POLITECNICO
MILANO 1863

Modelica-based Digital Twin for the Italian Natural Gas Network Infrastructure

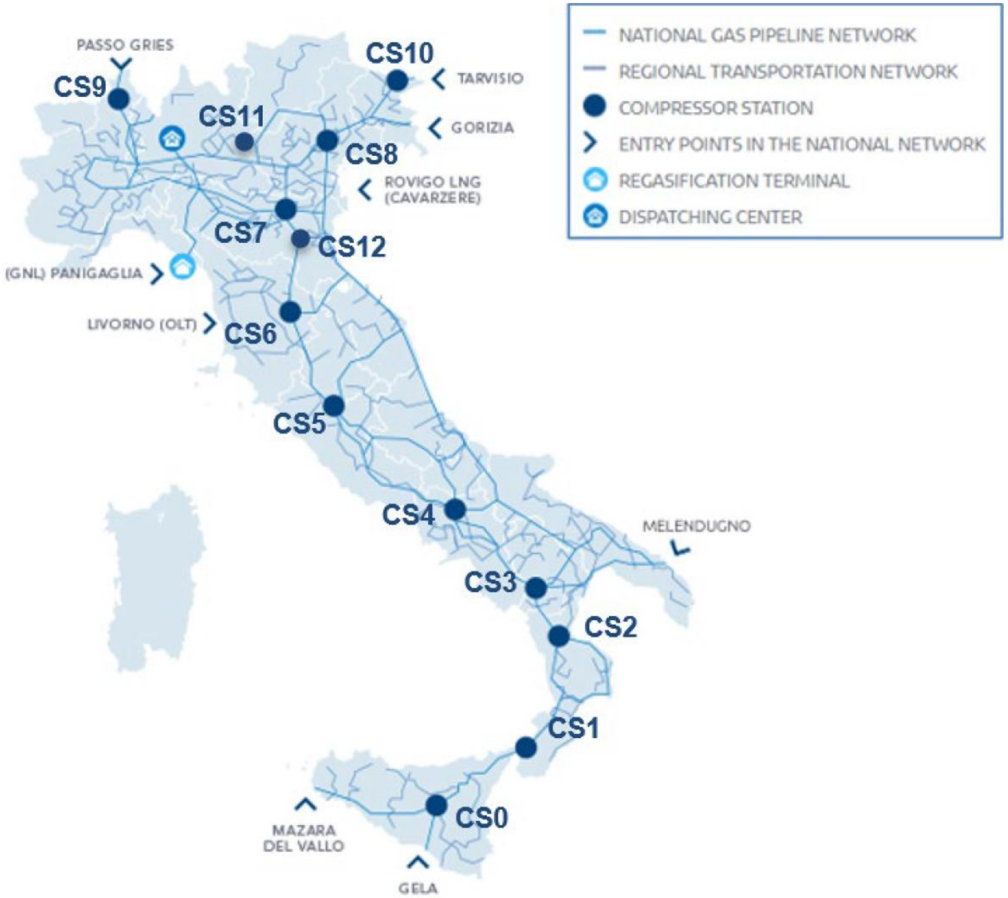
OpenModelica 2026 – February 2, 2026

PhD student Sergio Milo - sergio.milo@polimi.it

Prof. Francesco Casella - francesco.casella@polimi.it

Outline

- Problem setting and motivation
- Methodology used
- FMU model
- Filtering (Discrete Time Extended Kalman Filtering)
- Results
- Conclusion and future work



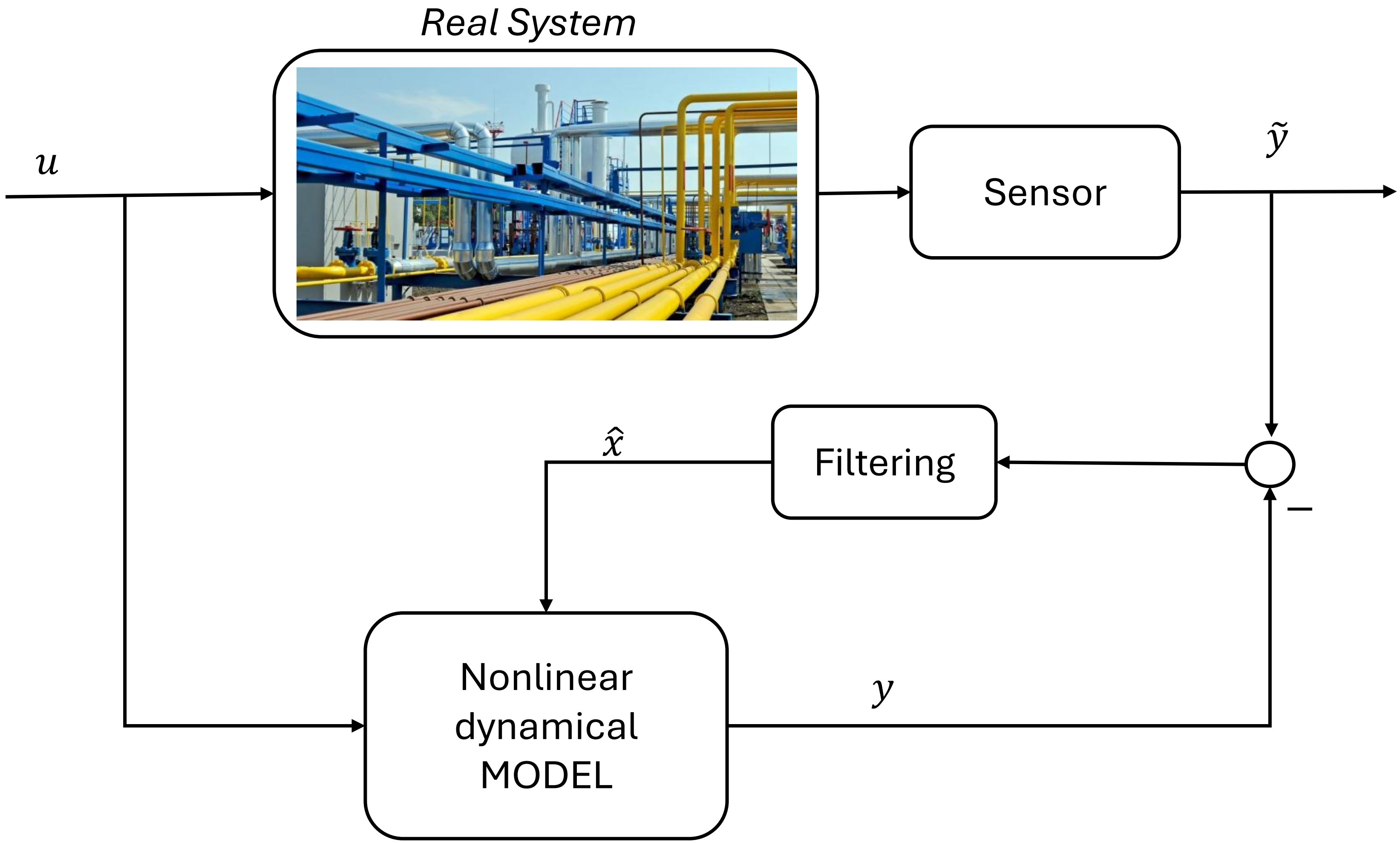
- ~ 10 000 km of high pressure network (40 000 km total)
- 13 compression stations for a total 960 MW
- ~ 2000 valves
- ~ 130 control valves
- 62 billion of Scm of gas transported in a year (Snam, 2026)
- Natural gas is 40% of total energy supply in Italy (IEA, 2024)

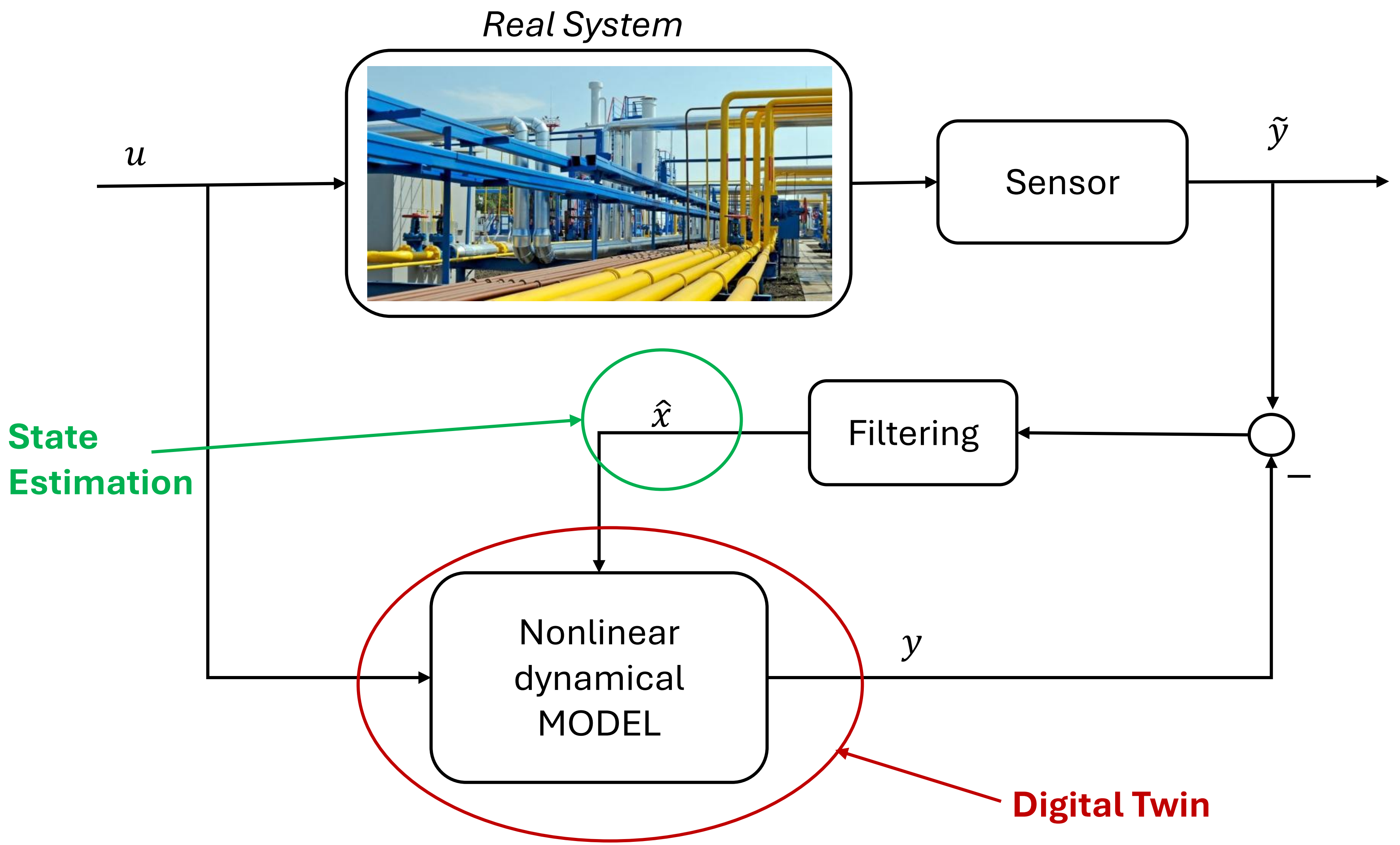
Snam S.p.A. is the Italian Transmission System Operator (TSO) for Natural Gas

The objective:

Development of a system for measurement reconciliation, state estimation, and real-time monitoring (Digital Twin) capable of determining the presence of faulty sensors throughout the entire network.

General Concept



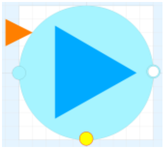

Real System
u
Sensor
$\tilde{y}$
State
Estimation
$\hat{x}$
Filtering
$-$
Nonlinear
dynamical
MODEL
y
Digital Twin

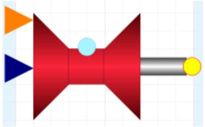
“GasNetworks” Modelica Library

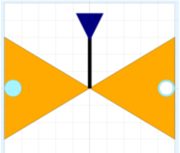
The Modelica simulation library “GasNetworks” has been developed (De Pascali et al. (2022)).

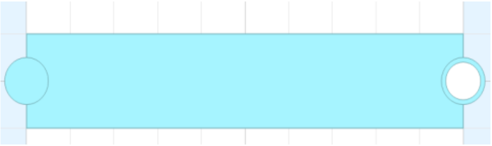
It contains the model of all components present in the network:

- Centrifugal Compressor
- Gas Turbine
- Control Valve
- Pipe







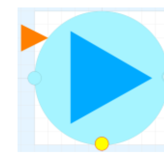


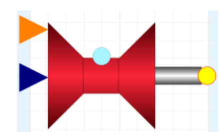
“GasNetworks” Modelica Library

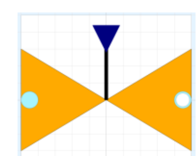
The Modelica simulation library “GasNetworks” has been developed (De Pascali et al. (2022)).

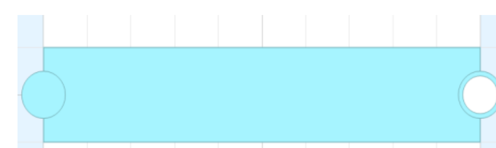
It contains the model of all components present in the network:

- Centrifugal Compressor
- Gas Turbine
- Control Valve
- Pipe





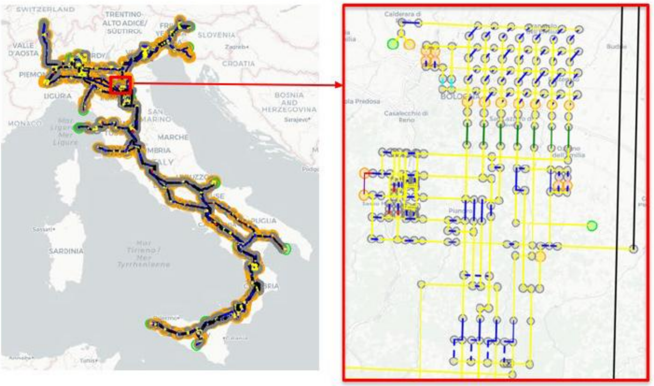




Unfortunately, this library is not open-source yet 😰

Snam’s raw dataframe

Create a graph in Python with NetworkX



Reduction Algorithm

Automatic generation of .mo file defining the components topology

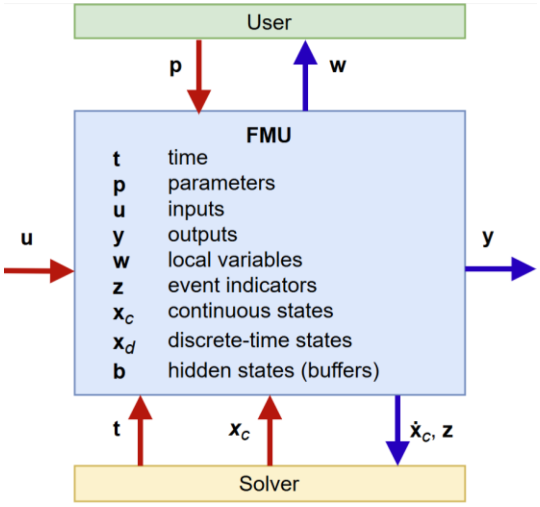
```python
## write model declaration
pipeDecl = f'  {pipeModel} pipe{edgeid:05}({gasModelPipe}, {coefficients}, {initialization});\n'
file.write(pipeDecl)
```

```python
file.write(f'  connect(pipe{edgeid:05}.inlet, node{inletNode:05}.port);\n')
file.write(f'  connect(pipe{edgeid:05}.outlet, node{outletNode:05}.port);\n\n')
```

Methodology

Two models:

1. **Time invariant** → will be exported as FMU and used by the filter with external inputs
2. **Time variant** → will contain all the signals of the network and will serve as reference for the validation of the digital twin



Discrete-time formulation is required so we chose **Model Exchange** to control the whole discretization loop and to access Jacobians needed for the Extended Kalman Filter.

INPUTS

$\boldsymbol{u}$

- Gas flows import and export
- Internal gas consumption
- Flow and pressure SP of control valves
- Activation status of compressors
- RPM of active compressors

STATES

$\boldsymbol{x}$

- Pressure along pipelines
- Internal states of the valve controllers

FMU MODEL

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \\ \boldsymbol{y} = \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}) \end{cases}$$

OUTPUTS

$\boldsymbol{y}$

- Pressure measurements
- Mass flow rate measurements

INPUTS
$\boldsymbol{u}$

- Gas flows import and export
- Internal gas consumption
- Flow and pressure SP of control valves
- Activation status of compressors
- RPM of active compressors

STATES
$\boldsymbol{x}$

- Pressure along pipelines
- Internal states of the valve controllers

OUTPUTS
$\boldsymbol{y}$

- Pressure measurements
- Mass flow rate measurements

FMU MODEL

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \\ \boldsymbol{y} = \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}) \end{cases}$$

Also

$$\boldsymbol{A}(\boldsymbol{t}) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}},\ \boldsymbol{B}(\boldsymbol{t}) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}},\ \boldsymbol{C}(\boldsymbol{t}) = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}},\ \boldsymbol{D}(\boldsymbol{t}) = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{u}}$$

INPUTS
$\boldsymbol{u}$

- Gas flows import and export
- Internal gas consumption
- Flow and pressure SP of control valves
- Activation status of compressors
- RPM of active compressors

STATES
$\boldsymbol{x}$

- Pressure along pipelines
- Internal states of the valve controllers

OUTPUTS
$\boldsymbol{y}$

- Pressure measurements
- Mass flow rate measurements

FMU MODEL

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \\ \boldsymbol{y} = \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}) \end{cases}$$

Not strictly proper system

FMU model

INPUTS
$\boldsymbol{u}$

- Gas flows import and export
- Internal gas consumption
- Flow and pressure SP of control valves
- Activation status of compressors
- RPM of active compressors

STATES
$\boldsymbol{x}$

- Pressure along pipelines
- Internal states of the valve controllers

FMU MODEL

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \\ \boldsymbol{y} = \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}) \end{cases}$$

OUTPUTS
$\boldsymbol{y}$

- Pressure measurements
- Mass flow rate measurements

Hybrid Model

We have to manage events

Discretization

Inside FMU it's continuous time but measured data are sampled every 4 minutes (240s)

we need a **Discrete Time** formulation for the dynamical model and the filter

To this end, a **backward Euler** discretization is applied, using Newton's method to solve the implicit formula for the next state value, with an upper limit on the number of iterations.

$$x_{k+1} - x_k - \Delta t f_c(x_{k+1}) = 0$$

fixed Δt

We get: f_d, h_d, A_k, C_k

We can now apply the classical discrete time Extended Kalman theory!

Discrete Time Model:

$$x_k = f(x_{k-1},\ u_{k-1}) + w_{k-1}$$

$$y_k = h(x_k, u_k) + v_k$$

$$w_k \sim (0,\ Q_k)$$

$$v_k \sim (0,\ R_k)$$

Initialization:

$$\hat{x}_0(+) = x_0$$

$$P_0(+) = P_0$$

Prediction:

$$P_k(-) = A_{k-1}\, P_{k-1}(+)\, A_{k-1}^T + Q_{k-1}$$

$$\hat{x}_k(-) = f(\hat{x}_{k-1}(+), u_{k-1})$$

Update:

$$K_k = P_k(-)\, C_k^T \left(C_k\, P_k(-)\, C_k^T + R_k\right)^{-1}$$

$$\hat{x}_k(+) = \hat{x}_k(-) + K_k\,[y_k - h(\hat{x}_k(-), u_k)]$$

$$P_k(+) = (I - K_k\, C_k)\, P_k(-)$$

It is possible to estimate not only the network states but also additional variables

Bias Sensor Error Estimation

If we suppose that the bias error enters the output equation as an additive term:

$$y_k = h(x_k, u_k) + v_k + \beta_k$$

Bias term β_k

with

$$\beta_k = \beta_{k-1} + w_{k-1}^{\beta}$$

then

$$x_k^{en} = \begin{bmatrix} x_k \\ \beta_k \end{bmatrix}$$

$$A_k^{en} = \begin{bmatrix} A_k & 0 \\ 0 & I \end{bmatrix},$$

$$C_k^{en} = \begin{bmatrix} C_k & I \end{bmatrix}$$

Physical network dynamic are very slow: the pressure distribution within the pipelines has time constants on the order of several hours. Hammelmann et al. (2023)

A natural choice would be $\Delta t = 240s$ in the Euler discretization since measurement data are collected every 4 minutes ($240s$)

$$x_{k+1} - x_k - \Delta t f_c(x_{k+1}) = 0 \qquad \textbf{fixed } \boldsymbol{\Delta t}$$

Physical network dynamic are very slow: the pressure distribution within the pipelines has time constants on the order of several hours. Hammelmann et al. (2023)

→ A natural choice would be $\Delta t = 240s$ in the Euler discretization since measurement data are collected every 4 minutes ($240s$)

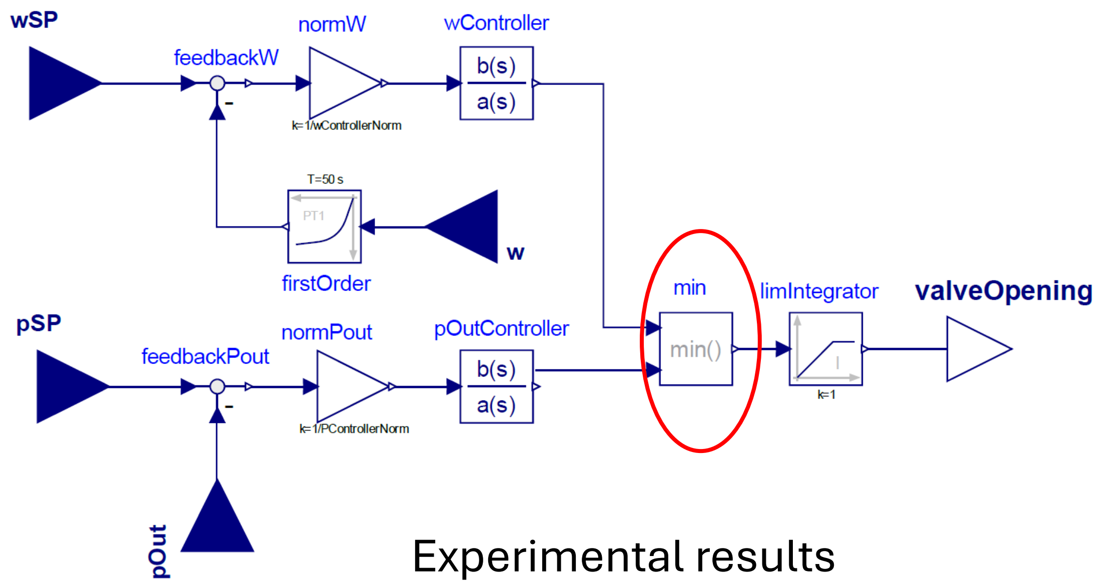
$$x_{k+1} - x_k - \Delta t f_c(x_{k+1}) = 0 \qquad \textbf{fixed } \boldsymbol{\Delta t}$$

BUT SOLVER FAILS

Newton's iterations fail to converge. Why?

Solver Convergence

Controls of control valves



Fast dynamics and very nonlinear

Problem for the numerical integration if Δt is too large

Experimental results

- $\Delta t = 240s$ Simulation failure
- $\Delta t = 120s$ Convergence limit
- $\boldsymbol{\Delta t = 80s}$ **Robust convergence**

Three points per measurement

Validation up to now has been performed **only on synthetic data** suitably modified to introduce (known!) error sources, mimicking those expected in reality. We considered:

White Noise

$\mathcal{N}(\mu, \sigma^2)$ with $\mu = 0$ and $\sigma = 0.2 * 10^5$ Pa (0.2 bar) for pressure measurements
with $\mu = 0$ and $\sigma = 1$ Kg/s for mass flow rate measurements

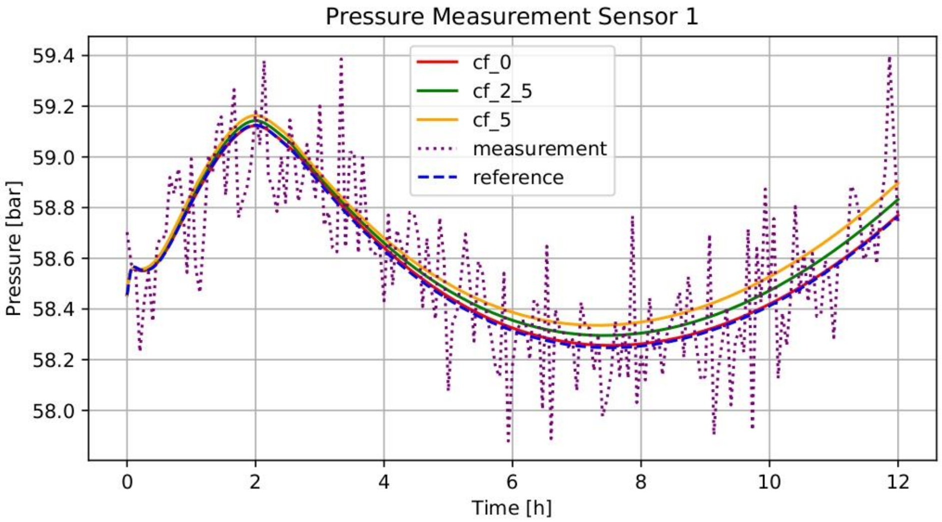
Constant Bias Error

$\beta = 1 * 10^5$ Pa (1 bar) in specific pressure measurement sensors

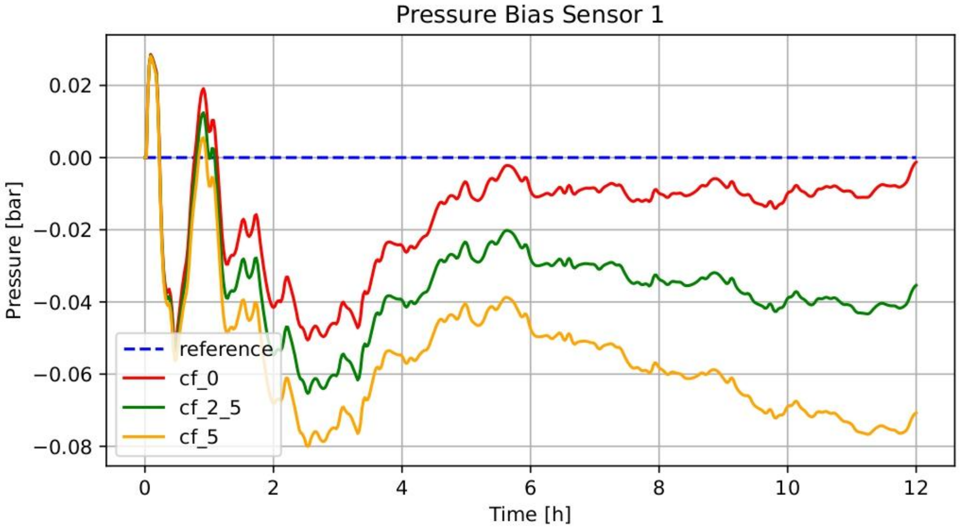
Parametric error on friction coefficient

$p_{out} - p_{in} = \boldsymbol{c_f} \rho \omega \frac{u^2}{2} LA - \rho g (z_{out} - z_{in})$ Error on $\boldsymbol{c_f}$ of 0% , 2.5% and 5%

Pressure Estimation

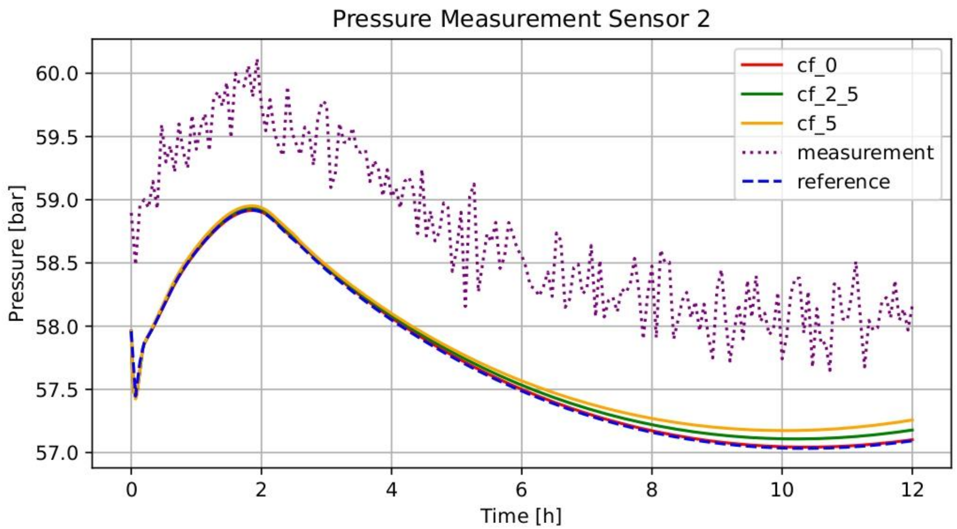


Bias Estimation

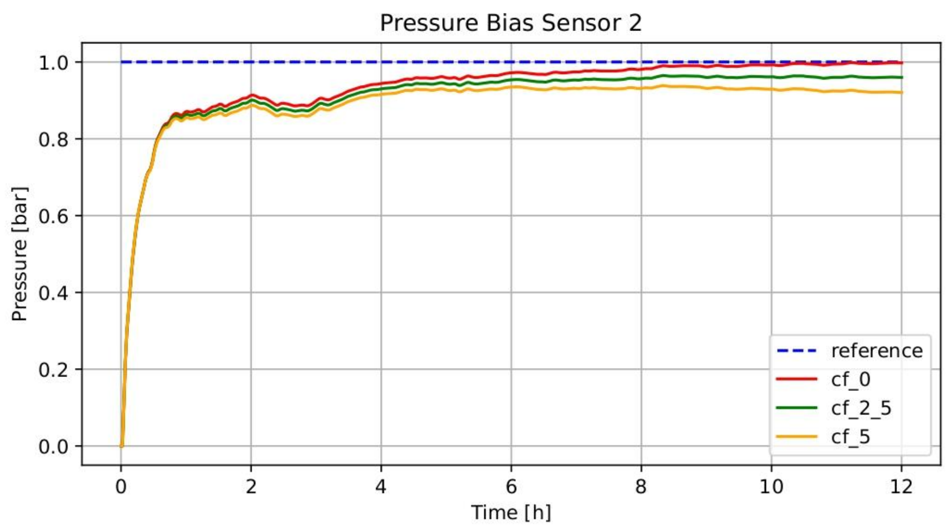


Results

Pressure Estimation

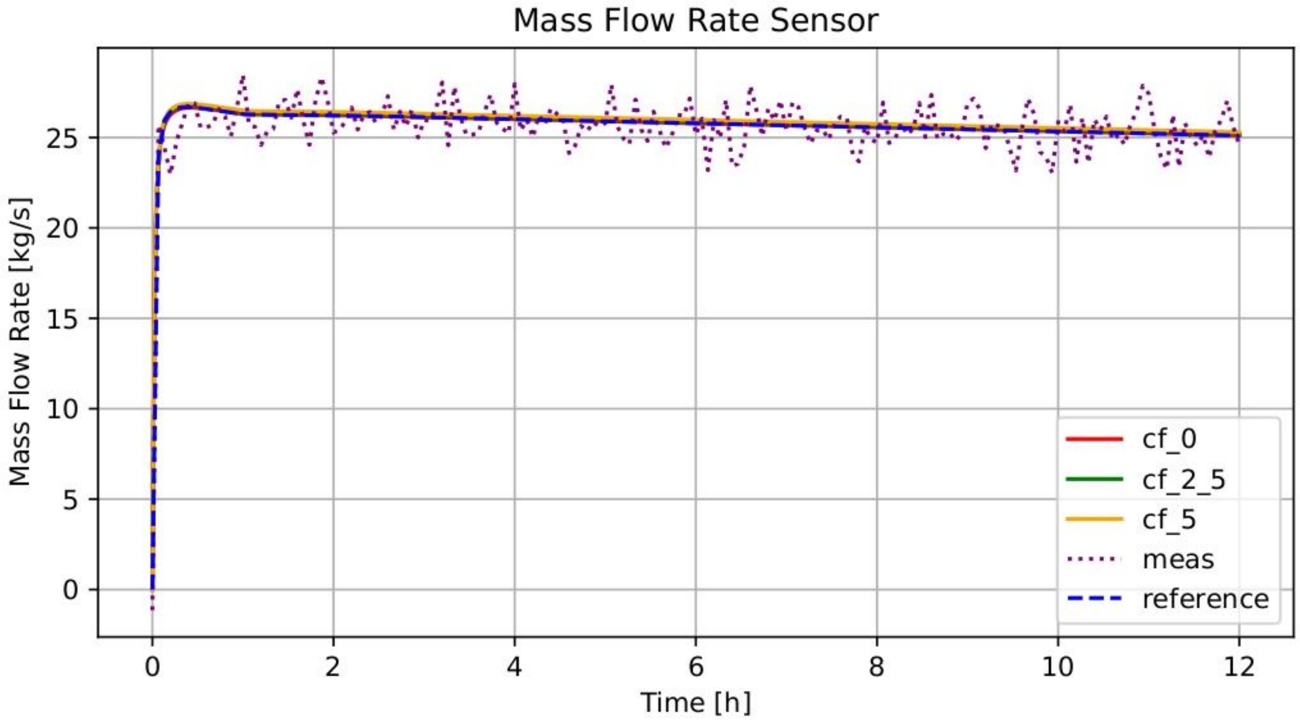


Bias Estimation



Fault detection and fault estimation!

Mass flow rate Estimation



Conclusion and Future Work

- The proposed methodology can generate a digital twin of a real gas network to filter noisy field measurements and reconstruct the system state, with the possibility, through state augmentation, of detecting sensors that have an offset.
- It has been shown that the digital twin behaves robustly even in the presence of model uncertainties.
- Potential for real-time applications
- So far, validation has been performed only on synthetic data.
- Future work will focus on testing the methodology with real SCADA data (every 4 minutes).
- Other types of state augmentation could be considered based on the operator’s needs.

Conclusion and Future Work

This work has been submitted to IFAC WC 2026

IFAC 2026 BUSAN

The project is conducted using only open-source software

OpenModelica

python™

fmi Functional Mock-up Interface

CATIA-Systems/
FMPy

Simulate Functional Mock-up Units (FMUs) in Python

7 Contributors | 120 Issues | 555 Stars | 139 Forks

1. FMU export in C → memory leaks problems
2. FMU export in C++ → Initialization problem, the event iteration during `when initial()` cannot handle more than one iteration
3. FMU C++ was not working with PyFMI
4. Now the FMU C++ runs smoothly in FMPy with the correct initialization 😜

Thank you for your attention!