Using JitterTime to Analyze Transient Performance in Adaptive and Reconfigurable Control Systems

Anton Cervin, Paolo Pazzaglia, Mohammadreza Barzegaran, Rouhollah Mahfouzi

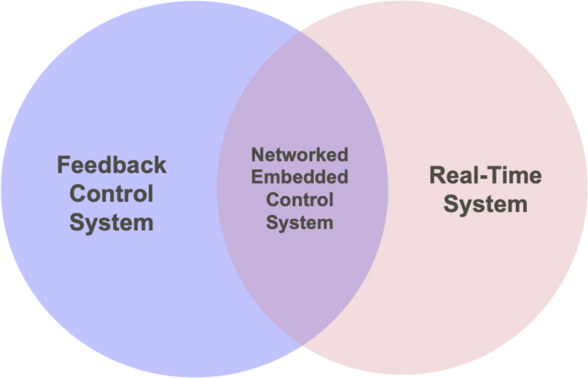
Lund University, Scuola Superiore S'Anna, Technical University of Denmark, Linköping University

What is JitterTime?

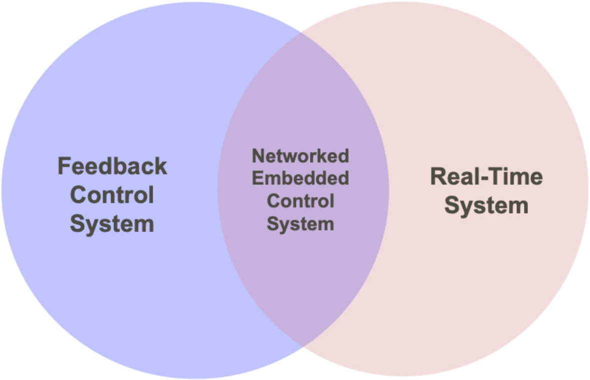
A small Matlab toolbox for calculating the performance of a control application under **non-ideal timing conditions**, e.g.,

- lost samples or lost controls due to packet loss or execution overruns
- delay and jitter due to resource contention (CPU, network, . . .)
- aperiodic behavior due to clock drift or asynchronous nodes/execution

Ultimate goal: optimal co-design



Ultimate goal: optimal co-design



Measure the quality of control using some cost function, e.g.,

$$J = \mathrm{E} \int_0^T \left(x^T(t) Q_1 x(t) + u^T(t) Q_2 u(t) \right) dt$$

Would like to evaluate

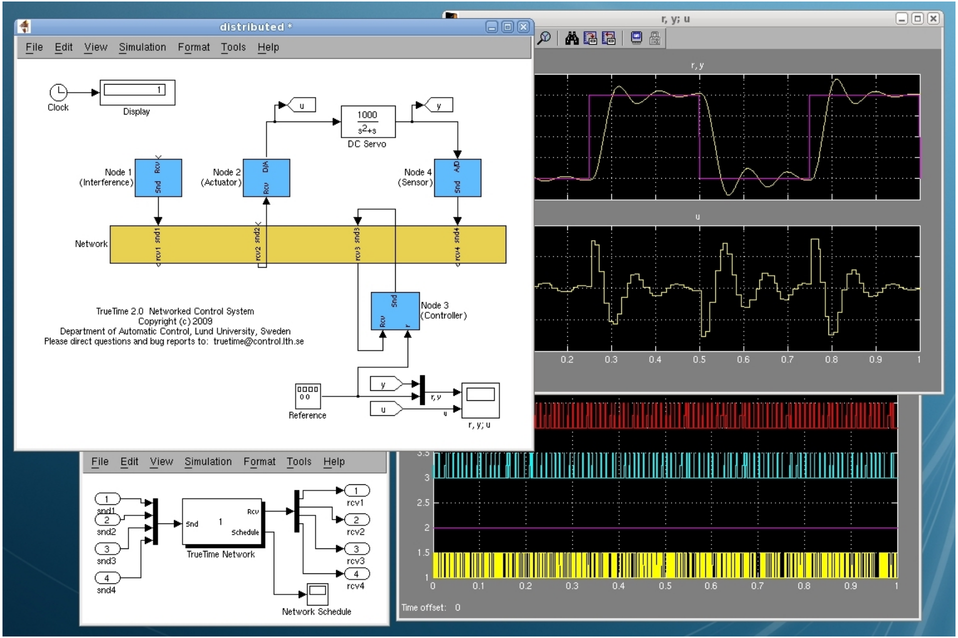
$$J(\text{controller, real-time implementation})$$

Other tools for analyzing real-time control performance

For example, two tools we have previously developed at Lund University:

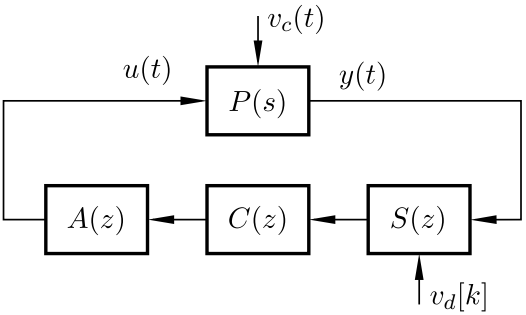
- Simulation using **TrueTime** (Henriksson & Cervin, 2002)
  - Any plant, network, controller models, any performance index
  - Any timing pattern (given by, e.g., scheduling policies)
  - Lengthy Monte Carlo runs needed to evaluate performance with some confidence
- Analysis using **Jitterbug** (Lincoln & Cervin, 2002)
  - Linear systems driven by white noise, quadratic cost function
  - Timing in each (fixed) period given by discrete probability distributions
  - Stationary performance calculated analytically

TrueTime

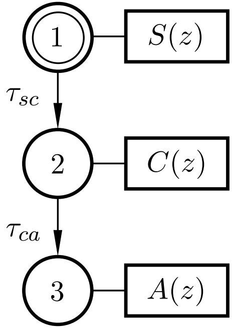

distributed *
File Edit View Simulation Format Tools Help
Clock
Display
u
1000
DC Servo
y
Node 1
(Interference)
Node 2
(Actuator)
Node 4
(Sensor)
Network
Node 3
(Controller)
TrueTime 2.0 Networked Control System
Copyright (c) 2009
Department of Automatic Control, Lund University, Sweden
Please direct questions and bug reports to: truetime@control.lth.se
Reference
r, y; u
r, y
u
TrueTime Network
Network Schedule
snd1
snd2
snd3
snd4
rcv1
rcv2
rcv3
rcv4
Time offset: 0

Jitterbug

Signal model



Timing model



- τ_{sc} and τ_{ca} are random delays with given probability density functions
- Quadratic cost function is evaluated analytically

JitterTime

JitterTime = Jitterbug analysis but with explicit timing

- Signal model is similar to Jitterbug: Linear systems, white noise, quadratic cost
- Timing is arbitrary; completely driven by the user
  - Timing from real system trace or from a discrete-event simulation
  - Performance of deterministic timing scenarios are evaluated exactly
  - Average performance of stochastic timing scenarios require Monte Carlo simulations

JitterTime commands

Command	Purpose
`jtInit`	Initialize a new model
`jtAddContSys`	Add a continuous-time linear system
`jtAddDiscSys`	Add a discrete-time linear system
`jtCalcDynamics`	Calculate the total system dynamics
`jtPassTime`,`jtPassTimeUntil`	Simulate the passing of time
`jtExecSys`	Execute a (version of a) discrete-time system

Internal workings

- All subsystems are merged into a large state-space model
- When time passes, the state covariance P evolves as

$$\frac{dP(t)}{dt} = AP(t) + P(t)A^T + R_c$$

- When a discrete-time system s is executed at time t_k, the covariance is updated according to

$$P(t_k^+) = E_s P(t_k) E_s^T + R_d$$

- The accumulated cost between two discrete executions is given by

$$\Delta\, J = \int_{t_k}^{t_{k+1}} \text{tr}\; Q_c P(t)\, dt,$$

All of the above can be calculated by expressions involving matrix exponentials

Simple example

Minimum-variance sampled-data control of an integrator driven by white noise,

$$\dot{y}(t) = u(t) + v_c(t), \qquad J(t) = \int_0^t y^2(\tau)\, d\tau$$

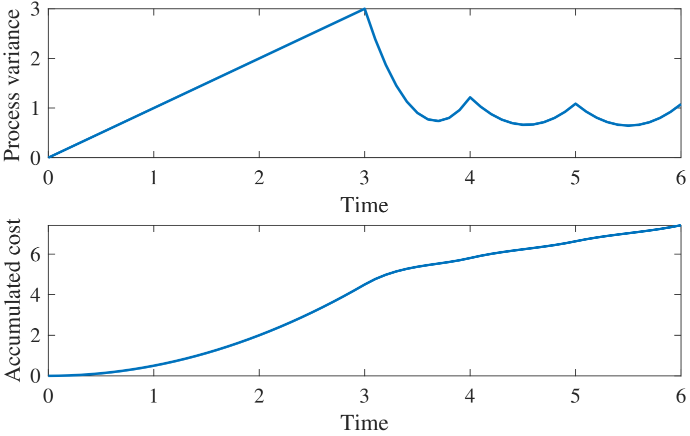
Process runs in open loop for 3 seconds before the controller is activated with $h = 1$

Simple example

Minimum-variance sampled-data control of an integrator driven by white noise,

$$\dot{y}(t) = u(t) + v_c(t), \qquad J(t) = \int_0^t y^2(\tau)\, d\tau$$

Process runs in open loop for 3 seconds before the controller is activated with $h = 1$



Use cases

1. Optimization of static schedules for Fog Control Nodes – M. Barzegaran et al. [1]
2. Analysis of execution overruns in real-time control tasks – P. Pazzaglia et al. [2]
3. Routing and scheduling of control applications over TSN Networks – R. Mahfouzi et al

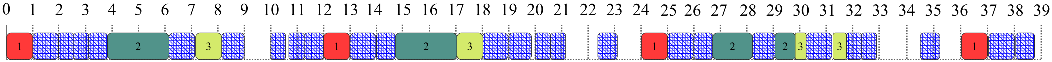
[1] M. Barzegran, A. Cervin, P. Pop, “Towards quality-of-control-awarescheduling of industrial applications on fog computing platforms,” in 1st Workshop on Fog Computing and the IoT (IoT-Fog’19), Montreal, Canada, 2019.

[2] P. Pazzaglia, C. Mandrioli, M. Maggio, A. Cervin, “DMAC: Deadline-miss-aware control,” in 31st Euromicro Conference on Real- Time Systems (ECRTS’19), Stuttgart, Germany, 2019.

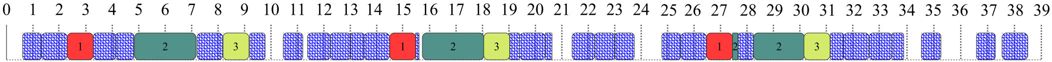
Optimization of static schedules for Fog Control Nodes

Controller consists of three tasks: Input (red), Calculate (dark green), Output (light green)

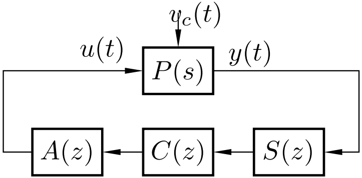
EDF schedule (default):

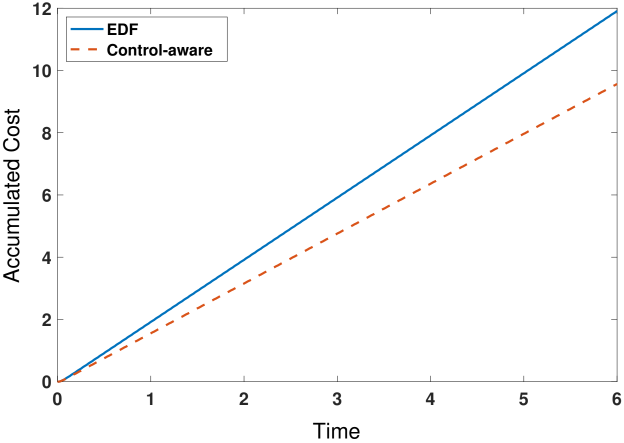


Control-aware schedule:



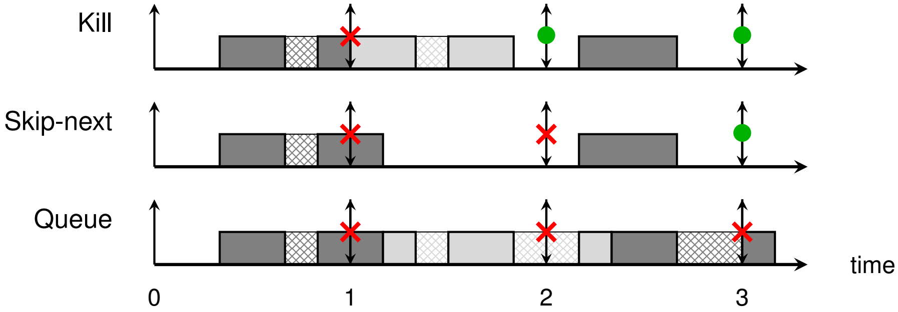
Example of results





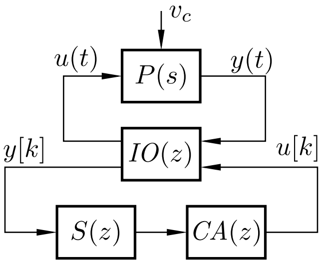
Analysis of execution overruns in real-time control tasks

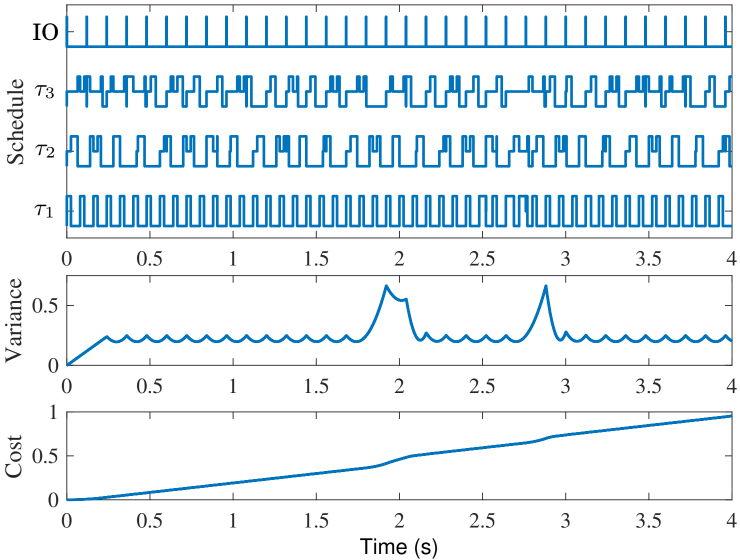
In a periodic control task, what to do if the deadline is missed?



Co-simulation of the dynamic scheduling algorithm and the control performance index (TrueTime + JitterTime)

Example of results





Conclusion

- Trade-off: Expressiveness vs analytical power
- JitterTime offers efficient analysis of very simple models:
  - Linear systems (including MIMO)
  - White noise disturbances
  - Quadratic cost function
- All of the above parameters can be time-varying

Future work:

- Julia implementation
- Deterministic disturbances
- Better numerical stability for large examples