Eskild Schroll-Fleischer

Model Predictive Control:
From Design to Implementation

12^{th} of July 2017

Overview

1. Introduction to design vs. implementation
2. Four tank process as illustrative example
3. Brief overview of OPC
4. Testing before commissioning using a process emulator
5. Implementation of MPC on four tank process

Design vs. Implementation

- MPC may be designed using simulation models
  - Sensors and actuators are ideal and local
  - System states are known
  - Computation time is not an issue

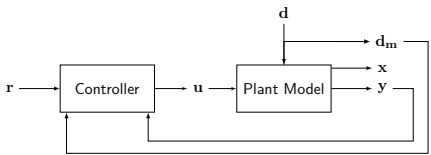

Figure: System considered in the design phase.

- This is seldom the case when implementing

Design vs. Implementation

- An advanced controller interfaces with equipment through an interoperability layer
  - Sensors and actuators are *not* ideal
  - System states are *not* known
  - Computation time *may* be an issue

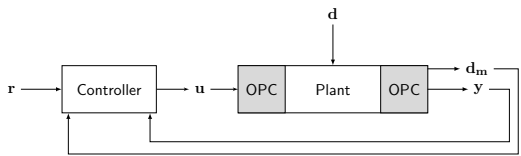

Figure: System considered in the implementation phase.

Four Tank Process

Figure: Four tank process.

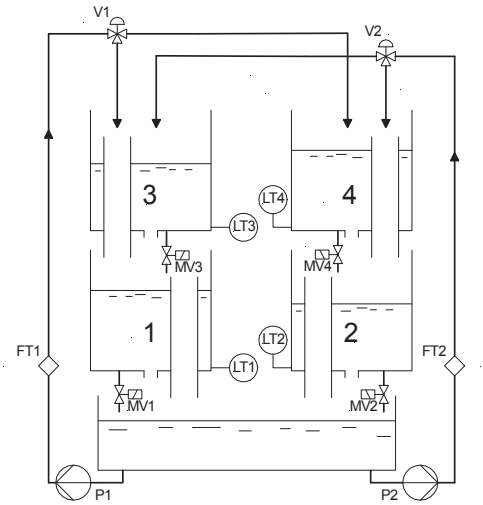

Figure: Four tank process P&ID.

Four Tank Process

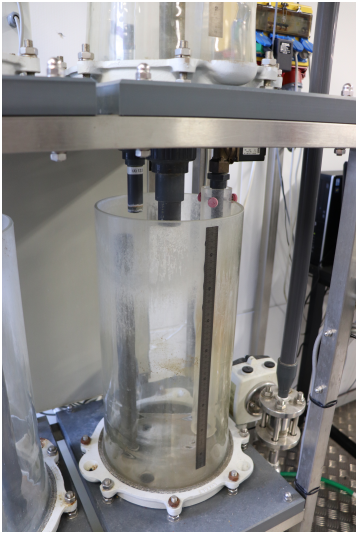

Figure: Four tank process.

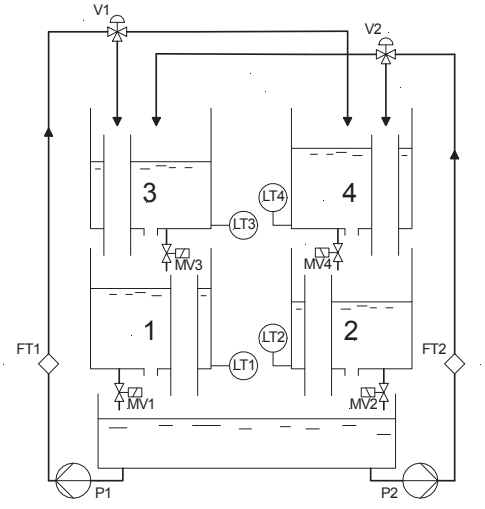

Figure: Four tank process P&ID.

Model Predictive Control of Four Tank Process

- Constrained MPC using discretized linearization of first-principles ODE model

$$\dot{m}_1 = \rho\,(q_{1,\text{in}} + q_{3,\text{out}} - q_{1,\text{out}})$$
$$\dot{m}_2 = \rho\,(q_{2,\text{in}} + q_{4,\text{out}} - q_{3,\text{out}})$$
$$\dot{m}_3 = \rho\,(q_{3,\text{in}} - q_{3,\text{out}})$$
$$\dot{m}_4 = \rho\,(q_{4,\text{in}} - q_{4,\text{out}})$$

$$q_{i,\text{out}} = a\sqrt{\frac{2\,g}{\rho\,A}\,m_i},\ \ q_{1,\text{in}} = F_1\,\gamma_1,\ \ q_{2,\text{in}} = F_2\,\gamma_2$$

$$q_{3,\text{in}} = F_2\,(1-\gamma_2),\ \ q_{4,\text{in}} = F_1\,(1-\gamma_1),$$

$$A = 380\,\text{cm}^2,\ a = 1.22\,\text{cm}^2,\ \gamma_1 = 0.38, \gamma_2 = 0.24$$

- Variable classification:
  u: P1, P2
  y: LT1, LT2, LT3, LT4
  z: LT1, LT2
  d: MV1, MV2, MV3, MV4

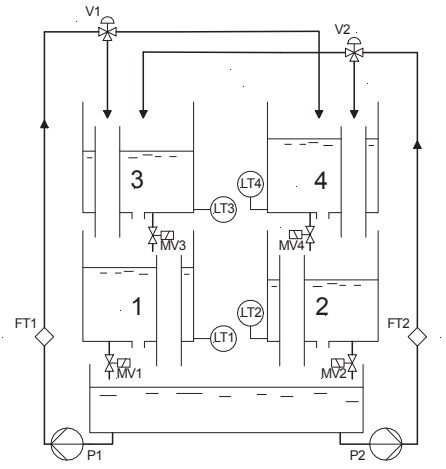

Figure: Four tank process P&ID.

Regulator

- Model is augmented with integrating states to enable offset-free control
- Linear time-invariant measurement-update time-update Kalman filter
- Linear MPC:

$$\min_{\mathbf{u}} \quad \frac{1}{2} \sum_{k=1}^{N_{\mathrm{mpc}}} ||\mathbf{z}_k - \mathbf{r}_k||^2_{\mathbf{Q}_z} + ||\mathbf{u}_{k-1} - \mathbf{u}_{k-2}||^2_{\mathbf{Q}_{\Delta u}}$$

subject to

$$\begin{aligned}
&\mathbf{x}_{k+1} = \mathbf{A}_{\mathrm{d}} + \mathbf{B}_{\mathrm{d}}\mathbf{u}_k + \mathbf{E}_{\mathrm{d}}\mathbf{d}_k, && k = 0, 1, 2, .., N_{\mathrm{MPC}} - 1 \\
&\mathbf{z}_k = \mathbf{C}_{\mathrm{d},\,z}\mathbf{x}_k, && k = 1, 2, 3, ..., N_{\mathrm{MPC}} \\
&\mathbf{u}_{\mathrm{min},\,k} \leq \mathbf{u}_k \leq \mathbf{u}_{\mathrm{max},\,k}, && k = 0, 1, 2, ..., N_{\mathrm{MPC}} - 1 \\
&\Delta\mathbf{u}_{\mathrm{min},\,k} \leq \mathbf{u}_{k+1} - \mathbf{u}_k \leq \Delta\mathbf{u}_{\mathrm{max},\,k}, && k = 0, 1, 2, ..., N_{\mathrm{MPC}} - 1
\end{aligned}$$

- $\mathbf{Q}_z = 5 \cdot 10^5 \mathbf{I}$, $\mathbf{Q}_{\Delta u} = \mathbf{I}$, $|\Delta u| \leq 100\,\mathrm{cm^3\,s^{-1}}$ and $170\,\mathrm{cm^3\,s^{-1}} \leq u \leq 500\,\mathrm{cm^3\,s^{-1}}$
- Prediction horizon: 200, control horizon: 190, $T_s = 5\,\mathrm{s}$

Instrumentation

Table: Process equipment.

P&ID Id	Brand	Model	IO
V1, V2	Samson	3226, I/P positioner 3760	$4-20\,\mathrm{mA}$
MV1–MV4	Bürkert	6213	0 or $10\,\mathrm{V}$
LT1–LT4	Siemens SITRANS	7MF1563-3AA-00	$4-20\,\mathrm{mA}$
FT1, FT2	Danfoss MAG	1100/5000	$4-20\,\mathrm{mA}$
P1, P2	Grundfos MAGNA	25-60 N 180 1x230-240V PN6/10	$0-10\,\mathrm{V}$

- Sensors and actuators are calibrated
- Regulator is implemented in Python

  Libraries: OpenOPC, freeopcua, cvxopt, scipy, matplotlib

The Problem

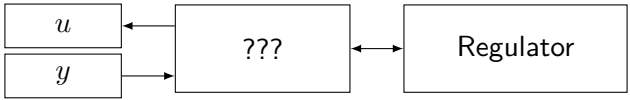

Figure: Communications between process equipment and regulator.

The OPC Foundation

> *OPC technologies were created to allow information to be easily and securely exchanged between diverse platforms from multiple vendors and to allow seamless integration of those platforms without costly, time-consuming software development.*
>
> *– OPC Foundation*

- Acronym for Microsoft Object Linking & Embedding for **Process Control**
- Original OPC standard (now referred to as OPC Classic) was published in 1996
- Most recent OPC standard called OPC Unified Architecture was published in 2006
- 512 members representing industry including ABB, Siemens, Rockwell, Schneider Electric and Omron

OPC Classic

OPC Classic comprises the following components:

- **OPC Data Access (OPC DA)**
  The OPC DA specification defines the exchange of data including values, time and quality information.
- **OPC Alarms & Events (OPC AE)**
  The OPC A&E specification defines the exchange of alarm and event type message information, as well as variable states and state management.
- **OPC Historical Data Access (OPC HDA)**
  The OPC HDA specification defines query methods and analytics that may be applied to historical, time-stamped data.

Drawbacks:

- Only implemented on Windows
- Low-level data structures
- Difficult to handle securely on a network

Advantages:

- Widely adopted by industry

OPC Unified Architecture

OPC-UA is the key to IoT and Industry 4.0

- All COM OPC Classic specifications are mapped to UA ensuring compatibility
- Platform independent
- Encrypted communications
- Supports complex data structures
- A single interoperability standard

A major drawback to OPC-UA is that the lifetime of industrial equipment is typically longer than the period since OPC-UA was published. A lot of equipment currently in place was only designed for OPC Classic.

OPC in General

- **Client-server model**
  - Process is the server
  - Regulator is the client
- An OPC server may be thought of as a database
- An OPC client can retrieve data (measure) and save data (actuate) on the server

Matlab and Python Minimal Client Examples

```
% Connect to OPC-UA server
ua = opcua('opc.tcp://kt-pr-4tank:4840/four_tank_process/');
connect(ua);
% Browse namespace to find IW04 (FT1) and QW04 (P1)
% The ua.browseNamespace function opens a GUI where the
% namespace may be browsed.
S7200    = findNodeByName(ua.Namespace,'S7200.OPCServer');
Microwin = findNodeByName(S7200,'Microwin');
CPU224   = findNodeByName(Microwin,'CPU224');
IW04     = findNodeByName(CPU224,'IW04');
QW04     = findNodeByName(CPU224,'QW04');
% Read value of IW04
readValue(IW04)
% Write value to QW04
writeValue(QW04,0);
% Disconnect
disconnect(ua);
```

```
from opcua import Client
# Connect to OPC-UA server
ua = Client('opc.tcp://kt-pr-4tank:4840/four_tank_process/')
ua.connect()
# Browse namespace to find IW04 (FT1) and QW04 (P1)
# You may browse the namespace in a
# GUI using the opcua-client software.
objects = ua.get_root_node().get_child('0:Objects')
S7200 = objects.get_child('2:S7200.OPCServer')
Microwin = S7200.get_child('2:Microwin')
CPU224 = Microwin.get_child('2:CPU224')
IW04 = CPU224.get_child('2:IW04')
QW04 = CPU224.get_child('2:QW04')
# Read value of IW04
print(IW04.get_value())
# Write value to QW04
QW04.set_value(0)
# Disconnect
ua.disconnect();
```

Testing Before Commissioning Using Process Emulator

You can make your own OPC server

- Wrap process simulator in OPC-UA server with structure identical to the actual plant
- Implement functionality such as PID loops, sensor/actuator calibration and dynamics in the OPC server

The emulator allows risk-free testing of regulators which communicate using OPC-UA before commisioning

If a regulator works as expected when applied on the test OPC-UA server then there is a good chance that it will also work when implemented on the actual process

Implementation of MPC on Four Tank Process

- The four tank process offers data access at three layers:
  1. PLC
  2. OPC Classic Server
  3. OPC-UA Server (proxy to OPC Classic Server)
- Regulator may be implemented using open source software

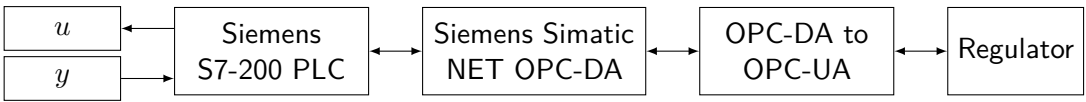

Figure: Communications between process equipment and regulator.

A simulation OPC-UA server which emulates the actual four tank process is implemented in Python to aid the development of regulators communicating using the OPC-UA standard

Implementation of MPC on Four Tank Process

Demonstration

Eskild Schroll-Fleischer
esksch@dtu.dk
and
John Bagterp Jøgensen
jbjo@dtu.dk

DTU Compute
Technical University of Denmark