Hardware-in-the-loop test-bed of an Unmanned Aerial Vehicle using Orccad

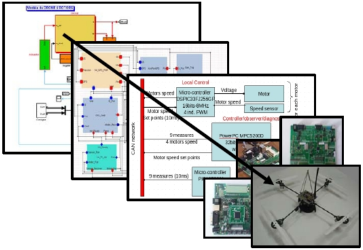
Daniel Simon

INRIA Grenoble Rhône-Alpes
NeCS project-team

CAR 2011
Control Architectures of Robots
May 25^{th}, 2011, Montbonnot

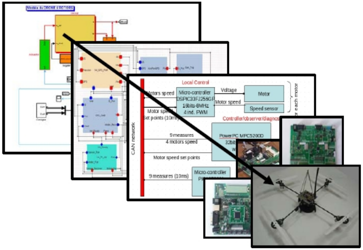
Outline

From control design to real-time



- ANR Safenecs: co-design for control, computing and networking
- Progressive integration of real-time features in control algorithms
- Incremental design and validation
- Reusing models, functions and code (as far as possible)
- Automatic tools when possible

From control design to real-time

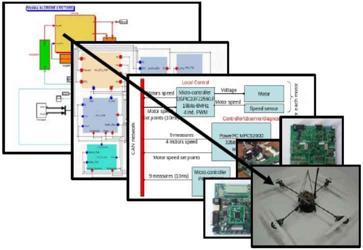


Continuous time design and simulation

- Matlab/Simulink, Scilab/Xcos,...
- Modeling capabilities, components libraries
- Fast prototyping

- Continuous time or simple sampling
- Slow simulation speed

From control design to real-time

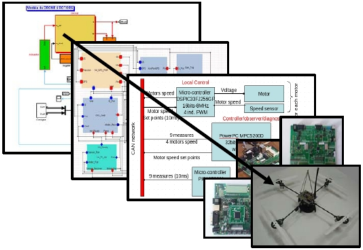


Real-time architecture

- Simulink + TrueTime
- Model of the RT scheduler
- Models of networks (high level)

- Assumptions of execution & transmission times
- Very slow simulation speed

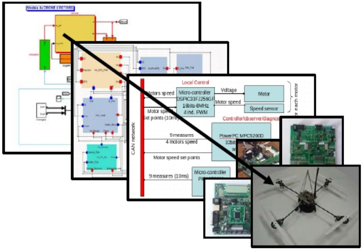
From control design to real-time



Hardware-in-the-loop

- Real-time execution of the control code, OS and protocols
- Real-time numerical integration of the physical process
- No need for final process development
- No risk for the real and costly process and crew
- Code generation from previous models and templates
- Trade-off between accuracy and time

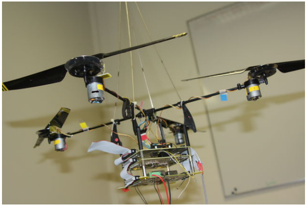
From control design to real-time



Real experiments

- Needs full development of hardware and software
- Cost of failures
- Feedback to previous steps

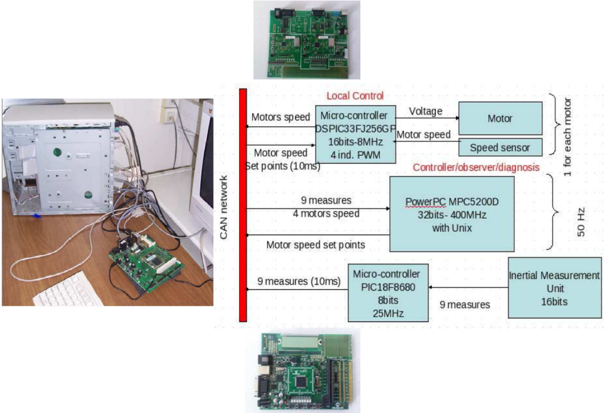
Hardware-in-the-loop setup



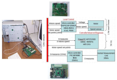
SafeNecs ANR project: Control and diagnosis in Networked Control Systems

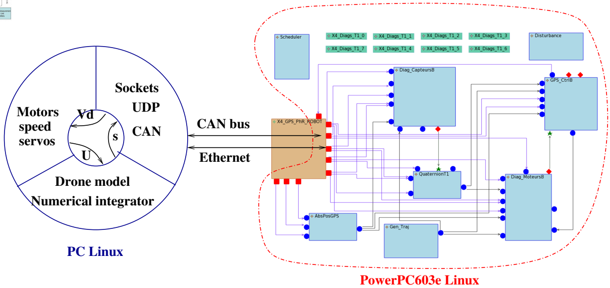
Evaluation of computing/network induced disturbances in control loops and FDI

Hardware-in-the-loop setup



Hardware-in-the-loop setup





Numerical Integrator

Numerical integration of the model, described by ODEs

- Precise enough to faithfully simulate the continuous process dynamics
- Fast enough (w.r.t. the control systems dynamics) to minimize disturbances

$$\frac{dy(t)}{dt} = f(t, y(t)), \quad y(t_0) = y_0, \quad y \in \mathbb{R}^n, \quad t \in \mathbb{R}$$

$$y(t_{i+1}) \simeq y(t_i) + \frac{dy(t_i)}{dt} h_{i+1} + \frac{1}{2!}\frac{d^2y(t_i)}{dt^2} h_{i+1}^2 + \ldots + \frac{1}{n!}\frac{d^ny(t_i)}{dt^n} h_{i+1}^n$$

Trade-off between speed/stability/precision
Governed by the order n, step h, plant's dynamics, method...

Numerical Integrator

Numerical integration of the model, described by ODEs

- Precise enough to faithfully simulate the continuous process dynamics
- Fast enough (w.r.t. the control systems dynamics) to minimize disturbances

$$\frac{dy(t)}{dt} = f(t, y(t)), \quad y(t_0) = y_0, \quad y \in \mathbb{R}^n, \quad t \in \mathbb{R}$$

$$y(t_{i+1}) \simeq y(t_i) + \frac{dy(t_i)}{dt} h_{i+1} + \frac{1}{2!} \frac{d^2 y(t_i)}{dt^2} h_{i+1}^2 + \ldots + \frac{1}{n!} \frac{d^n y(t_i)}{dt^n} h_{i+1}^n$$

Trade-off between speed/stability/precision
Governed by the order n, step h, plant's dynamics, method…

Numerical Integrator

Numerical integration of the model, described by ODEs

- Precise enough to faithfully simulate the continuous process dynamics
- Fast enough (w.r.t. the control systems dynamics) to minimize disturbances

$$\frac{dy(t)}{dt} = f(t, y(t)), \quad y(t_0) = y_0, \quad y \in \mathbb{R}^n, \quad t \in \mathbb{R}$$

$$y(t_{i+1}) \simeq y(t_i) + \frac{dy(t_i)}{dt} h_{i+1} + \frac{1}{2!}\frac{d^2 y(t_i)}{dt^2} h_{i+1}^2 + \ldots + \frac{1}{n!}\frac{d^n y(t_i)}{dt^n} h_{i+1}^n$$

Trade-off between speed/stability/precision
Governed by the order n, step h, plant's dynamics, method...

Numerical Integration

- Explicit (Forward Euler)
  $y(t+h) \simeq y(t) + hf(t, y(t))$
  fast but only conditionally stable for linear systems
- Implicit (Backward Euler)
  $y(t+h) \simeq y(t) + hf(t+h, y(t+h))$
  unconditionally stable for linear systems, stiff problems
- Single step (Runge-Kutta) $y(t+h)$ depends only on $y(t)$
- Multiple steps (Adams, BDF) $y(t+h)$ depends on $y(t), ..., y(t-nh)$
- Fixed step: fixed integration cost, unknown precision
- Adaptive step: precision is constrained, integration time is unpredictable

for a **given precision** variable step is cheaper than fixed step...

Lsoda (Odepack), variable step, multi-step, automatic switching between Adams and BDF, open-source

Numerical Integration

- Explicit (Forward Euler)
  $y(t+h) \simeq y(t) + hf(t, y(t))$
  fast but only conditionally stable for linear systems
- Implicit (Backward Euler)
  $y(t+h) \simeq y(t) + hf(t+h, y(t+h))$
  unconditionally stable for linear systems, stiff problems

- Single step (Runge-Kutta) $y(t+h)$ depends only on $y(t)$
- Multiple steps (Adams, BDF) $y(t+h)$ depends on $y(t), ..., y(t-nh)$

- Fixed step: fixed integration cost, unknown precision
- Adaptive step: precision is constrained, integration time is unpredictable

for a **given precision** variable step is cheaper than fixed step...

Lsoda (Odepack), variable step, multi-step, automatic switching between Adams and BDF, open-source

Numerical Integration

- Explicit (Forward Euler)
  $y(t+h) \simeq y(t) + hf(t, y(t))$
  fast but only conditionally stable for linear systems
- Implicit (Backward Euler)
  $y(t+h) \simeq y(t) + hf(t+h, y(t+h))$
  unconditionally stable for linear systems, stiff problems

- Single step (Runge-Kutta) $y(t+h)$ depends only on $y(t)$
- Multiple steps (Adams, BDF) $y(t+h)$ depends on $y(t), ..., y(t-nh)$

- Fixed step: fixed integration cost, unknown precision
- Adaptive step: precision is constrained, integration time is unpredictable

for a **given precision** variable step is cheaper than fixed step...

Lsoda (Odepack), variable step, multi-step, automatic switching between Adams and BDF, open-source

Numerical Integration

- Explicit (Forward Euler)
  $y(t+h) \simeq y(t) + hf(t, y(t))$
  fast but only conditionally stable for linear systems
- Implicit (Backward Euler)
  $y(t+h) \simeq y(t) + hf(t+h, y(t+h))$
  unconditionally stable for linear systems, stiff problems

- Single step (Runge-Kutta) $y(t+h)$ depends only on $y(t)$
- Multiple steps (Adams, BDF) $y(t+h)$ depends on $y(t), ..., y(t-nh)$

- Fixed step: fixed integration cost, unknown precision
- Adaptive step: precision is constrained, integration time is unpredictable

for a **given precision** variable step is cheaper than fixed step...

Lsoda (Odepack), variable step, multi-step, automatic switching between Adams and BDF, open-source

Numerical Integration

- Explicit (Forward Euler)
  $y(t+h) \simeq y(t) + hf(t, y(t))$
  fast but only conditionally stable for linear systems
- Implicit (Backward Euler)
  $y(t+h) \simeq y(t) + hf(t+h, y(t+h))$
  unconditionally stable for linear systems, stiff problems

- Single step (Runge-Kutta) $y(t+h)$ depends only on $y(t)$
- Multiple steps (Adams, BDF) $y(t+h)$ depends on $y(t), ..., y(t-nh)$

- Fixed step: fixed integration cost, unknown precision
- Adaptive step: precision is constrained, integration time is unpredictable

for a **given precision** variable step is cheaper than fixed step...

Lsoda (Odepack), variable step, multi-step, automatic switching between Adams and BDF, open-source

Numerical Integration

- Explicit (Forward Euler)
  $y(t+h) \simeq y(t) + hf(t, y(t))$
  fast but only conditionally stable for linear systems
- Implicit (Backward Euler)
  $y(t+h) \simeq y(t) + hf(t+h, y(t+h))$
  unconditionally stable for linear systems, stiff problems

- Single step (Runge-Kutta) $y(t+h)$ depends only on $y(t)$
- Multiple steps (Adams, BDF) $y(t+h)$ depends on $y(t), ..., y(t-nh)$

- Fixed step: fixed integration cost, unknown precision
- Adaptive step: precision is constrained, integration time is unpredictable

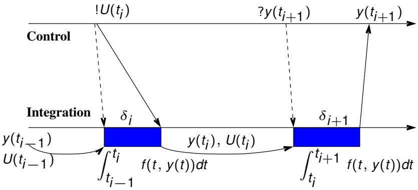
for a **given precision** variable step is cheaper than fixed step...

Lsoda (Odepack), variable step, multi-step, automatic switching between Adams and BDF, open-source

Numerical Integrator Synchronization

Real-time simulation

- Clock-driven controller
- N.I. triggered by I/O events
- Late w.r.t. real-time



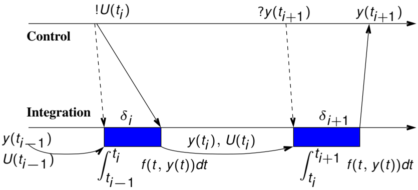
Events generated by the process

- Impacts, dry friction, ignition,...
- Root finding function (LsodaR)
- Integration ahead of real-time

Integration as fast as possible

- Integration driven control
- Consistency of the time scales

Numerical Integrator Synchronization



Real-time simulation

- Clock-driven controller
- N.I. triggered by I/O events
- Late w.r.t. real-time

Events generated by the process

- Impacts, dry friction, ignition,...
- Root finding function (LsodaR)
- Integration ahead of real-time

Integration as fast as possible

- Integration driven control
- Consistency of the time scales

Numerical Integrator Synchronization

Real-time simulation

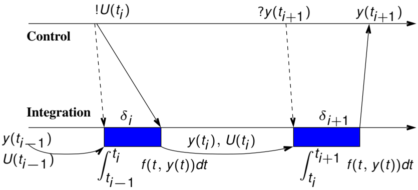
- Clock-driven controller
- N.I. triggered by I/O events
- Late w.r.t. real-time

Events generated by the process

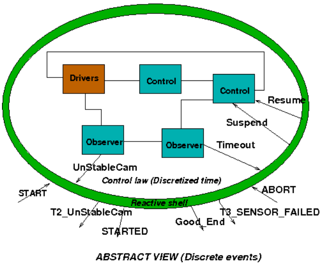
- Impacts, dry friction, ignition,...
- Root finding function (LsodaR)
- Integration ahead of real-time

Integration as fast as possible

- Integration driven control
- Consistency of the time scales



The ORCCAD model



RobotTasks

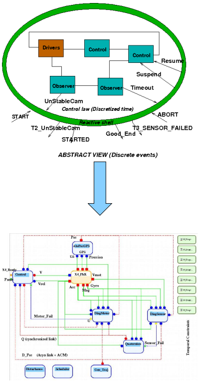
- Feedback Control
- Cyclic real-time data flow
- Event-based view

RobotProcedures

- Discrete Events Control
- Incremental design
- Exception processing
- Mission definition

Bottom up approach, from control to real-time

Control action: the RobotTask



Feedback control action

- Control algorithm definition
- Invariant structure for RT life
- Modular design
- Functional parameters
- Timing parameters

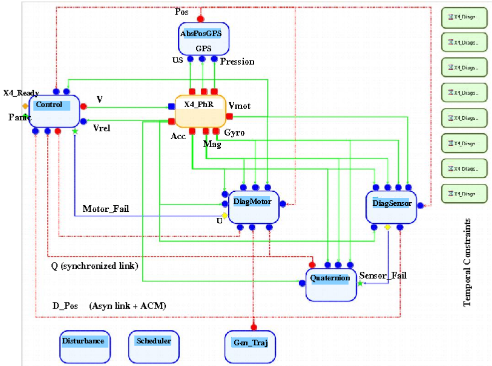
Event based behaviour

- Precondition (opt. timeout)
- Synchronization
- Exceptions
  - Weak T1
  - Strong T2
  - Fatal T3

Postcondition (opt. timeout)

Orccad model

Drone control block-diagram



Networked system

- CAN bus
- Distributed diagnosis
- Fault tolerant control

Flexible scheduling

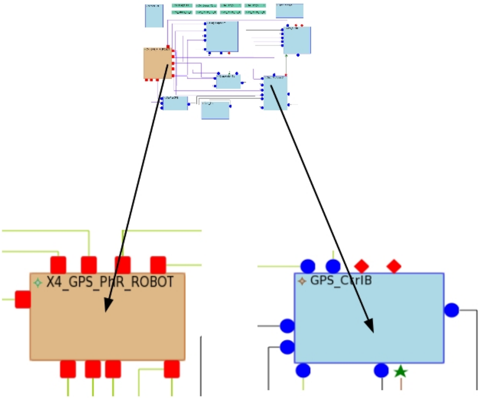
- Varying sampling
- (m,k)-firm
- Dynamic priorities

Hardware-in-the-loop

- Linux simulation
- PPC embedded

V4 Runtime update

Orccad components: Modules



Implement functions

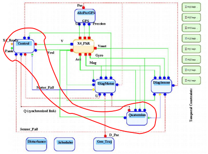
Algorithmic
Phy_Resource (drivers)

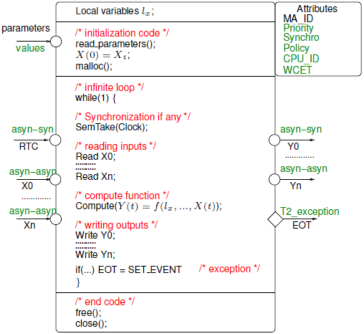
Typed Input/Output ports
- Data
- Drivers
- Parameters
- Events

User defined C code

init(inputs)
forever{
compute(inputs)
}
end()

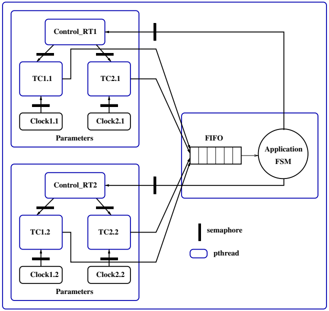
Orccad components: Temporal Constraints





- Task ID
- Module ID
- Priority
- Synchronization
  - Clock
  - Output port
  - Extern event
- Overrun policy
  - Skip, Soft, Hard
  - User's defined
- WCET
- CPU ID

Code generation



Code generation

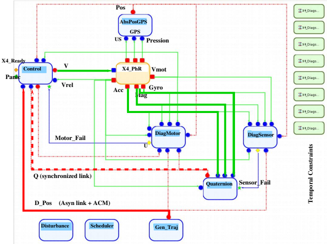
- C++ classes
- Virtual system calls

Compilation

- Binding to real calls
- Link with runtime library
- Linux/Posix
- Xenomai/Native
- ...

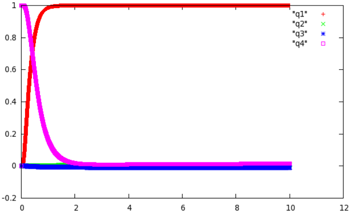
	Orccad	Linux/Posix	Xenomai/Native
launch a RT task	orcSpawn	pthread_create()	rt_task_spawn()
timer	orcTimer_t	timer_t	RT_ALARM
message queue	orcmsgQ_t	mqd_t	RT_QUEUE
semaphore	orcSem_t	sem_t	RT_SEM

Attitude controller



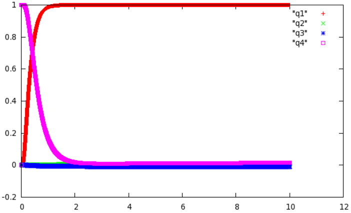
- C code from various sources
  - drone model from Matlab/Rtw
  - VTOL LQ
    saturated integrators
  - Non-Linear observer
    EKF with missing data
- Synchronized links for strongly affine modules
- Data protection: ACM on asyn links
- CPU affinity on multi-core
- UDP or CAN sockets

Attitude controller

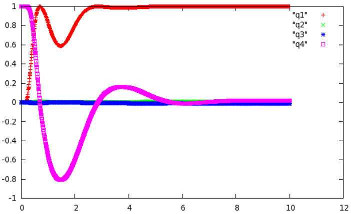


local loop IP=127.0.0.1 h = 5 msecs

Attitude controller



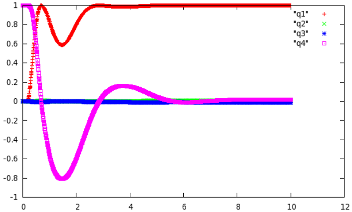
local loop IP=127.0.0.1 h = 5 msecs




```
Skipping 1053 on mt_X4_MT_2
CTRL  497712
Skipping 9 on mt_X4_MT_1
Skipping 1054 on mt_X4_MT_2
Skipping 9 on mt_X4_MT_3
Skipping 9 on mt_X4_MT_4
Skipping 18 on mt_X4_MT_5
Skipping 18 on mt_X4_MT_6
Skipping 19 on mt_X4_MT_7
Skipping 1055 on mt_X4_MT_2
Skipping 1056 on mt_X4_MT_2
Skipping 1057 on mt_X4_MT_2
CTRL  497876
CTRL  498104
Skipping 1058 on mt_X4_MT_2
```

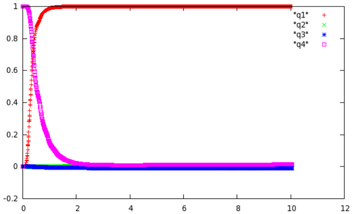
Ethernet PC <-> PowerPC h = 5msecs

Attitude controller




```
Skipping 1053 on mt_X4_MT_2
CTRL  497712
Skipping 9 on mt_X4_MT_1
Skipping 1054 on mt_X4_MT_2
Skipping 9 on mt_X4_MT_3
Skipping 9 on mt_X4_MT_4
Skipping 18 on mt_X4_MT_5
Skipping 18 on mt_X4_MT_6
Skipping 19 on mt_X4_MT_7
Skipping 1055 on mt_X4_MT_2
Skipping 1056 on mt_X4_MT_2
Skipping 1057 on mt_X4_MT_2
CTRL  497876
CTRL  498104
Skipping 1058 on mt_X4_MT_2
```

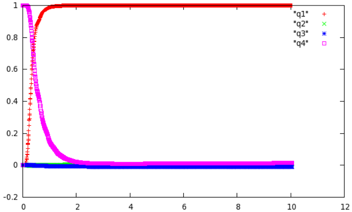
Ethernet PC <-> PowerPC h = 5msecs

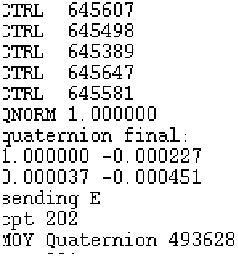



```
CTRL  645607
CTRL  645498
CTRL  645389
CTRL  645647
CTRL  645581
QNORM 1.000000
quaternion final:
1.000000 -0.000227
0.000037 -0.000451
sending E
cpt 202
MOY Quaternion 493628
```

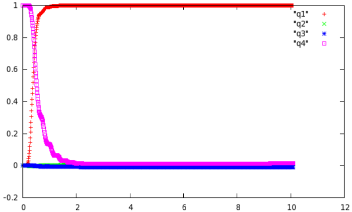
Ethernet PC <-> PowerPC h = 50msecs

Attitude controller



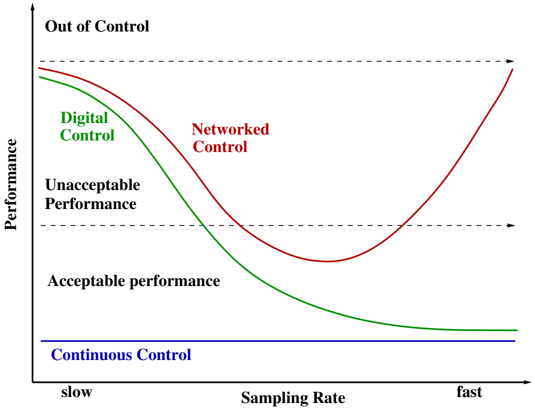


Ethernet PC <-> PowerPC h = 50msecs

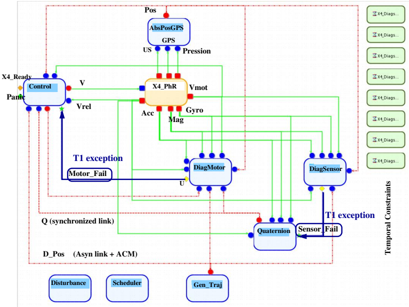


CAN PC <-> PowerPC 250 Kbps, h = 50 msecs

Attitude controller

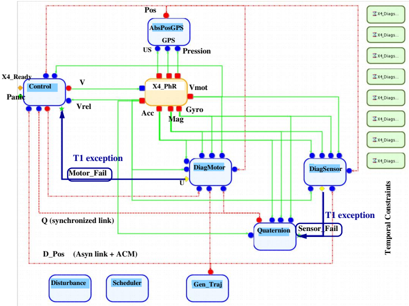


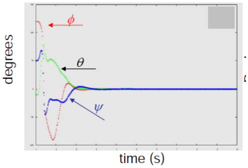
Diagnosis and FTC



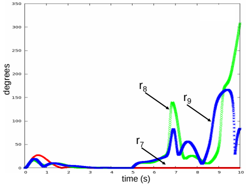
- Diagnosis functions raise T1 exception
- T1 signaled to control module
- Exception value sent on a parameter port
- Branch in function code

Diagnosis and FTC



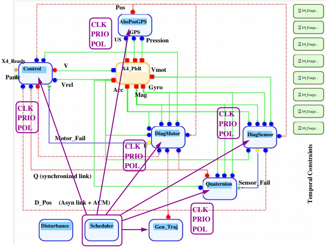


attitude with 10 % network packet loss



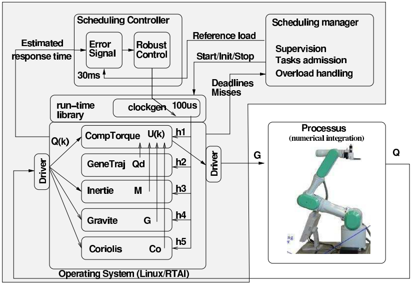
residuals

Feedback scheduling

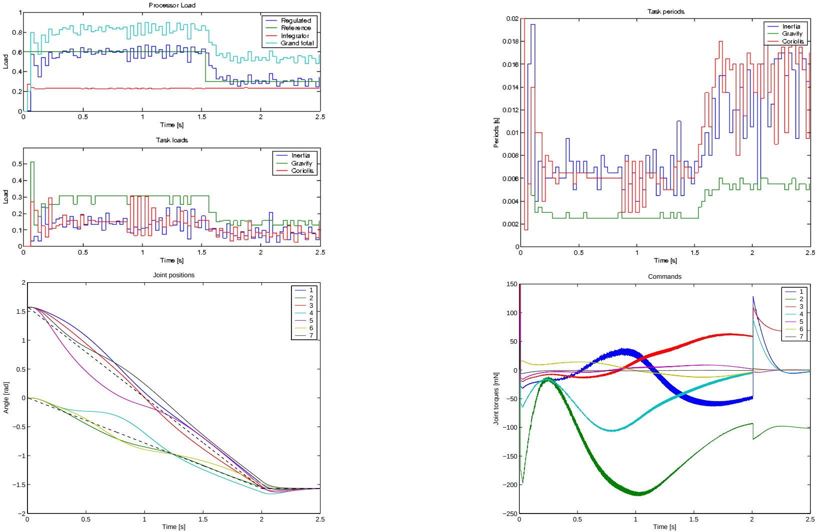


- Varying sampling, (m,k)-firm,. . .
- CAN priorities
- Overrun policies: skip, continue, stop,. . .
- dedicated API
  - orcTimerSetTime(id, period)
  - orcGetCpuTime()
  - orcGetExecTime(task)
  - MTSetSafeSampleTime(period)
  - task->Missed
  - O.S. dependent behaviour!

Feedback scheduling a robot controller



Feedback scheduling a robot controller



Conclusion

- HIL is an efficient step before real experiments
- Incremental development from control design to runtime
- Smart integration of physical and simulated components

- Choice and synchronisation of the Numerical Integrator
- Integrators with root finding capabilities
- Parallel implementation

Conclusion

- HIL is an efficient step before real experiments
- Incremental development from control design to runtime
- Smart integration of physical and simulated components

- Choice and synchronisation of the Numerical Integrator
- Integrators with root finding capabilities
- Parallel implementation

Conclusion

- HIL is an efficient step before real experiments
- Incremental development from control design to runtime
- Smart integration of physical and simulated components

- Choice and synchronisation of the Numerical Integrator
- Integrators with root finding capabilities
- Parallel implementation

Conclusion

- HIL is an efficient step before real experiments
- Incremental development from control design to runtime
- Smart integration of physical and simulated components

- Choice and synchronisation of the Numerical Integrator
- Integrators with root finding capabilities
- Parallel implementation

Conclusion

- HIL is an efficient step before real experiments
- Incremental development from control design to runtime
- Smart integration of physical and simulated components

- Choice and synchronisation of the Numerical Integrator
- Integrators with root finding capabilities
- Parallel implementation

Conclusion

- HIL is an efficient step before real experiments
- Incremental development from control design to runtime
- Smart integration of physical and simulated components

- Choice and synchronisation of the Numerical Integrator
- Integrators with root finding capabilities
- Parallel implementation

Conclusion

- HIL is an efficient step before real experiments
- Incremental development from control design to runtime
- Smart integration of physical and simulated components

- Choice and synchronisation of the Numerical Integrator
- Integrators with root finding capabilities
- Parallel implementation

Questions?