Embedded optimizing control using the OpenModelica C++ runtime
OpenModelica Workshop 2016
Overview

- Motivation and treatment of optimal control programs
- Embedded revolution
- C++ for a modern real-time runtime
- New development: synchronous equations
Dynamic Optimization
Treat optimal control programs basing on simulation models

For dynamic system model and sample time points $t_k$, $t_0 < t_1 < \ldots < t_K$

- find control $u$ (and/or initial states $x(0)$) that minimize criterion $J$
- subject to mixed discrete/continuous model, initial conditions
- and further constraints $g$

$$J = \sum_{k=0}^{K} f_0 \left[ k, \left( x_d(k), x_c(t_k), u_d(k), u_c(t_k) \right) \right] \rightarrow \min_{x_d(0), u_d(k), x_c(t_0), u_c(t_k)}$$

$$x_d(k + 1) = f_d[k, x_d(k), x_c(t_k), u_d(k)], \quad x_d(0) = x_d0, \quad k = 0, 1, \ldots, K$$

$$\frac{dx_c(t)}{dt} = f_c[t, x_d(k(t)), x_c(t), u_c(t)], \quad x_c(t_0) = x_c0, \quad t \in [t_0, t_K]$$

$$g[t, x_d(k(t)), x_c(t), u_d(k(t)), u_c(t)] \geq 0$$
Some industrial applications of model-based control with HQP solver
The power of mathematical programming

- Wagenpfeil et al, 2014: Water canal system (Uni Stuttgart)
- Franke et al, 2014: Virtual power plants (ABB)
- Neupert et al, 2010: Boom cranes (Uni Stuttgart, Liebherr)
- Nagy et al, 2007: Polymerization reactors (Uni Stuttgart, BASF)
- Franke et al, 2006: Power plant start-up (ABB)
- Linke et al, 1997: Water canal system (Uni Ilmenau, MLK)
Overview

- Motivation and treatment of optimal control programs
- Embedded revolution
- C++ for a modern real-time runtime
- New development: synchronous equations
Embedded revolution
Hardware leaped ahead during last decade – Software still too expensive

**Embedded traditional**

- Special purpose hardware
- Very low computing resources – kHz, kBytes, no floating point, …
- Simple special purpose operating systems
Embedded revolution
Hardware leaped ahead during last decade – Software still too expensive

**Embedded traditional**
- Special purpose hardware
- Very low computing resources – kHz, kBytes, no floating point, …
- Simple special purpose operating systems

**Embedded in the 21st century**
- General purpose hardware (mobile platforms) – cf. Raspberry PI starting at 5$
- High computing power – GHz, GBytes, HD Graphics, System on Chip (SoC)
- General purpose operating systems

**Way forward**
- Powerful hardware has become available for embedded at low cost
- Software still too expensive – need to increase productivity
- need to develop and exploit appropriate software technologies, such as C++
C++
High-level programming / type safety / high runtime performance

Initiated by Bjarne Stroustrup in 1979; motivated by object-oriented Simula 67

C++98 (ISO/IEC 14882:1998)
- Including Standard Template Library

C++03 (ISO/IEC 14882:2003)
- revised C++98

C++11 (ISO/IEC 14882:2011)
- New library modules, largely impacted by boost library:
  regular expressions, threads, time, containers, static array, …
- auto keyword, simpler array initialization, lambdas, …

C++14 (ISO/IEC 14882:2014)
- revised C++11

C++17 (upcoming)
C++ features used by the OpenModelica Cpp runtime

- Classes with public interfaces and protected implementations
- Deterministic memory management (no need for garbage collection)
- Templates (e.g. Arrays of different types, up to array of `std::string` or records)
- Type safety (e.g. dimension of static array being part of type)
- Exception handling

→ High-level features reduce implementation effort while C++ compilers generate very fast code

C++ aims to “leave no room for a lower-level language … (except for assembly code in rare cases)” (Stroustrup, 2014)
Obtained CPU times with different runtimes for same DrumBoiler example

Considerable speed-ups, in particular with C++ compiler optimization

<table>
<thead>
<tr>
<th>Modelica Tool for FMU export</th>
<th>CPU time with gcc 4.9.2 flag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-O0</td>
</tr>
<tr>
<td>OpenModelica 1.9.3</td>
<td>9.1 s</td>
</tr>
<tr>
<td>OpenModelica 1.9.3 +cseCall</td>
<td>4.0 s</td>
</tr>
<tr>
<td>Dymola 2015FD01</td>
<td>3.4 s</td>
</tr>
<tr>
<td>OpenModelica 1.9.3 +simCodeTaget=Cpp</td>
<td>5.6 s</td>
</tr>
<tr>
<td>OpenModelica 1.9.3 +simCodeTaget=Cpp +cseCall</td>
<td>2.7 s</td>
</tr>
</tbody>
</table>

Overview

- Motivation and treatment of optimal control programs
- Embedded revolution
- C++ for a modern real-time runtime
- New development: synchronous equations
Example: Double Integrator Discrete-time

model DID "Double Integrator Discrete-time"
  parameter Real p = 1 "gain for input";
  parameter Real y1_start = 1 "start value for first state";
  parameter Real y2_start = 0 "start value for second state";
  input Real u(start = -2);
  Real xd1(start = y1_start), xd2(start = y2_start);
  output Real y1, y2;
  equation
    when Clock(1, 20) then
      xd1 = previous(xd1) + p * u * interval(u);
      xd2 = previous(xd2) + previous(xd1) * interval(u) + 0.5 * u * interval(u)^2;
      y1 = previous(xd1);
      y2 = previous(xd2);
    end when;
  end DID;
Example: simulation in Dymola 2015 FD01
Example: simulation in OpenModelica (simCodeTarget=Cpp)
Minimize

- control effort

subject to model equations and

- initial states

- final states

- state/output constraint

\[
J = \sum_{k=0}^{K} u^2(k) \quad \text{min}
\]

\[
y_1(t_0) = 1, \quad y_2(t_0) = 0
\]

\[
y_1(t_K) = -1, \quad y_2(t_K) = 0
\]

\[
y_2(t) \leq 0.1, \quad t \in [t_0, t_K]
\]
Example: simulation and optimization using HQP
Importing FMU exported with OpenModelica (simCodeTarget=Cpp)
Conclusions

- Model-based applications are often treated as optimal control programs
- New embedded trends enable more applications
  - Powerful hardware has become available at low cost
  - Software still too expensive – need to increase productivity
  - Need to develop and exploit appropriate software technologies, such as C++
- OpenModelica C++ runtime
  - Exploit C++ features (e.g. memory management, templates, type safety)
  - Achieved superior results, compared to other Modelica tools or runtimes
  - Drawback of C++: higher compile/linker requirements – encapsulate in FMI
  - Increased maturity with new compilers supporting C++11 (replacing boost)
  - Serve as basis for new development of FMI export with clocked equations
Power and productivity for a better world™