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\textsuperscript{th}, 2011, Montbonnot
Outline

1. From control design to real-time
2. Hardware-in-the-loop setup
   - Architecture
   - Numerical Integration
3. Controller design
   - Orccad model
   - Runtime
4. NCS experiments
   - Attitude control
   - Diagnosis
   - Feedback scheduling
5. Summary

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

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

Architecture

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

Architecture

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

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

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

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

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

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

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

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

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

Trade-off between speed/stability/precision
Governed by the order \( n \), step \( h \), plant’s dynamics, method...
Numerical Integration

- **Explicit (Forward Euler)**
  \[ y(t + h) \approx y(t) + hf(t, y(t)) \]
  fast but only conditionally stable for linear systems

- **Implicit (Backward Euler)**
  \[ y(t + h) \approx y(t) + hf(t + h, y(t + h)) \]
  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)**
\[ y(t + h) \approx y(t) + hf(t, y(t)) \]
fast but only conditionally stable for linear systems

**Implicit (Backward Euler)**
\[ y(t + h) \approx y(t) + hf(t + h, y(t + h)) \]
unconditionally stable for linear systems, stiff problems

**Single step (Runge-Kutta)**
\[ y(t + h) \text{ depends only on } y(t) \]

**Multiple steps (Adams, BDF)**
\[ y(t + h) \text{ depends on } y(t), ..., y(t - nh) \]

- 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)
  \[ y(t + h) \approx y(t) + hf(t, y(t)) \]
  fast but only conditionally stable for linear systems

- Implicit (Backward Euler)
  \[ y(t + h) \approx y(t) + hf(t + h, y(t + h)) \]
  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 \( y(t), \ldots, y(t - nh) \)

- 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)
  \[ y(t + h) \approx y(t) + hf(t, y(t)) \]
  fast but only conditionally stable for linear systems

- Implicit (Backward Euler)
  \[ y(t + h) \approx y(t) + hf(t + h, y(t + h)) \]
  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)
  \[ y(t + h) \approx y(t) + hf(t, y(t)) \]
  fast but only conditionally stable for linear systems

- Implicit (Backward Euler)
  \[ y(t + h) \approx y(t) + hf(t + h, y(t + h)) \]
  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)**
  \[ y(t + h) \approx y(t) + hf(t, y(t)) \]
  fast but only conditionally stable for linear systems

- **Implicit (Backward Euler)**
  \[ y(t + h) \approx y(t) + hf(t + h, y(t + h)) \]
  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)**
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

**From control design to real-time**

**Hardware-in-the-loop setup**

**Controller design**

**NCS experiments**

**Summary**

**Orccad model**

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

- C++ classes
- Virtual system calls

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

<table>
<thead>
<tr>
<th>Feature</th>
<th>Orccad</th>
<th>Linux/Posix</th>
<th>Xenomai/Native</th>
</tr>
</thead>
<tbody>
<tr>
<td>launch a RT task</td>
<td>orcSpawn</td>
<td>pthread_create()</td>
<td>rt_task_spawn()</td>
</tr>
<tr>
<td>timer</td>
<td>orcTimer_t</td>
<td>timer_t</td>
<td>RT_ALARM</td>
</tr>
<tr>
<td>message queue</td>
<td>orcmqQ_t</td>
<td>mqd_t</td>
<td>RT_QUEUE</td>
</tr>
<tr>
<td>semaphore</td>
<td>orcSem_t</td>
<td>sem_t</td>
<td>RT_SEM</td>
</tr>
</tbody>
</table>

Hardware-in-the-loop setup

Controller design

NCS experiments

Summary
Attitude control

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 control

Attitude controller

local loop IP=127.0.0.1   h = 5 msecs
Attitude control

Attitude controller

local loop IP=127.0.0.1  h = 5 msecs

Ethernet PC <-> PowerPC  h = 5msecs
Attitude controller

Ethernet PC <-> PowerPC  h = 5msecs

Ethernet PC <-> PowerPC  h = 50msecs
Attitude control

Attitude controller

Ethernet PC <-> PowerPC  h = 50msecs

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

Attitude controller

![Diagram showing the relationship between sampling rate and performance with three regions: Out of Control, Unacceptable Performance, and Acceptable performance. The diagram uses lines to show the transition from continuous control to digital control and networked control at different sampling rates.]

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

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

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

Feedback scheduling a robot controller
Feedback scheduling

Feedback scheduling a robot controller

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