networked control loops
- TrueTime
  - Simulation of networked control loops

Motivation



- Reduced cabling costs
- Network hardware cheaper
- Manufacturer independent nodes
- Modularity and flexibility in system design
- ….

Networks from a control point of view

- Networks induce delays:
  - limited bandwidth
  - overhead in network interface
  - overhead in network
- Time delays in control loops:
  - give rise to phase lag
  - degenerate system stability and performance

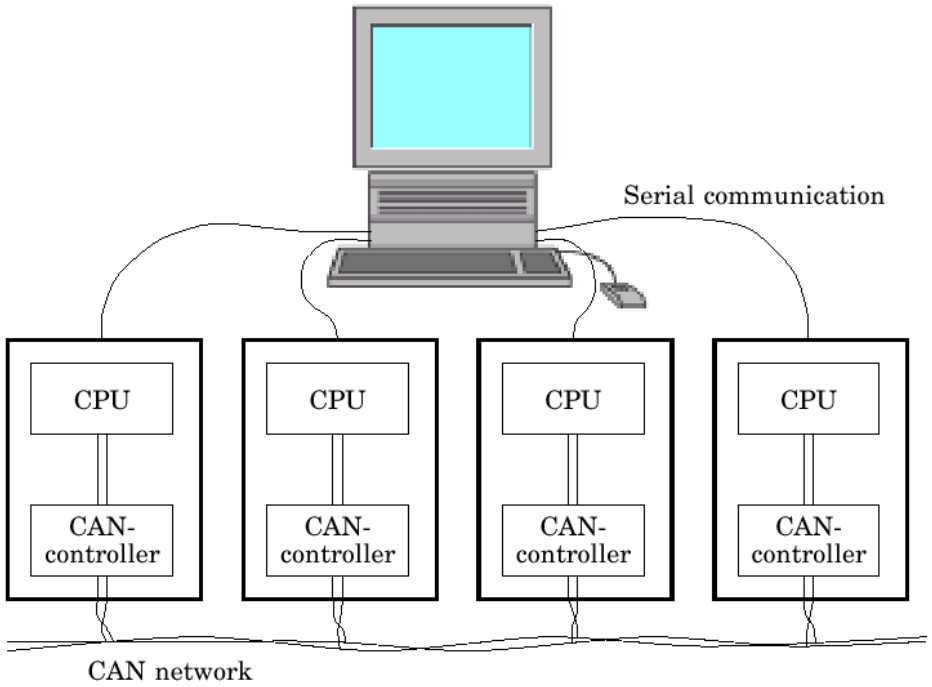
Control messages

- Small in size
- Frequent, often periodic
- Fresh data with “Best consumed before” time
- Losing occasional messages is often acceptable

Network Types

- Different networks give different levels of determinism:
  - constant delay (no jitter)
  - stochastically varying delays
- Design trade-off between delay and jitter

CAN: Experimental Data



CAN: Experimental Data

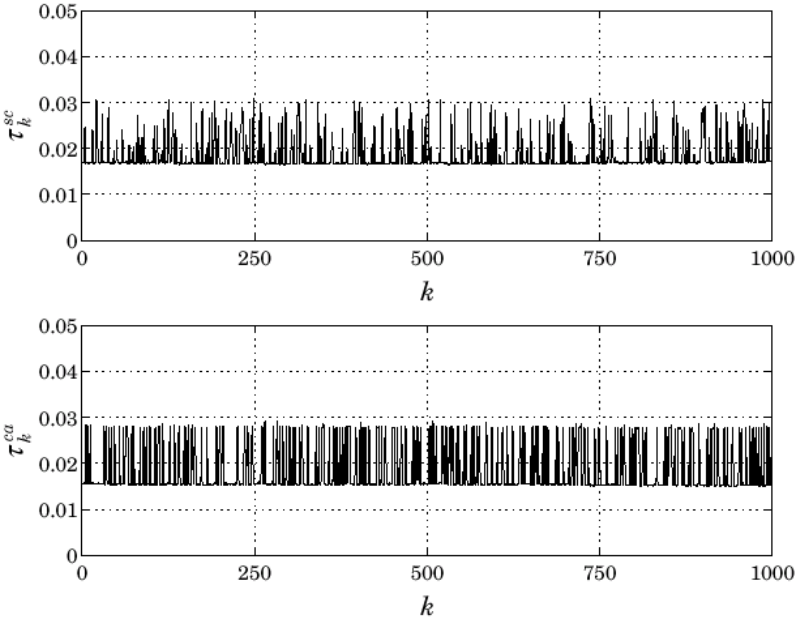


Figure 4.9 Load: One sender with period rect(40, 80) ms and higher priority. Message length: 8 bytes. The delay behavior is non-periodic.

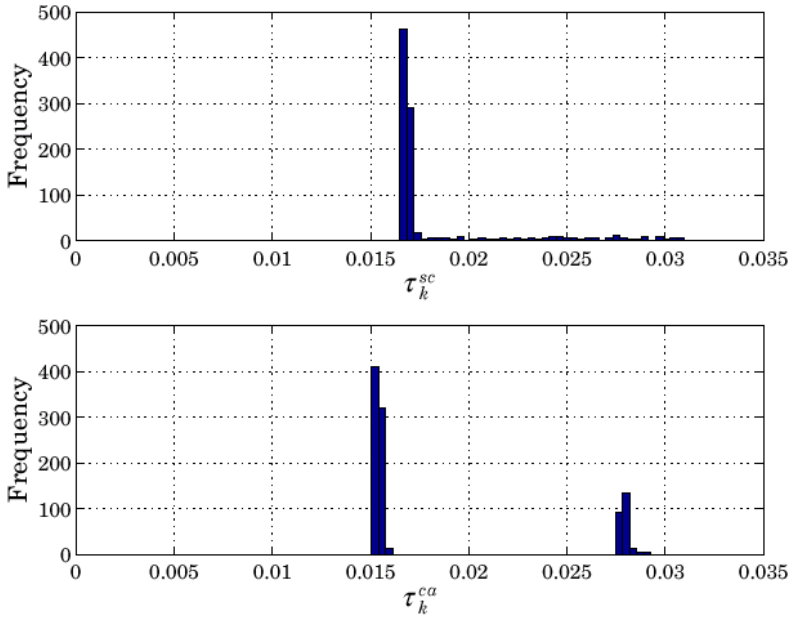
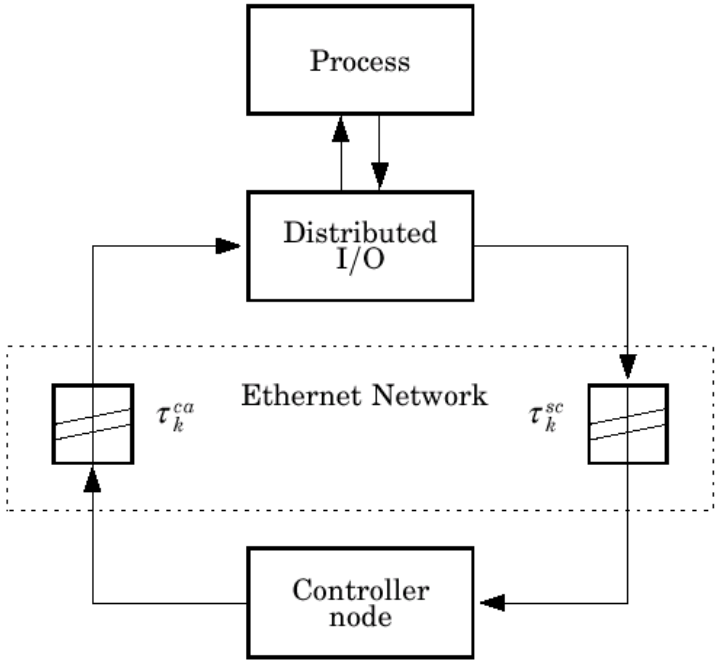


Figure 4.10 Distribution of the 1000 samples in the experiment. Load: One sender with period rect(40, 80) ms and higher priority. Message length: 8 bytes.

Ethernet: Experimental data



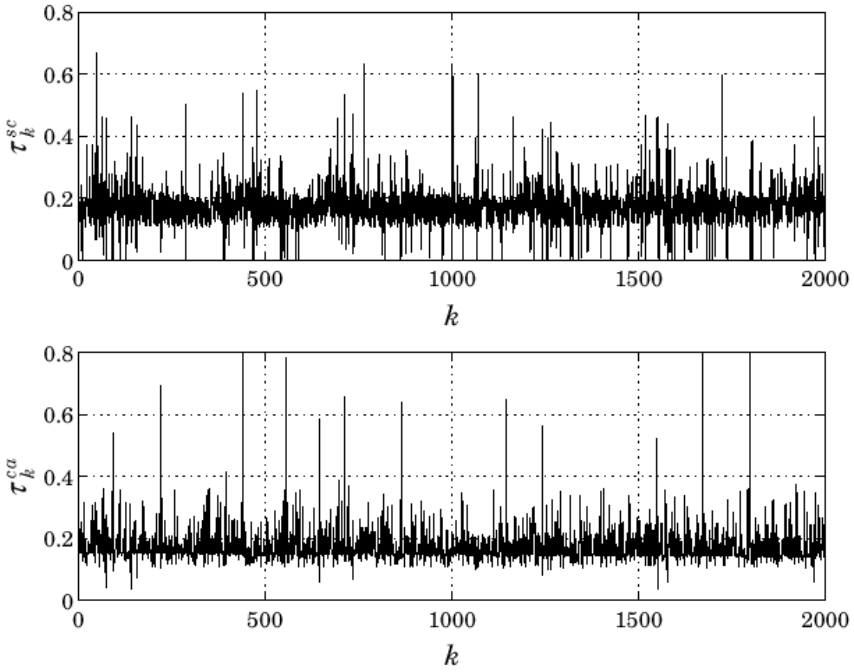


Figure 4.15 Delay measurements with two extra loads on the Ethernet network.

Two Points of View

- Computer Science
  - Scheduling principles
  - Resource allocation
  - What can be guaranteed?
- Control
  - Model the delays
  - Design of controllers
  - Robustness, stability, performance

Two design approaches

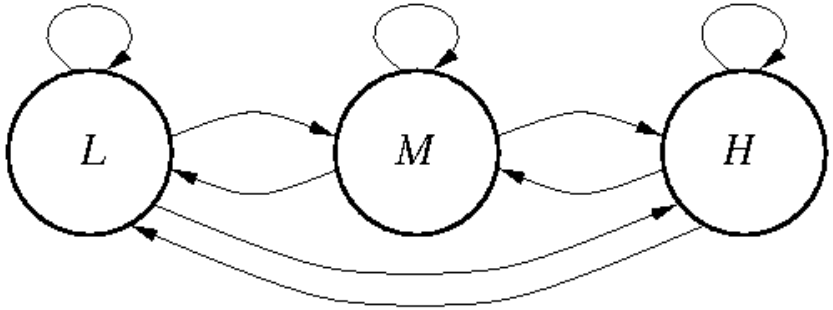
- Maximize temporal determinism
  - use protocols and scheduling techniques that maximize determinism
  - e.g. TTA/TTP
  - well suited for formal analysis, safety critical systems
  - matches sampled control theory well
  - non-COTS, requires complete knowledge
- Compensate for temporal non-determinism
  - inherent robustness of feedback
  - temporally robust off-line design methods
  - on-line compensation
    - need to measure delays, e.g. “time-stamping”
    - gain-scheduling, feedforward, ...

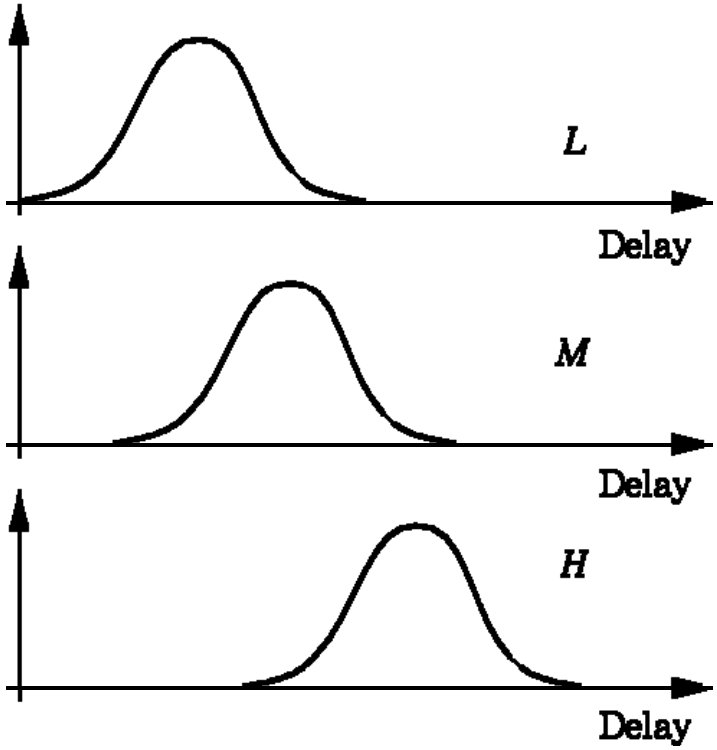
Outline

- Overview
- **Analysis and Design**
  - Constant network delays
  - Varying network delays
- Jitterbug
  - Stochastic analysis of networked control loops
- TrueTime
  - Simulation of networked control loops

Delay Models

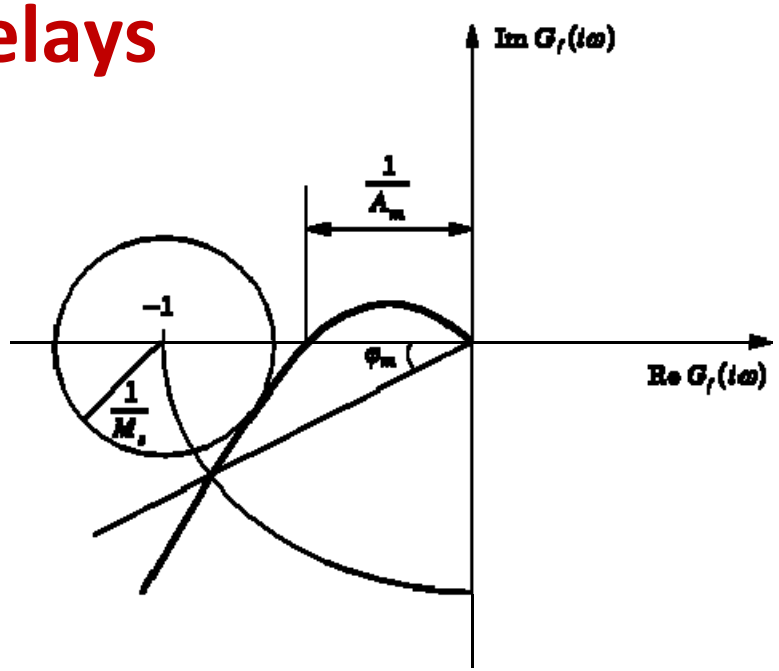
- Constant delay
- Random delay
  - independent from transfer to transfer
- Random delay
  - dependent
  - e.g. probability distribution governed by Markov chain
  - “Low load”, “Medium load”,





Constant Delays

- Straightforward
- Continuous time
  - e.g. Otto-Smith Controller
  - e.g. Predictive PI
- Discrete Time
  - sampling of a system with time delay



$$\frac{dx}{dt} = Ax(t) + Bu(t-\tau)$$

$$y(t) = Cx(t) + Du(t)$$

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

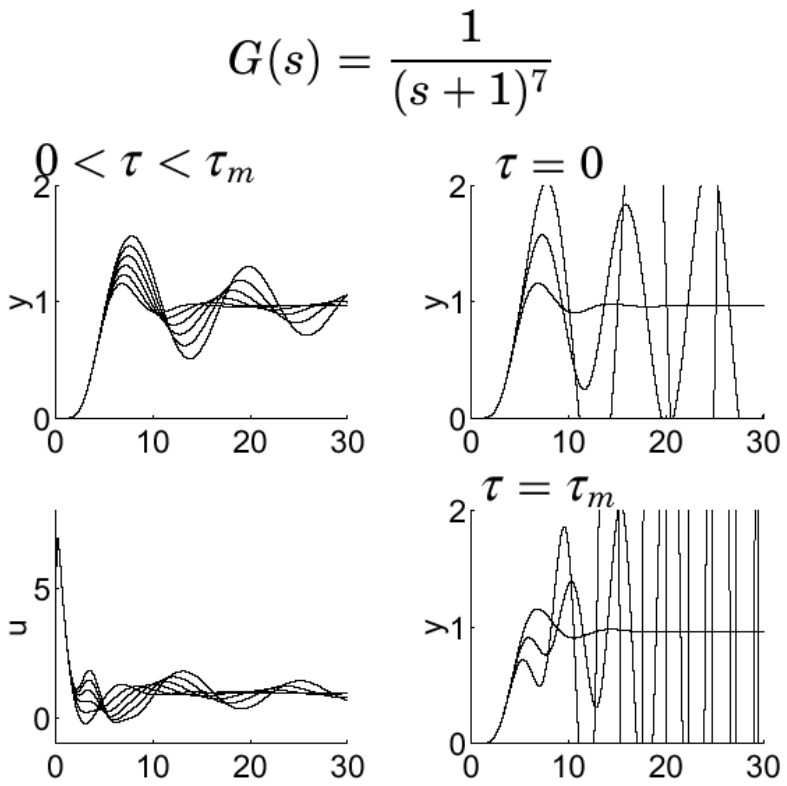
$$\Phi = e^{Ah}$$

$$\Gamma_0 = \int_0^{h-\tau} e^{As}\,ds\,B$$

$$\Gamma_1 = \int_{h-\tau}^{h} e^{As}\,ds\,B$$

Robust vs Worst-Case

- Left: Robust design taking the delay into account
- Top Right: Design for zero delay
- Bottom Right: Design for worst-case delay



Random Delays

- More tricky
- Time-varying system
- Examples can be found of systems that are stable for all constant delays, but become unstable when the delay varies
- It is important to be clear of what the available results cover:
  - constant delays within a certain range
  - varying delays within a certain range

Sampling of systems with varying delays

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

$$\Phi = e^{Ah}$$
$$\Gamma_0(\tau_k) = \int_0^{h-\tau_k} e^{As}\,ds\,B$$
$$\Gamma_1(\tau_k) = \int_{h-\tau_k}^{h} e^{As}\,ds\,B$$

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

- Closed Loop System (plant + controller)

$$z(kh+h) = \Phi_{cl}(\tau_k)z(kh) + \Gamma_{cl}(\tau_k)e(kh)$$

- Similar for sampling jitter

$$z_{k+1} = \Phi_{cl}(h_k, \tau_k)z_k + \Gamma_{cl}(\tau_k)e_k$$

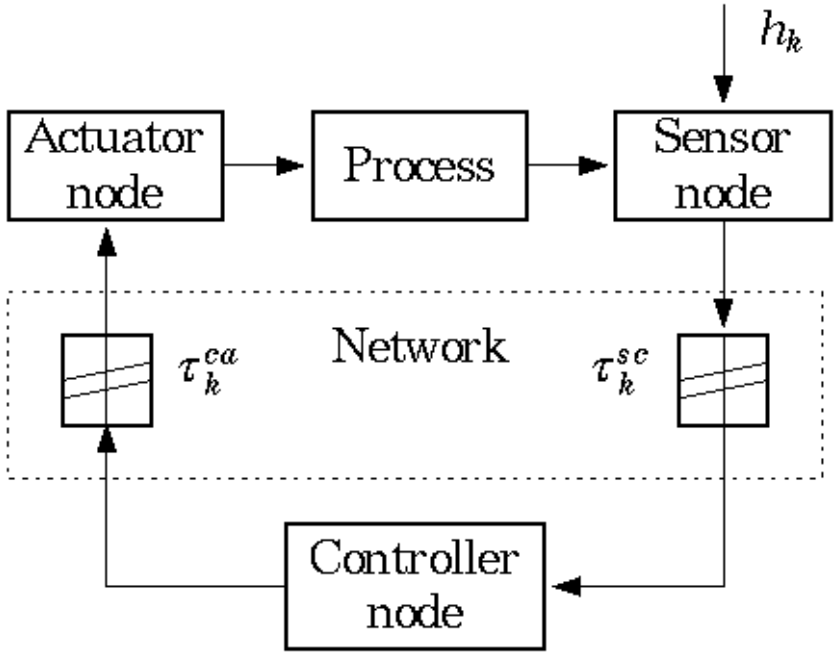
Stability

- Delays that change according to a finite, repeating cycle

$$\rho(\Phi_{cl}(\tau_1) \cdot \Phi_{cl}(\tau_2) \cdot \ldots \cdot \Phi_{cl}(\tau_n)) < 1$$

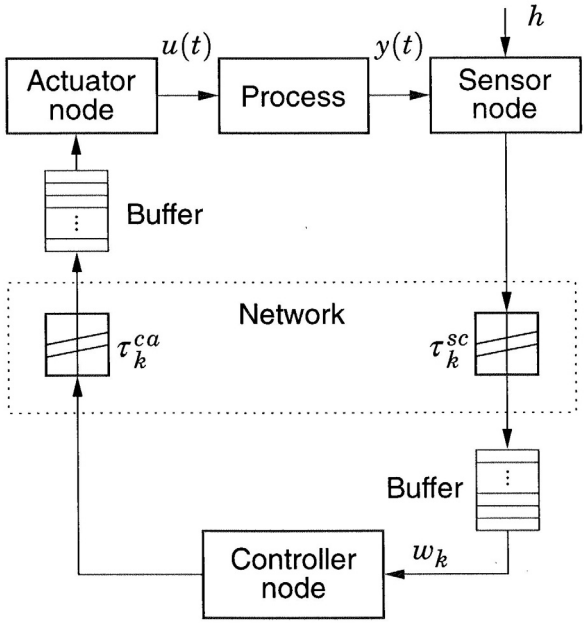
- Delays that change randomly
  - Lyapunov stability theory
  - Stable if we can find a common quadratic Lyapunov function for all delays

SISO Control - Basic Setup



- Computational Model:
  - different possibilities
  - our approach (Nilsson): time-driven sensor & event-driven controller and actuator

Alternative Approach



- Luck and Ray
- Make invariant through max-delay buffers
- Longer delays than necessary
- Almost always worse than having shorter, varying delays

LQG Control - Independent delays

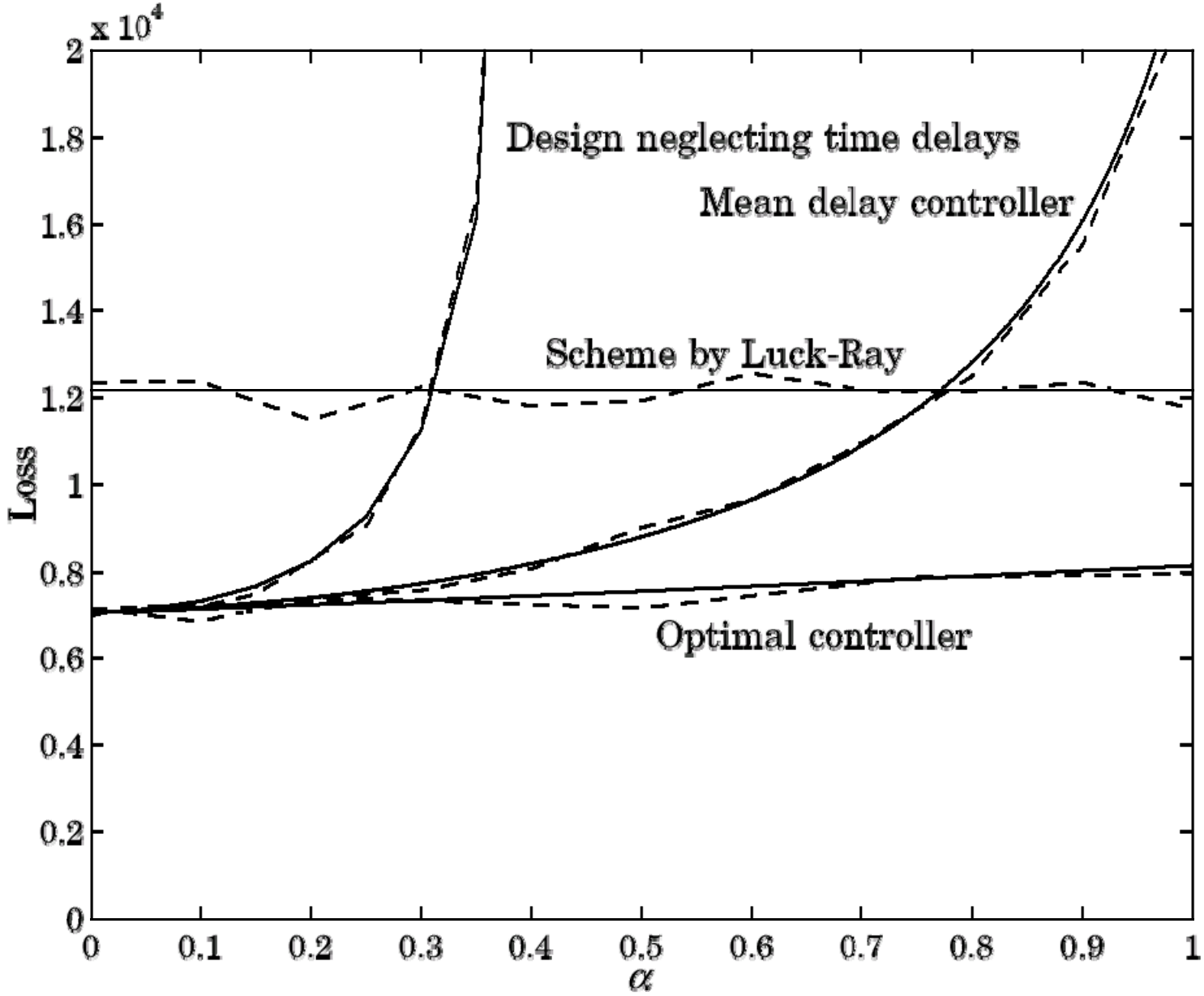
- Johan Nilsson – PhD thesis
  - "Real-Time Control Systems with Delays"
- Time stamping
- Old delays known when calculating u_k
  - sensor-controller delay up to k
  - controller-actuator delay up to k - 1
- Independent random delays with known distributions
- State feedback
  - full state information
  - all states from the same node in the same frame

LQG Control

$$J_N = \mathrm{E}\, x_N^T Q_N x_N + \mathrm{E} \sum_{k=0}^{N-1} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T Q \begin{bmatrix} x_k \\ u_k \end{bmatrix}$$

$$u_k = -L(\tau_k^{sc}, S_{k+1}) \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix}.$$

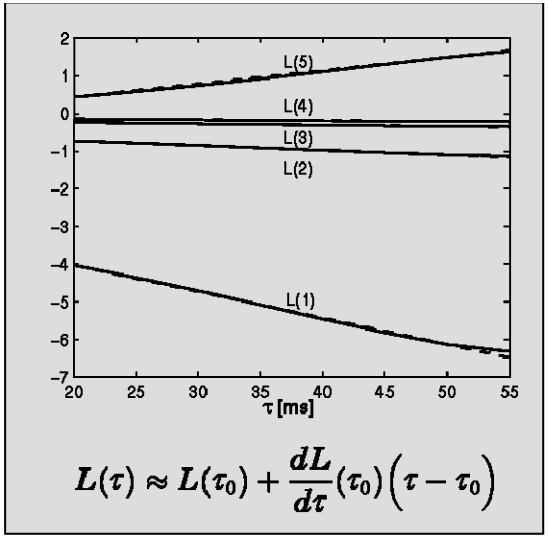
- Stochastic Riccati equation
- Updating of S not always possible in real-time


x 10^4
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Loss
Design neglecting time delays
Mean delay controller
Scheme by Luck-Ray
Optimal controller
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
α

Simplifications

- Off-line calculation of stationary Riccati
  - tabular for L
  - interpolation
  - linear approximation
- Suboptimal scheme
  - delay-free feedback vector
  - predict from time k over average delay

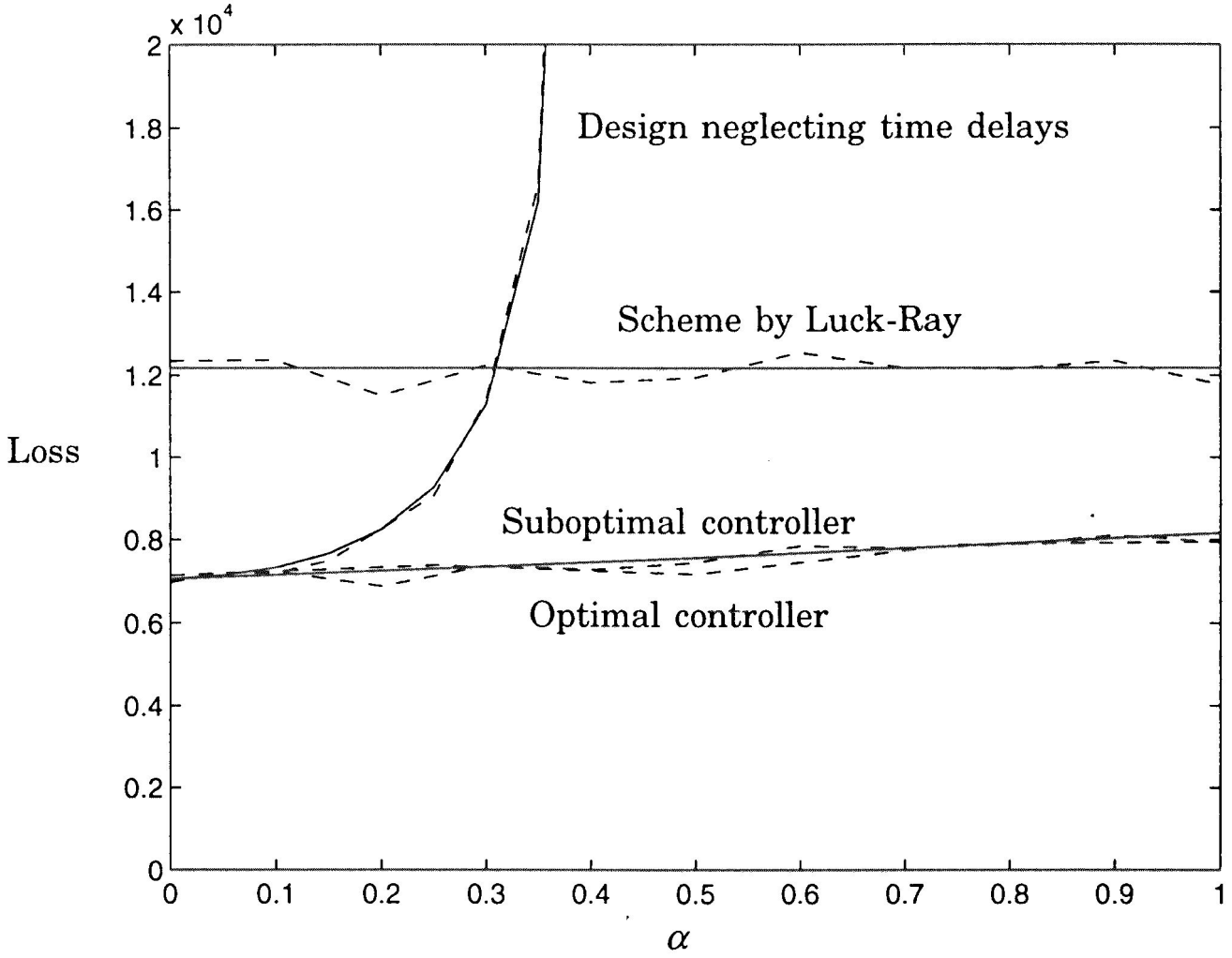
$$u_k = -L_k(\tau_k^{sc}) \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix}$$



$$L(\tau) \approx L(\tau_0) + \frac{dL}{d\tau}(\tau_0)\left(\tau - \tau_0\right)$$

$$u_k = -L \begin{bmatrix} \Phi_k^{pred} & \Gamma_k^{pred} \end{bmatrix} \begin{bmatrix} \hat{x}_{k|k} \\ u_{k-1} \end{bmatrix}, \quad \Phi_k^{pred} = e^{A(\tau_k^{sc} + \mathbf{E}\,\tau_k^{ca})},$$

$$\Gamma_k^{pred} = \int_0^{\tau_k^{sc} + \mathbf{E}\,\tau_k^{ca}} e^{As} ds B,$$


x 10^4
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Loss
Design neglecting time delays
Scheme by Luck-Ray
Suboptimal controller
Optimal controller
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
α

Extensions

- Optimal state estimator
- Optimal output feedback controller
  - separation principle holds

LQG Control - Dependent delays

- Delay distributions governed by Markov chain
- Optimal state feedback

$$u_k = -L_k(\tau_k^{sc}, r_k) \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix}$$

  - requires knowledge of the delay mode (Markov chain state)
- Optimal state estimate & output feedback

LQG Control: Extensions

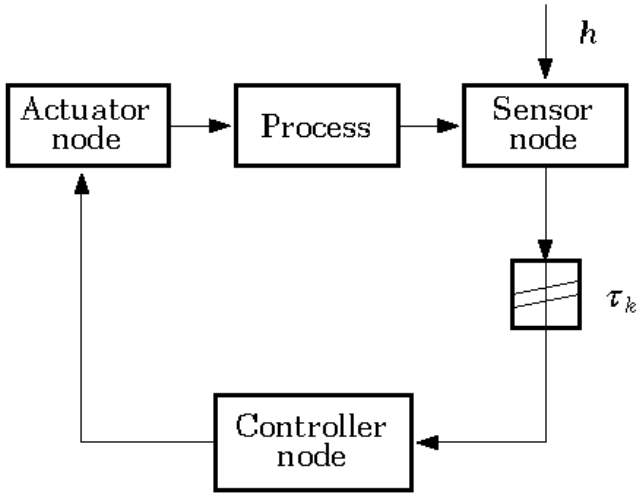
- Markov chain with two transitions every sample
  - different delay distributions for sender-controller message and controller-actuator message
- Sampling interval jitter in sensor node
  - optimal state feedback, estimator & output feedback
  - requires the solution of the Riccati in every sample
- Estimation of the Markov state

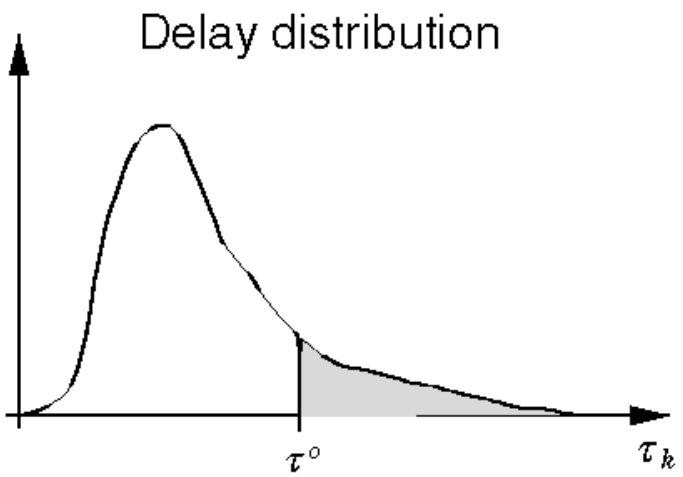
LQG Control: Extensions

- MIMO Control
  - multiple sensor and actuator nodes
  - time-driven sampling (synchronized clocks)
  - optimal state feedback and estimator results has been derived
  - based on the delay of the latest received sensor measurement

Timeout

- Can control performance be improved by having a timeout on sensor values?





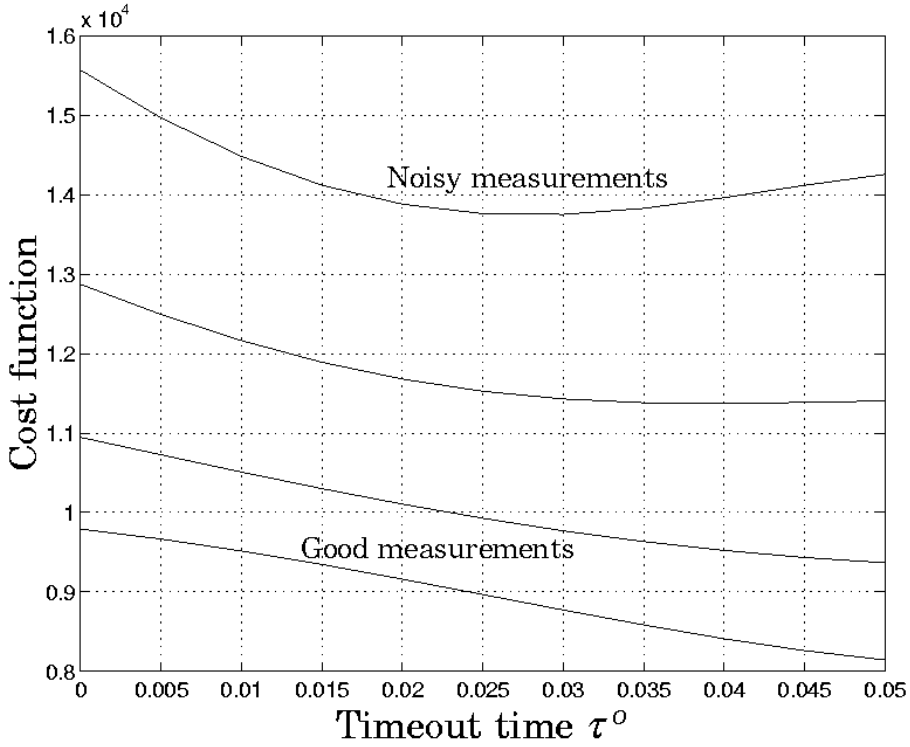
Timeout Control

- Prediction-Based Controller

$$u_k = \begin{cases} -L(\tau_k) \begin{bmatrix} \hat{x}_{k|k} \\ u_{k-1} \end{bmatrix} & \text{if } \tau_k < \tau^o, \\ -L(\tau^o) \begin{bmatrix} \hat{x}_{k|k-1} \\ u_{k-1} \end{bmatrix} & \text{if } \tau_k \geq \tau^o. \end{cases}$$

- Two versions:
  - Lost samples: The delayed samples will eventually arrive and can be used for updating the filters
  - Vacant samples: The delayed samples are lost

Timeout Control



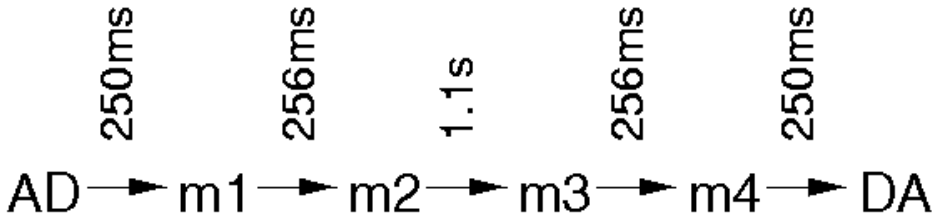
- 2nd order process, uniform delay on [0,h]
- LQG optimal control

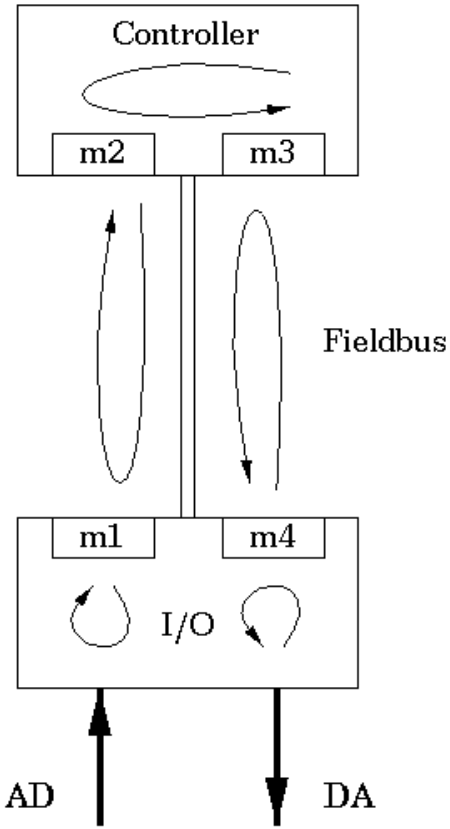
  Why wait for a noisy measurement?

Multi-rate Periodic Control

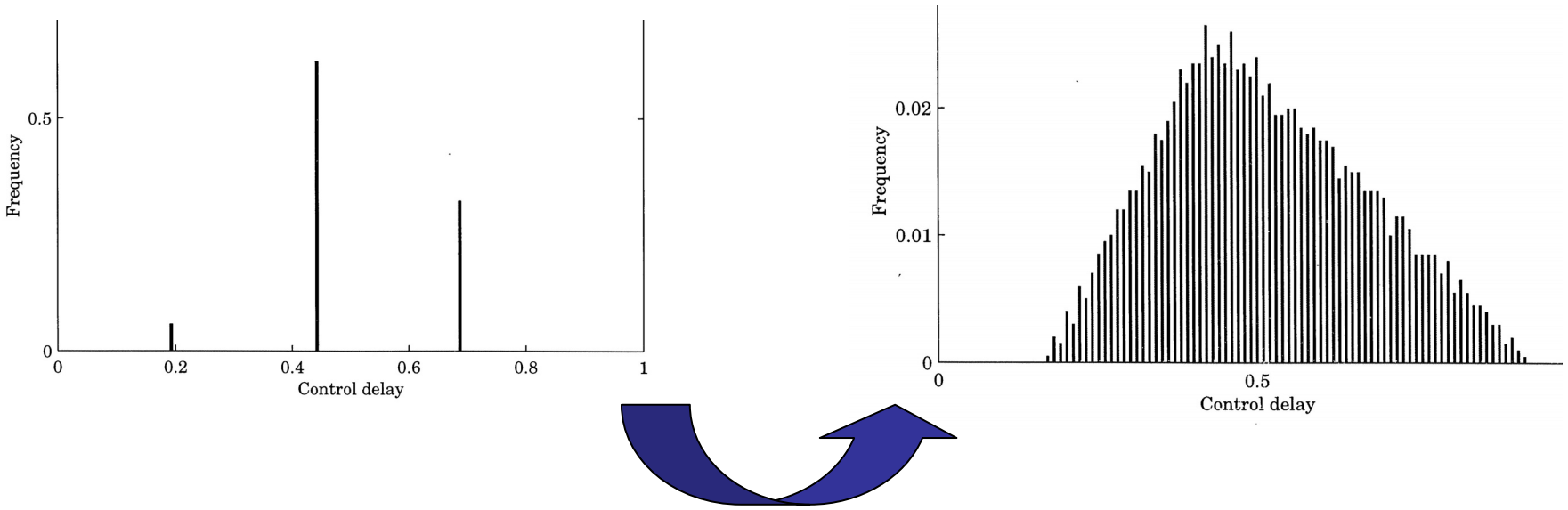
- Asynchronous periodic loops

Periodic mirroring of signals:





Interesting Delay Patterns



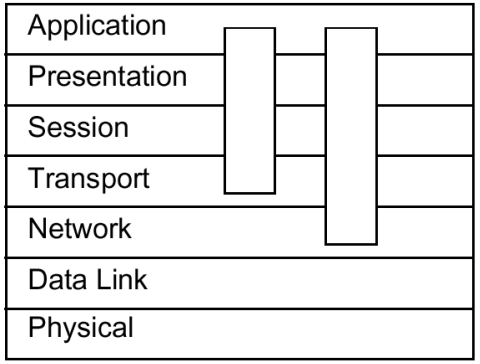
Small change in timing patterns

MIMO Controllers - Other approaches

- Strobe connection
  - time-driven controller multicasts a message telling the sensors to sample
- Poll connection
  - time-driven controller sends individual messages to each sensor node in turn, requesting them to sample

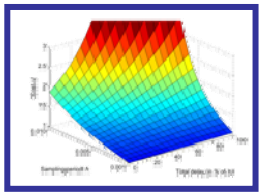
Collision Detection

- Opens up interesting possibilities
- Sensor nodes that detect collisions:
  - re-send old sample
  - discard sample
  - resample and send new sample …
  - increase sampling interval to reduce communication load - tradeoff
- Layered model inadequate



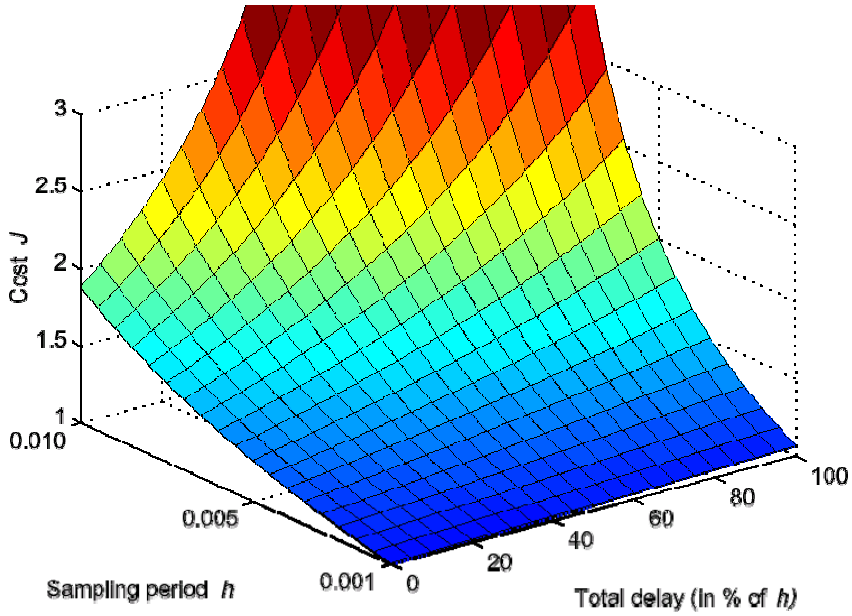
Outline

- Overview
- Analysis and Design
  - Constant network delays
  - Varying network delays
- **Jitterbug**
  - Stochastic analysis of networked control loops
- TrueTime
  - Simulation of networked control loops

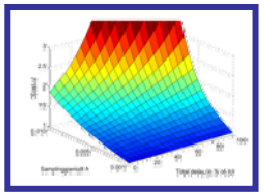


Jitterbug

- 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



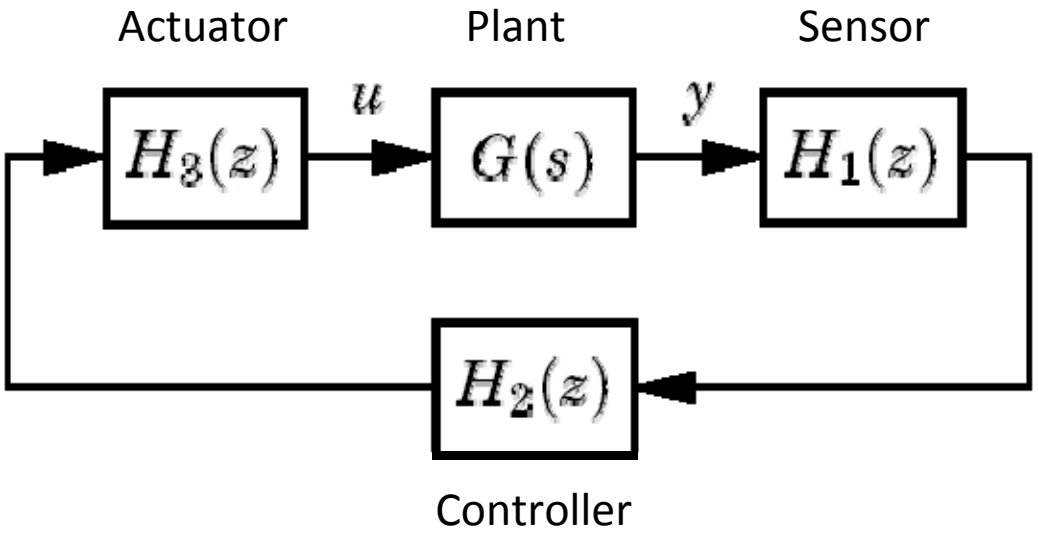
Developed by Bo Lincoln and Anton Cervin

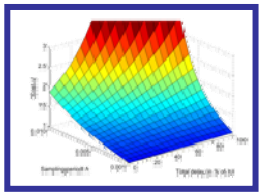


Jitterbug Analysis

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

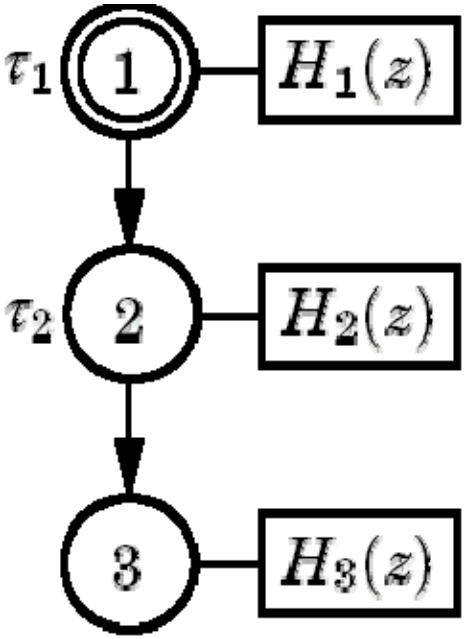
Distributed Control Loop:

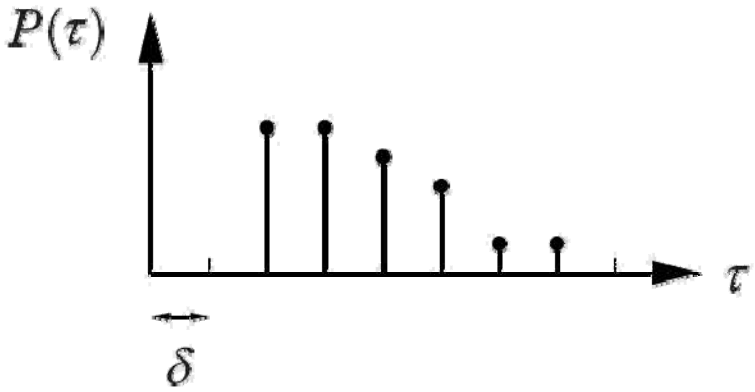


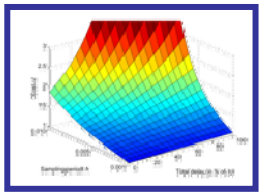


Jitterbug Analysis

- 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

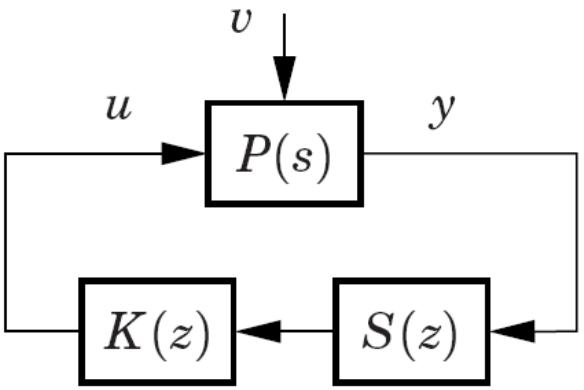




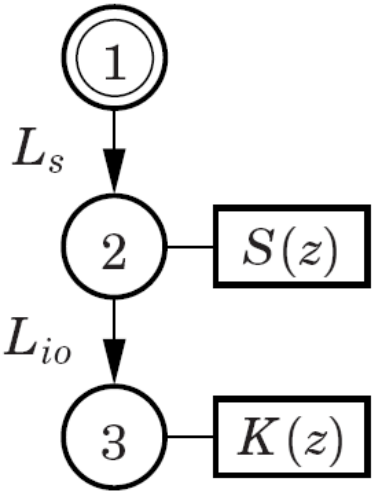


Jitterbug Model - Example

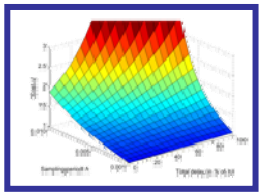
Signal model:



Timing model:

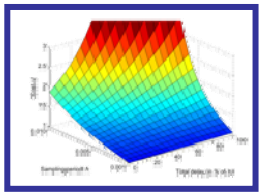


- $P(s)$ – process
- $S(z)$ – sampler
- $K(z)$ – controller/actuator



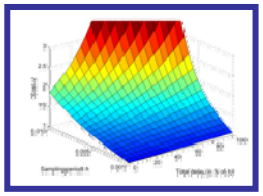
Jitterbug Example Script

```
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
```

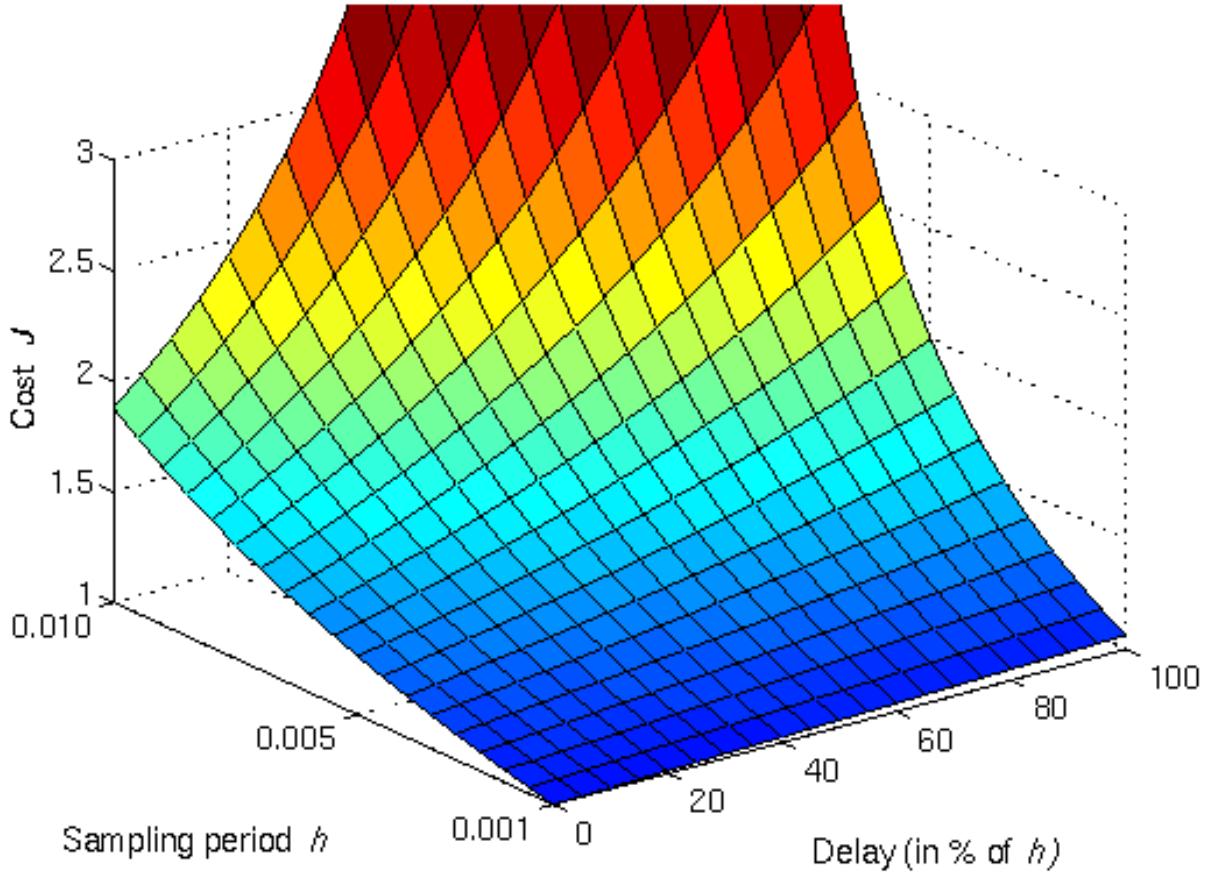


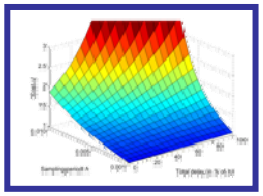
Jitterbug example script

```
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
```

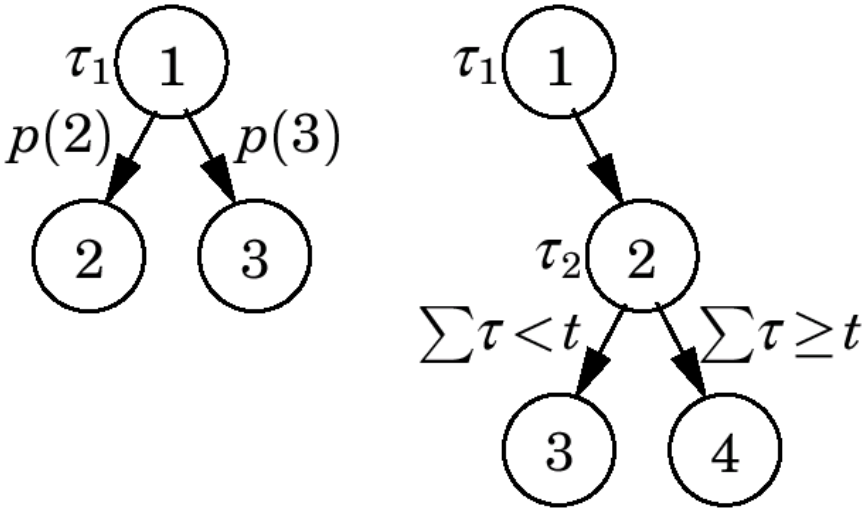


Results

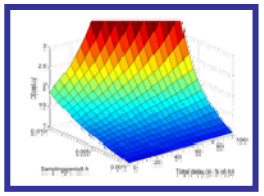

Cost J
3
2.5
2
1.5
1
0.010
0.005
0.001
Sampling period h
0
20
40
60
80
100
Delay (in % of h)



More complicated cases

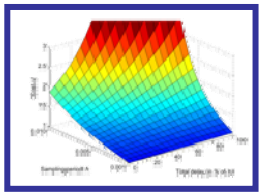


- random choice of path
- choice of path depending on delay
- different update equations in different nodes
- aperiodic systems
- . . .



Aperiodic Systems

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



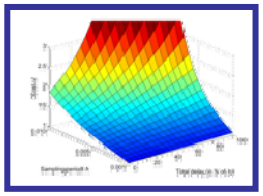
Pros and cons

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
- Only linear systems and quadratic costs
- Requires knowledge about latency distributions



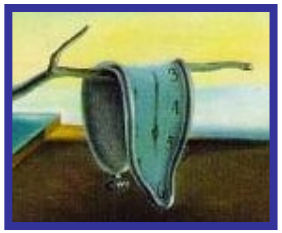
Jitterbug Availability

Jitterbug 1.2 can be downloaded from

`http://www.control.lth.se/user/lincoln/jitterbug/`

Outline

- Overview
- Analysis and Design
  - Constant network delays
  - Varying network delays
- Jitterbug
  - Stochastic analysis of networked control loops
- **TrueTime**
  - Simulation of networked control loops

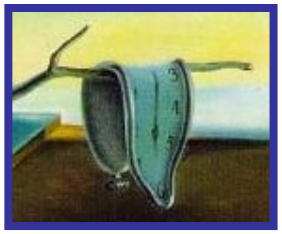


TrueTime Main Idea

Co-simulation of controller task execution, network transmissions, and continuous plant dynamics.

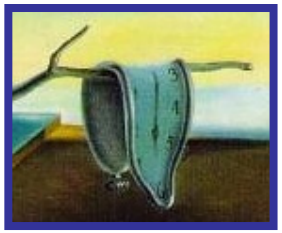
- Accomplished by providing models of real-time kernels and networks as Simulink blocks
- User code in the form of tasks and interrupt handlers is modeled by MATLAB or C-code

Developed by Anton Cervin, Dan Henriksson, Martin Ohlin, Johan Eker

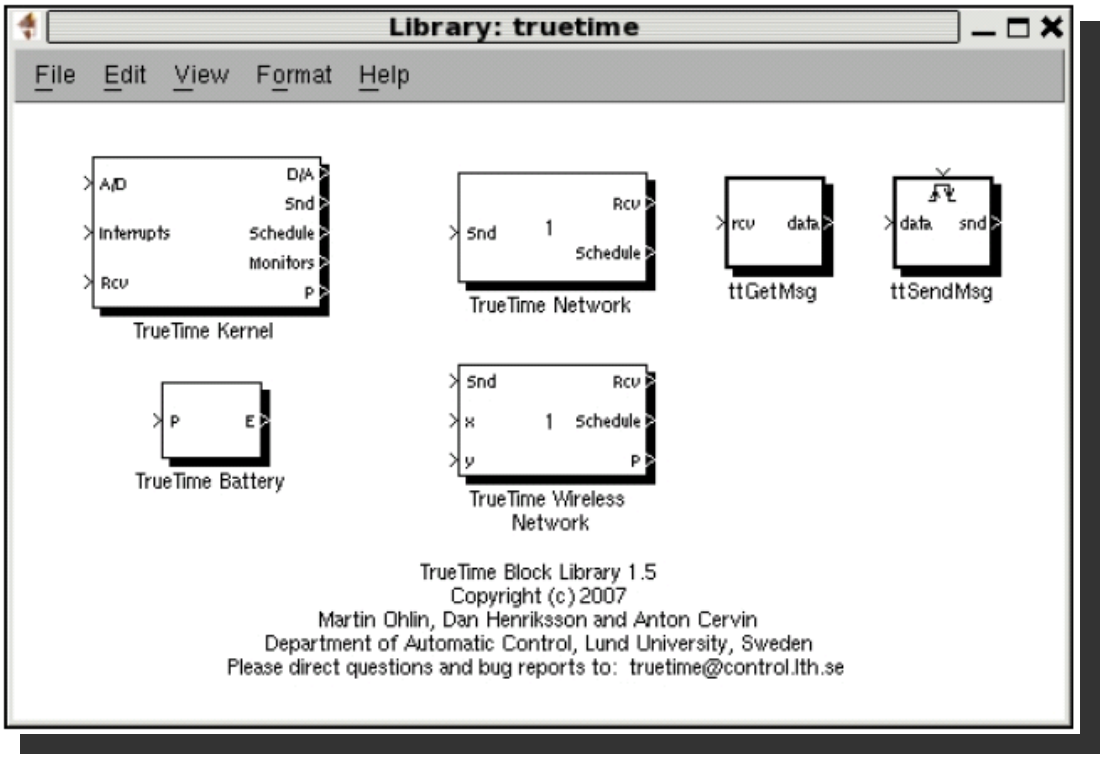


TrueTime Possibilities

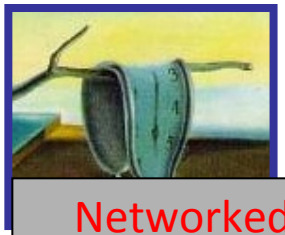
- Investigate the True Timely behavior of time- or event-triggered control loops subject to sampling jitter, input-output latency and jitter, and lost samples, caused by real-time computing and communication
- Evaluate different scheduling methods
- Evaluate the performance of different wired or wireless MAC protocols
- Simulate scenarios involving batter-powered mobile vehicles communicating using ad hoc wireless networks



TrueTime Simulink Blocks



- Offers a *Kernel* block, two *Network* blocks, and a *Battery* block
  - Simulink S-functions written in C++
  - Event-based implementation



Screen Dump

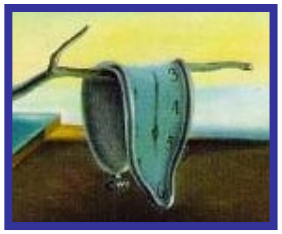
Networked Control Loop

CPU Schedule

Control Signal

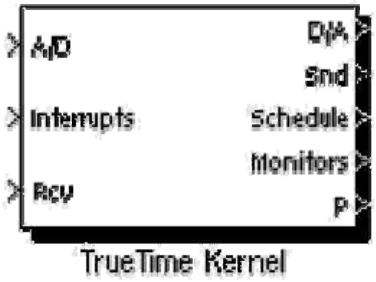
Network Schedule

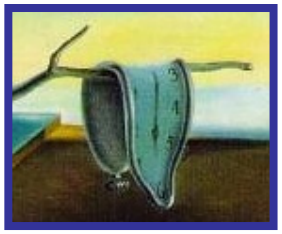
Step Response



The Kernel Block

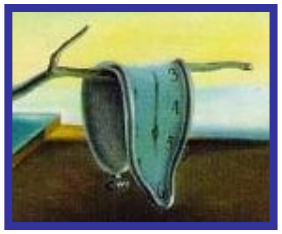
- Simulates an event-based real-time kernel
- Executes user-defined tasks and interrupt handlers
- Arbitrary user-defined scheduling policy
- Supports external interrupts and timers
- Supports common real-time primitives (sleepUntil, wait/notify, setPriority, etc.)
- Generates a task activation graph
- More features: context switches, overrun handlers, task synchronization, data logging





Kernel Implementation Details

- TrueTime implements a complete real-time kernel
  - ready queue
  - time queue
  - waiting queues associated with monitors and events
- The simulated kernel is ideal
  - no interrupt latency
  - context switch overhead may be defined

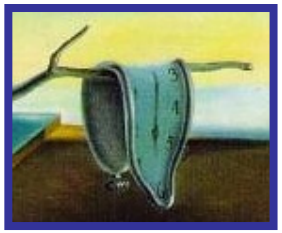


TrueTime Code

Three choices:

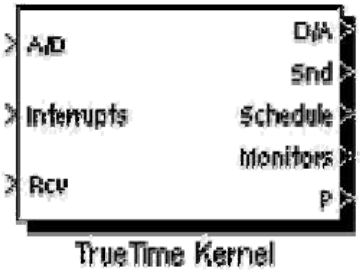
- C++ code (fast)
- MATLAB code (medium)
- Simulink block diagram (slow)

The TrueTime code is a model of the real code

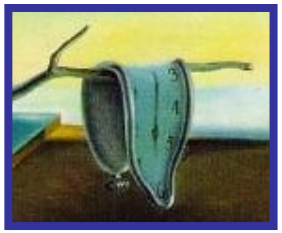


Interrupt Handlers

- Code executed in response to interrupts
- Scheduled on a higher priority level than tasks
- Available interrupt types
  - Timers (periodic or one-shot)
  - External (hardware) interrupts
  - Task overruns
  - Network interface

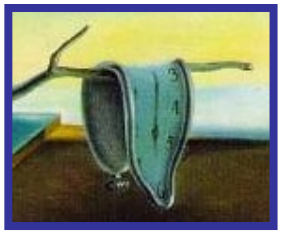



```
ttCreateInterruptHandler(hdlname, priority, codeFcn, data)

ttCreateTimer(timername, time, hdlname)
ttCreatePeriodicTimer(timername, start, period, hdlname)
ttCreateExternalTrigger(hdlname, latency)
```



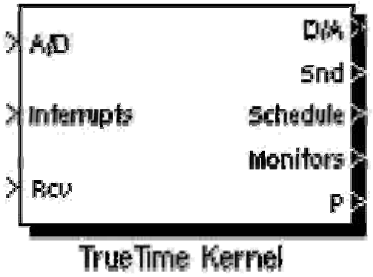
Task Communication

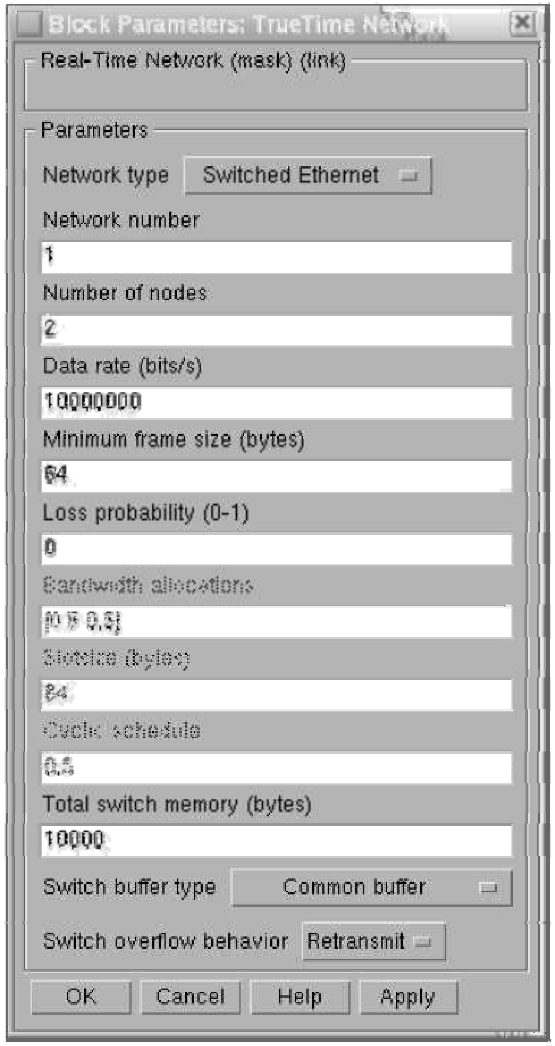
- *Monitors* with priority-sorted waiting queues are used to model mutual exclusion between tasks that share common resources (e.g., data)
- *Events* are used for task synchronization and may be free or associated with a monitor (condition variables)
- *Mailboxes* for asynchronous message passing between tasks

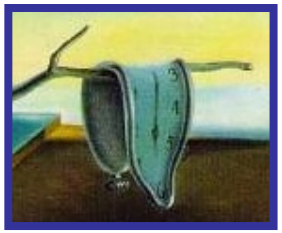


The Wired Network Block

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



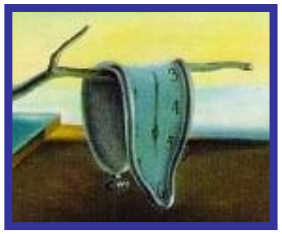




Network Communication

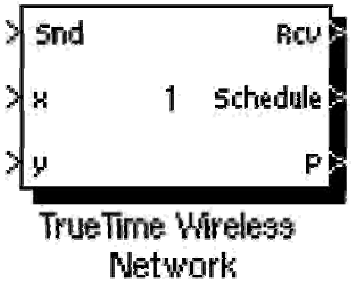
- Each node (kernel block) may be connected to several network blocks
- Dedicated interrupt handler associated with each network receive channel
  - Triggered as a packet arrives
  - Similar to external interrupts
- The actual message data can be an arbitrary MATLAB variable (struct, cell array, etc)
- Broadcast of messages by specifying receiver number 0

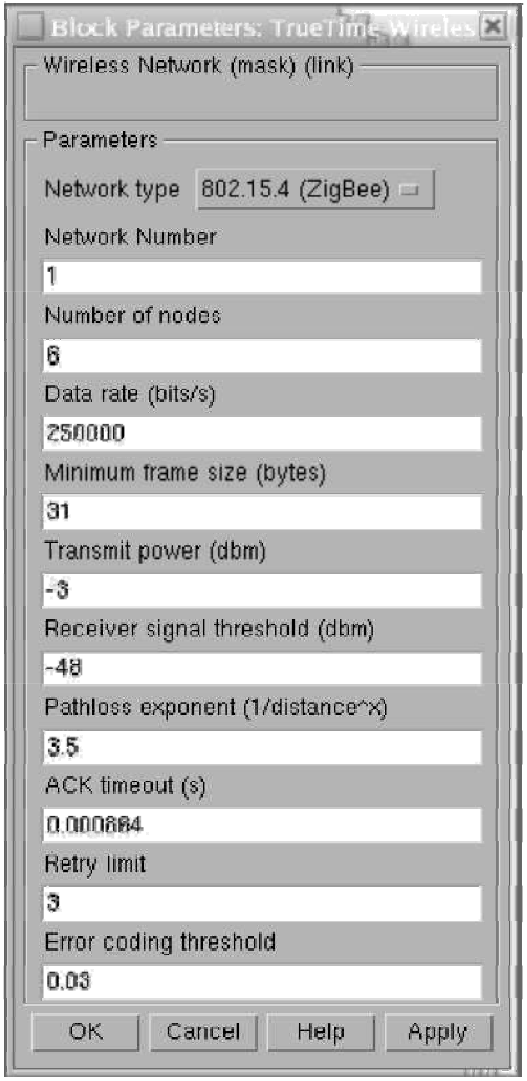
```
ttInitNetwork(network, nodenumber, hdlname)

ttSendMsg([network receiver], data, length, priority)
ttGetMsg(network)
```

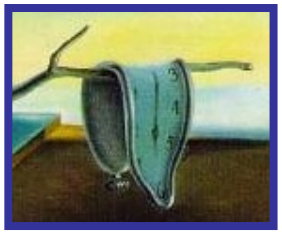


The Wireless Network Block

- Used in 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

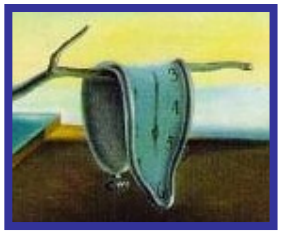






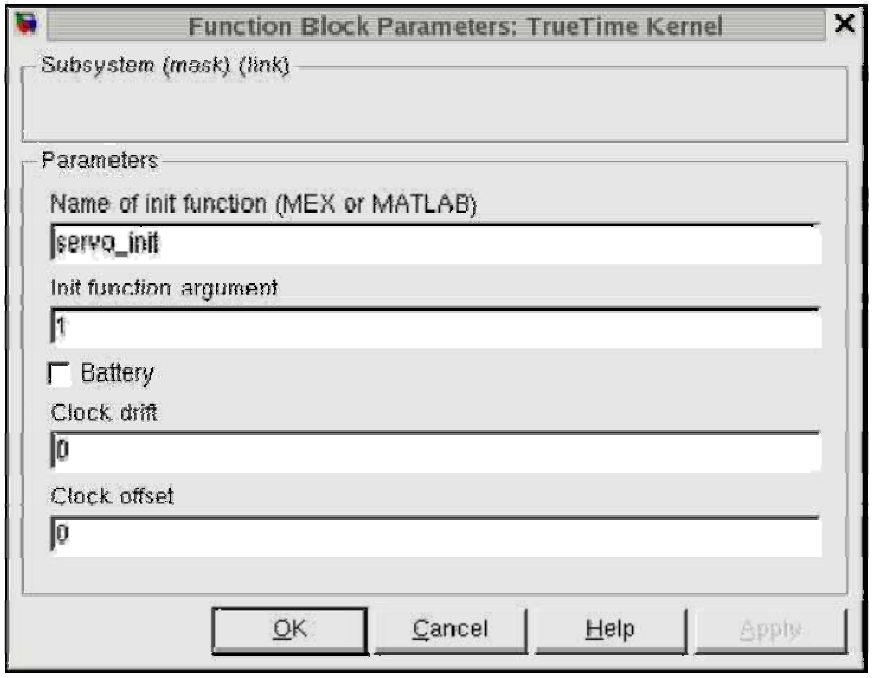
The Battery Block

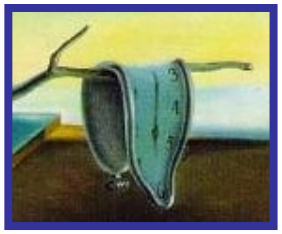
- Simulation of battery-powered devices
- Simple integrator model
  - discharged or charged (energy scaffolding)
- Energy sinks:
  - computations, radio transmissions, usage of sensors and actuators, . . .
- Dynamic Voltage Scaling
  - change kernel CPU speed to consume less power



Clocks with Offset and Drift

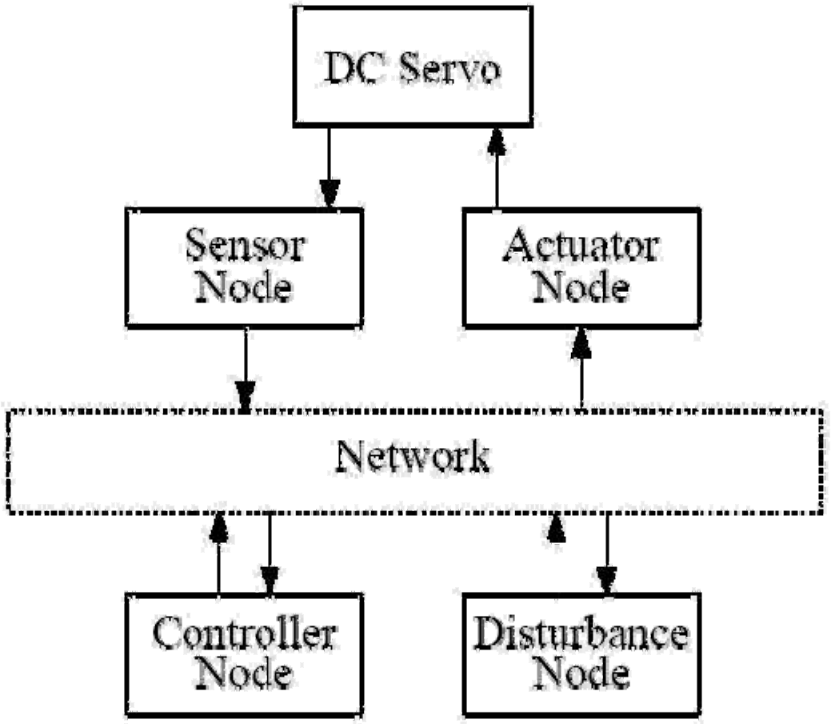
- To simulate distributed systems with local time
- Simulate clock synchronization protocols

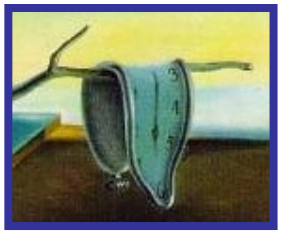




Networked Control Example

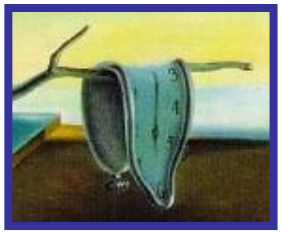
- Time-driven sensor node
- Event-driven controller node
- Event-driven actuator node
- High priority interference task in the controller node
- Disturbance node generating high priority network traffic





TrueTime Possibilities

- Co-Simulation of:
  - computations inside the nodes
    - tasks, interrupt handlers, scheduling hooks
  - wired or wireless communication between nodes
  - sensor and actuator dynamics
  - mobile robot dynamics
  - dynamics of the environment
  - dynamics of the physical plant under control
  - the batteries in the nodes
- Control performance assessment
  - time domain
  - cost function calculations



TrueTime Limitations

- Developed as a research tool rather than as a tool for system developers
- Cannot express tasks and interrupt handler directly using production code
  - code is modeled using TrueTime MATLAB code or TrueTime C code
  - no automatic translation
- Execution times or distributions assumed to be known
- How to support automatic code generation from TrueTime models?
  - Generate POSIX-thread compatible code?
  - Generate monolithic code (TVM = TrueTime Virtual Machine)
- Based on MATLAB/Simulink

Sensor Network & MANET Simulations with TrueTime

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

Localization Overview

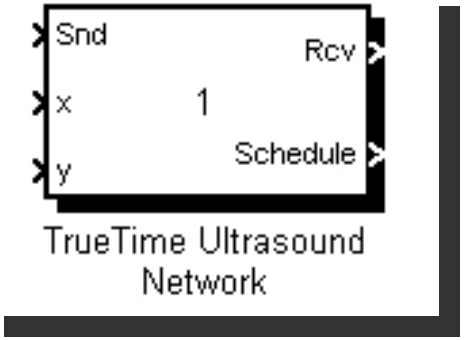
- Ultrasound-based
  - Active mobile robots
  - Passive stationary nodes
- Robot broadcasts radio packet and ultrasound pulse ”simultaneously”
- Difference in time-of-arrival allows each reachable node to calculate its distance to the robot
- After a predefined interval each node sends its distance measurement back to the robot

Localization Overview cont.

- The robot combines together the distance measurements using an Extended Kalman Filter (EKF)
  - In the update part of the EKF
- The robot combines the distance mesurements with the measured wheel positions (dead reckoning)
  - In the predictor part of the EKF
- The EKF provides estimates of the
  - x & y positions
  - orientation
- Navigator and Robot Controller for the control of the robot

TrueTime Extensions

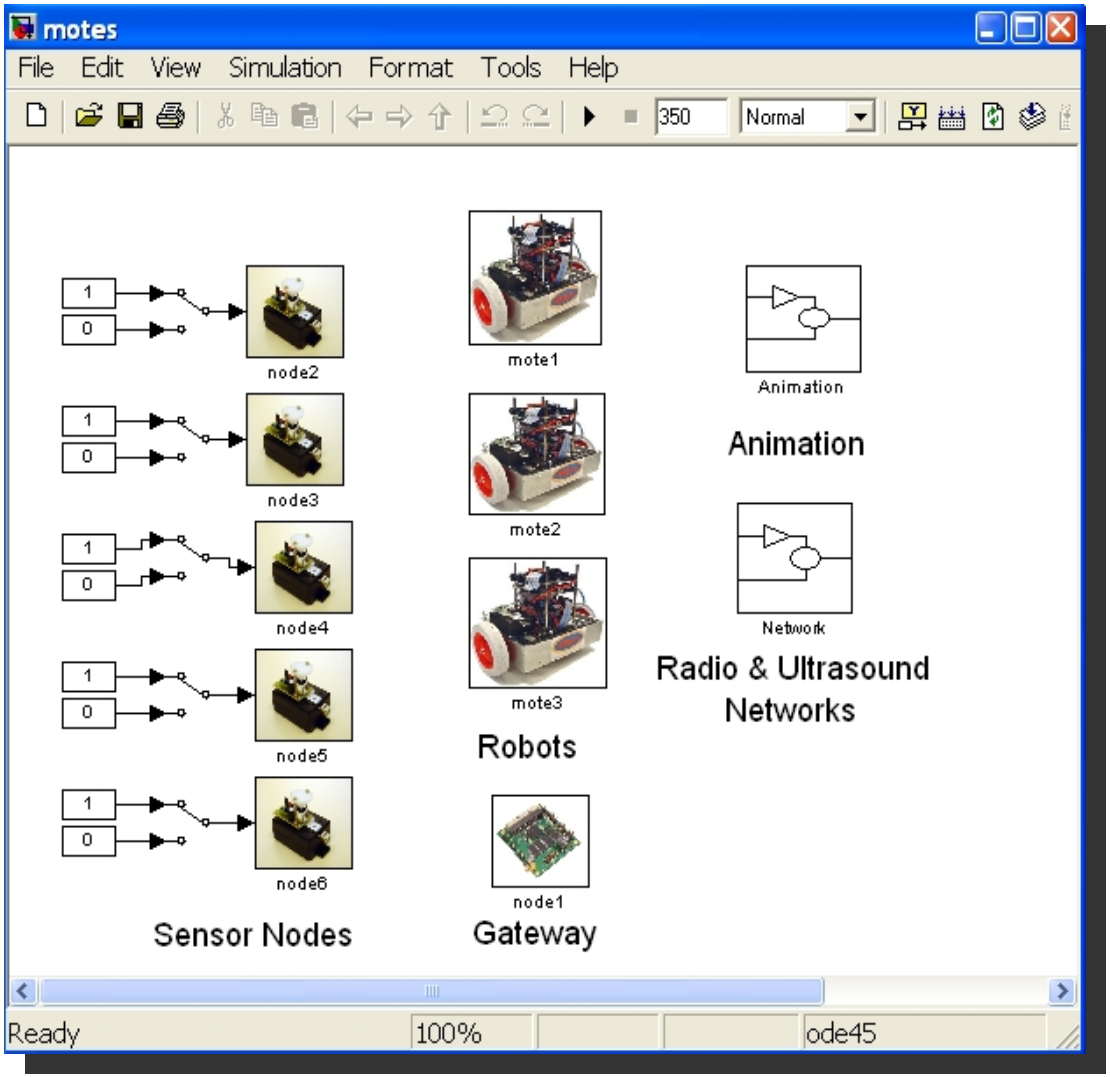
- Ultrasound Block
  - A version of the wireless network block that models the propagation delay of ultrasound in air



- AODV - Ad-hoc On-Demand Distance Vector Routing
  - One of the more popular ad hoc routing algorithms used in sensor networks and MANETs

The RUNES Tunnel Scenario Model

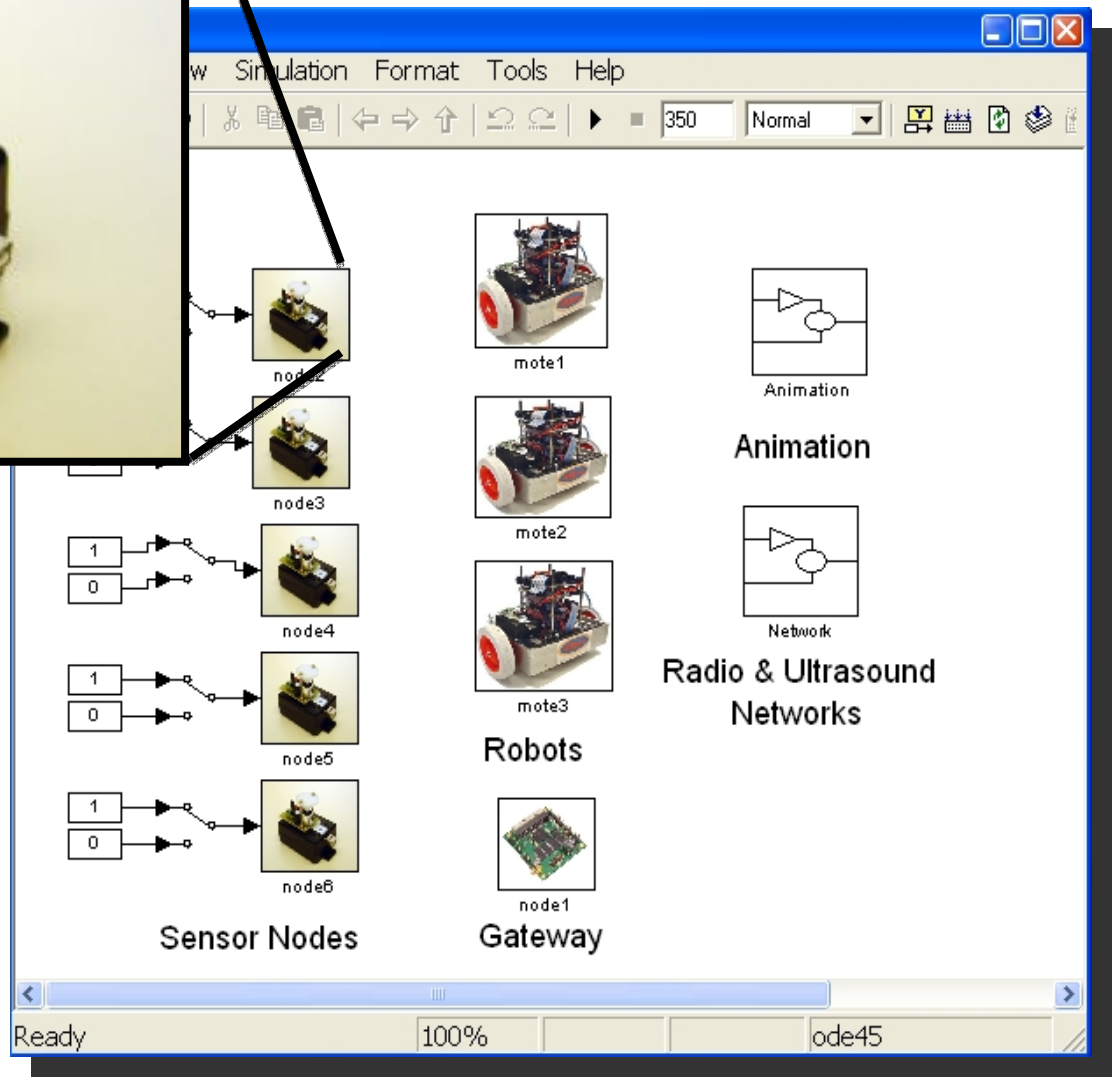
- Six sensor nodes
  - one being the gateway
  - turned on and off
- Three robots
- Radio & Ultrasound networks
- Animation



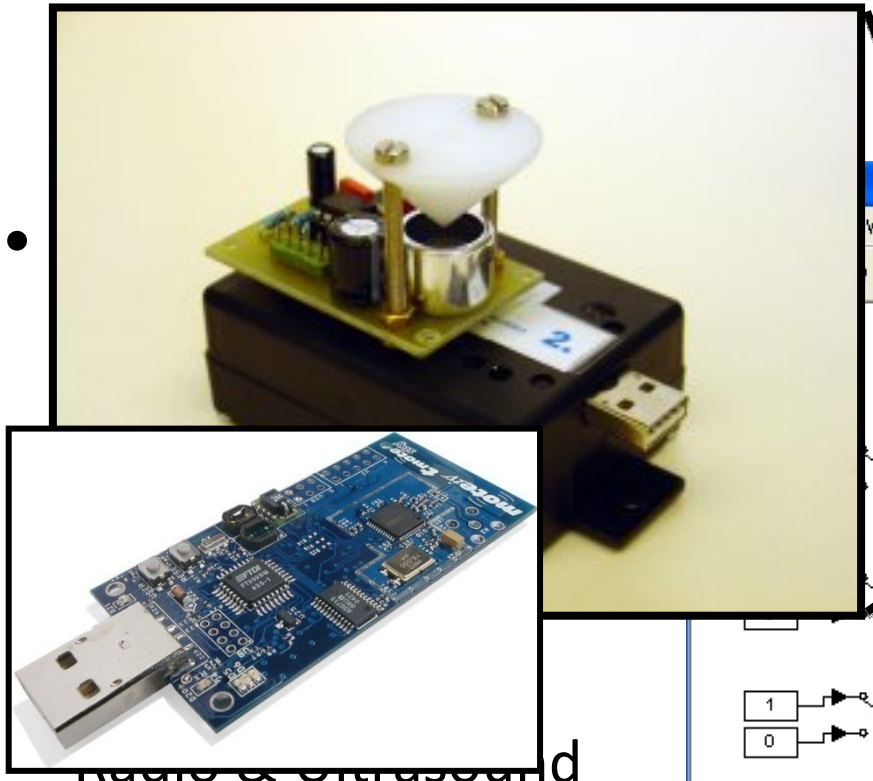
The RUNES Tunnel Scenario Model

- 

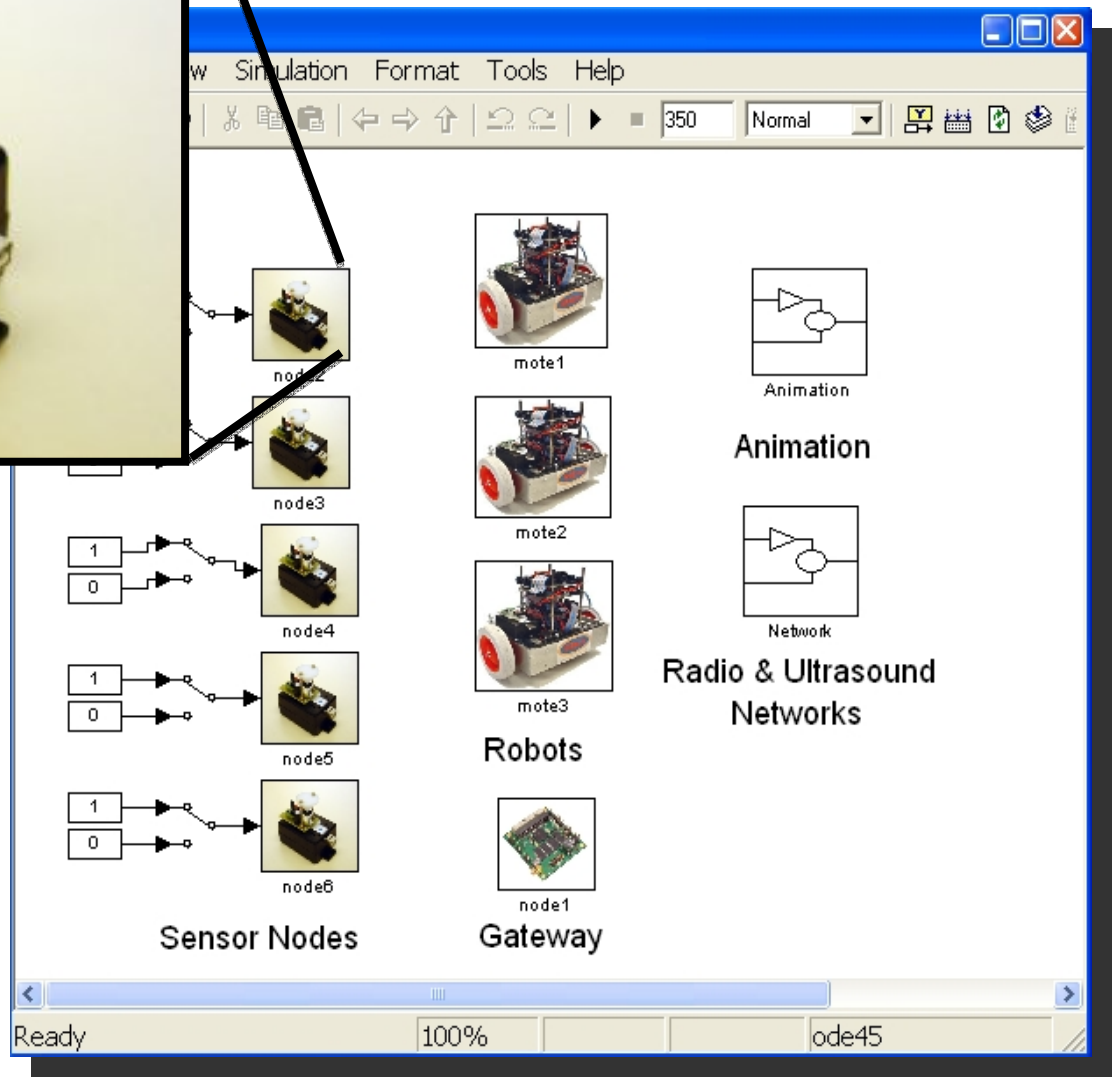
- Three robots
- Radio & Ultrasound networks
- Animation



The RUNES Tunnel Scenario Model

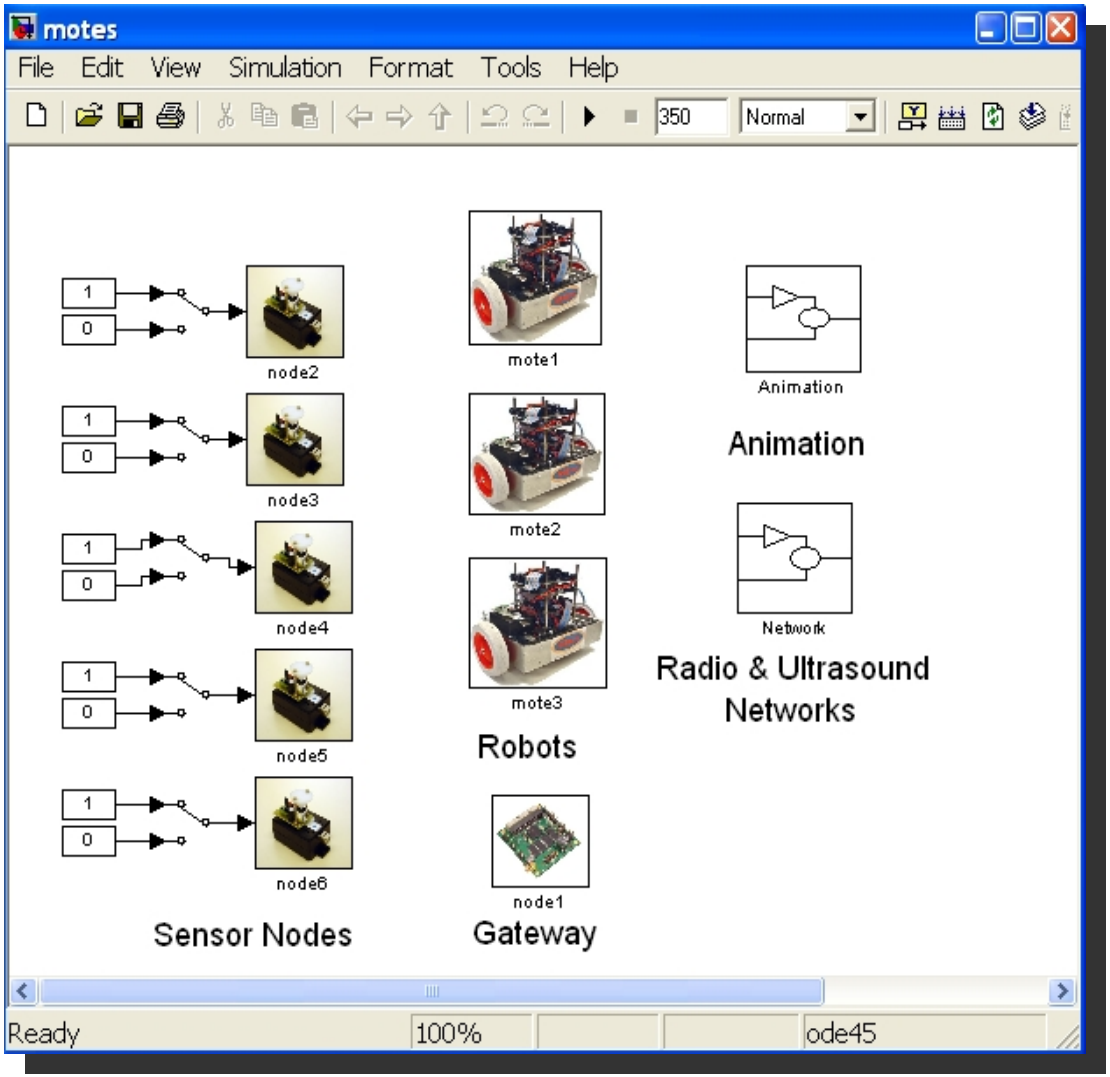


- Radio & Ultrasound networks
- Animation

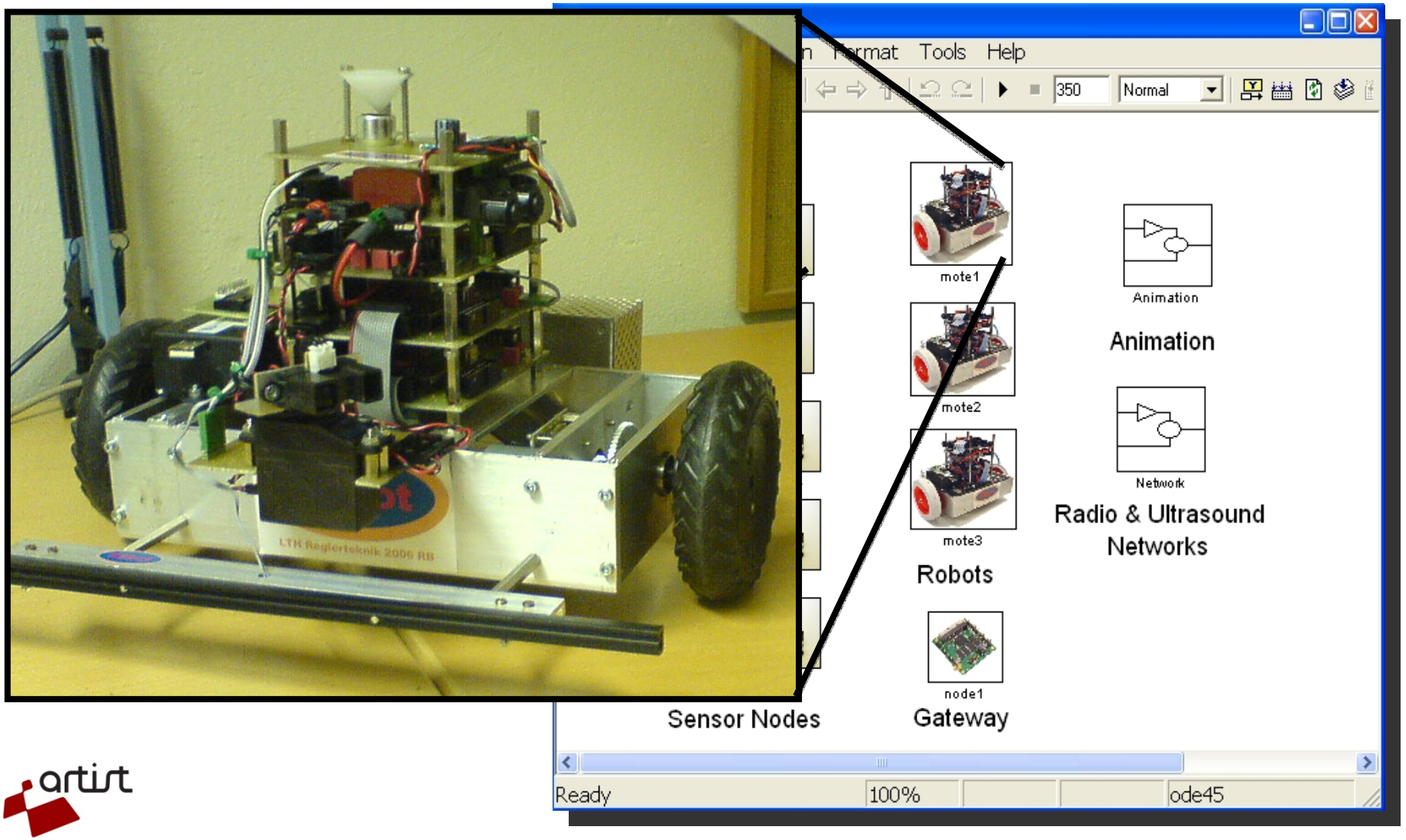


The RUNES Tunnel Scenario Model

- Six sensor nodes
  - one being the gateway
  - turned on and off
- Three robots
- Radio & Ultrasound networks
- Animation

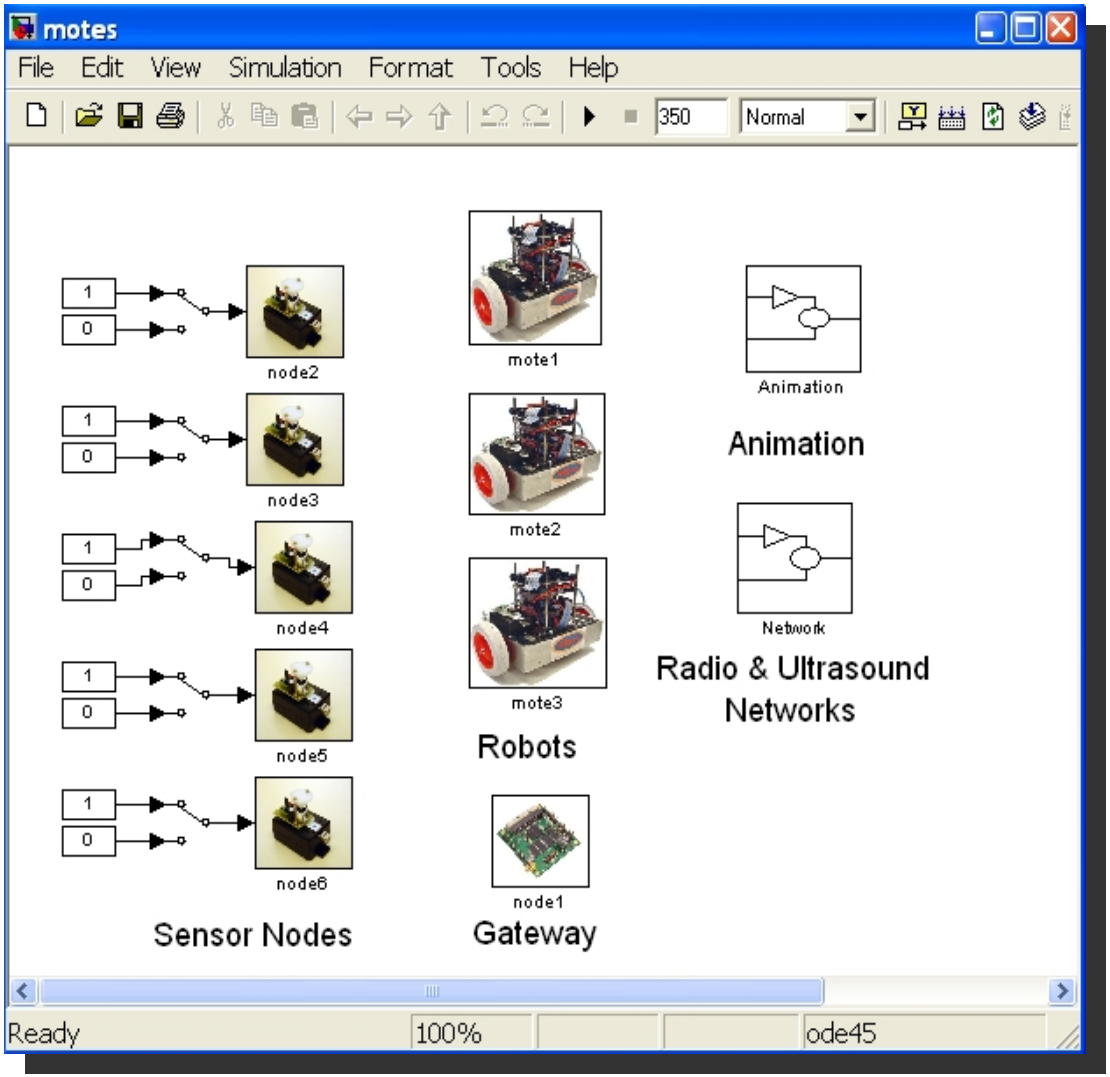


The RUNES Tunnel Scenario Model



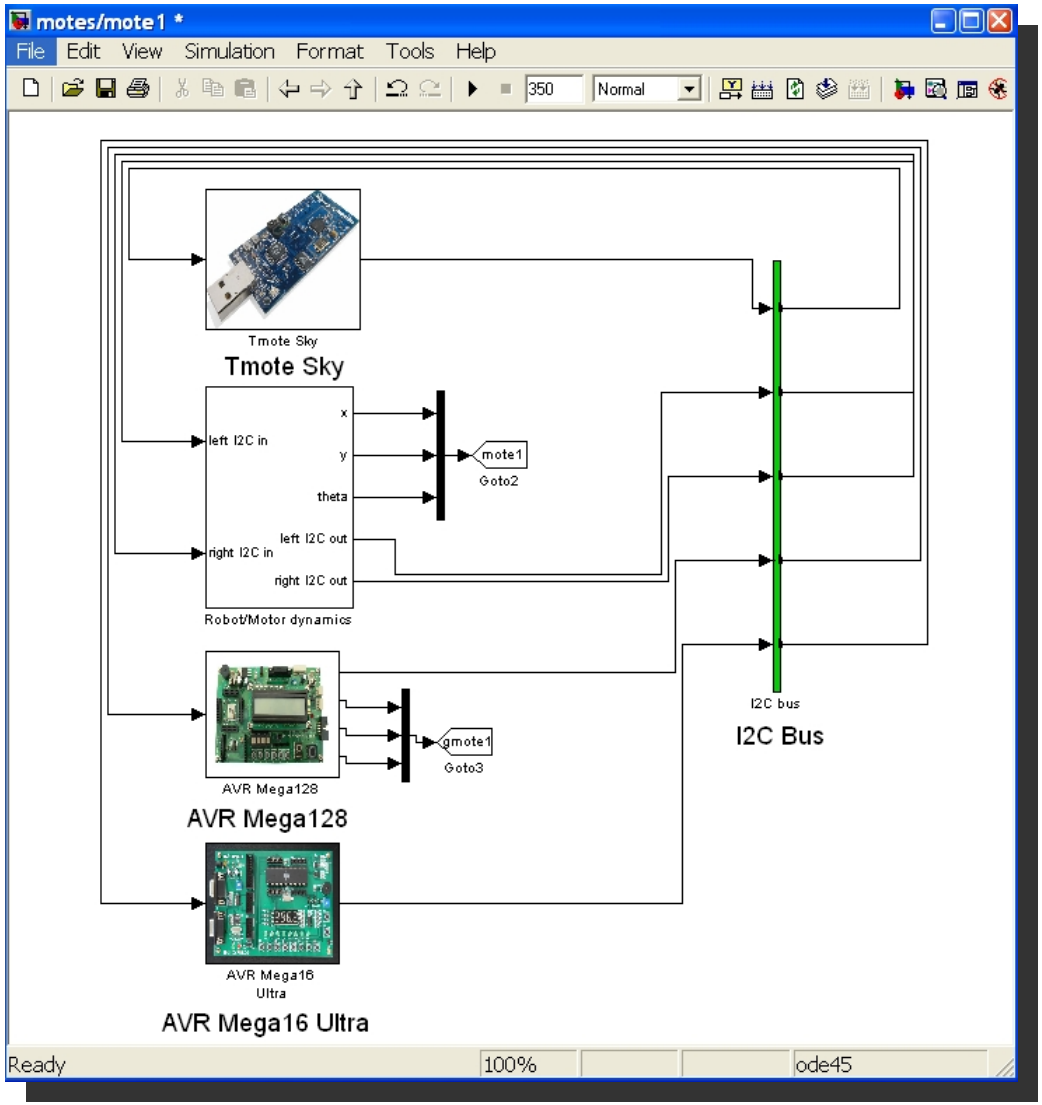
The RUNES Tunnel Scenario Model

- Six sensor nodes
  - one being the gateway
  - turned on and off
- Three robots
- Radio & Ultrasound networks
- Animation

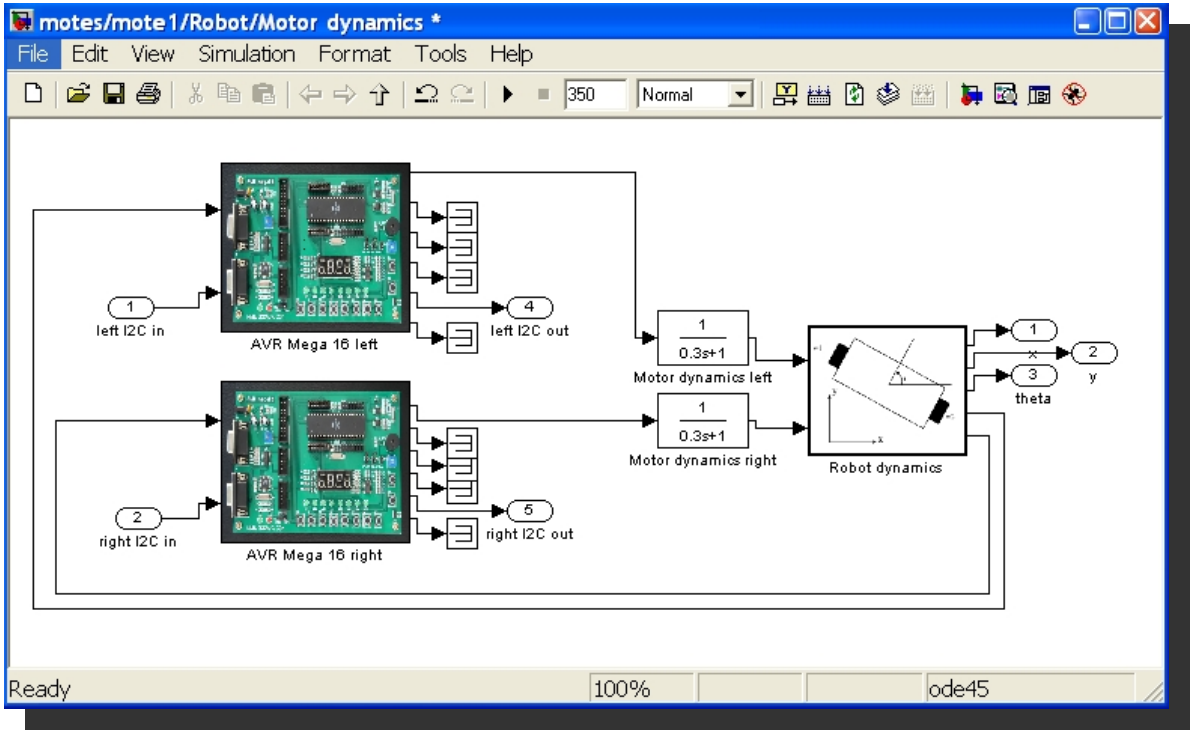


Robot Submodel

- 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 modeled by CAN
- Wheel and motor submodel

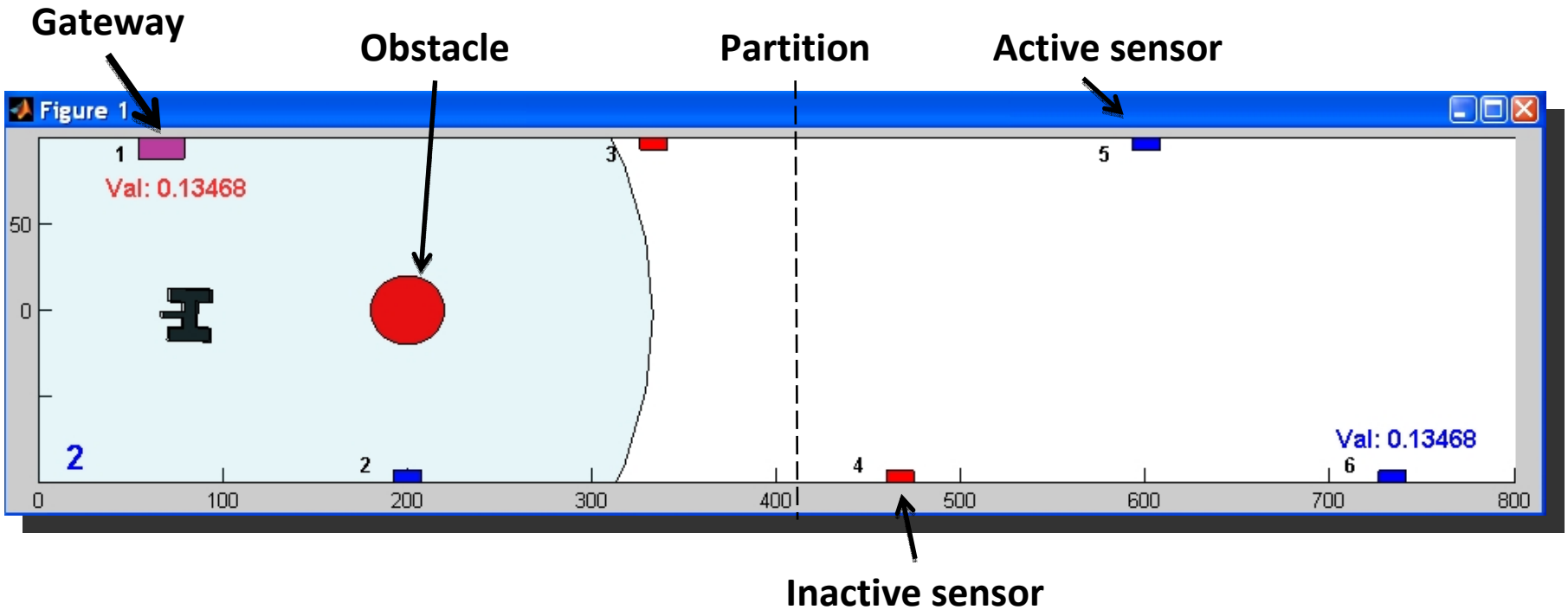


Wheel and Motor Submodel



- 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

Reality