End-to-End Process Monitoring: Workshop

L Auret*[*,**] & TM Louw[**]

[*]Stone Three; [**]Stellenbosch University

Agenda

08h30 - 09h00: Welcome and introductions

09h00 - 10h00: (Presentation) Overview: Motivation and framework structure

10h00 - 10h30: (Interactive session) Computer setup and repository familiarization

10h30 - 11h00: Coffee break

11h00 - 12h00: (Presentation) Framework module details

12h00 - 12h30: (Interactive example) Framework familiarization and experiments

12h30 - 13h30: **Lunch**

13h30 - 14h30: (Presentation) Framework module details (continued)

14h30 - 15h30: (Interactive example) Framework familiarization and experiments

15h30 - 16h00: Coffee break

16h00 - 17h30: (Interactive challenge) Custom configurations and modules, improve process monitoring performance

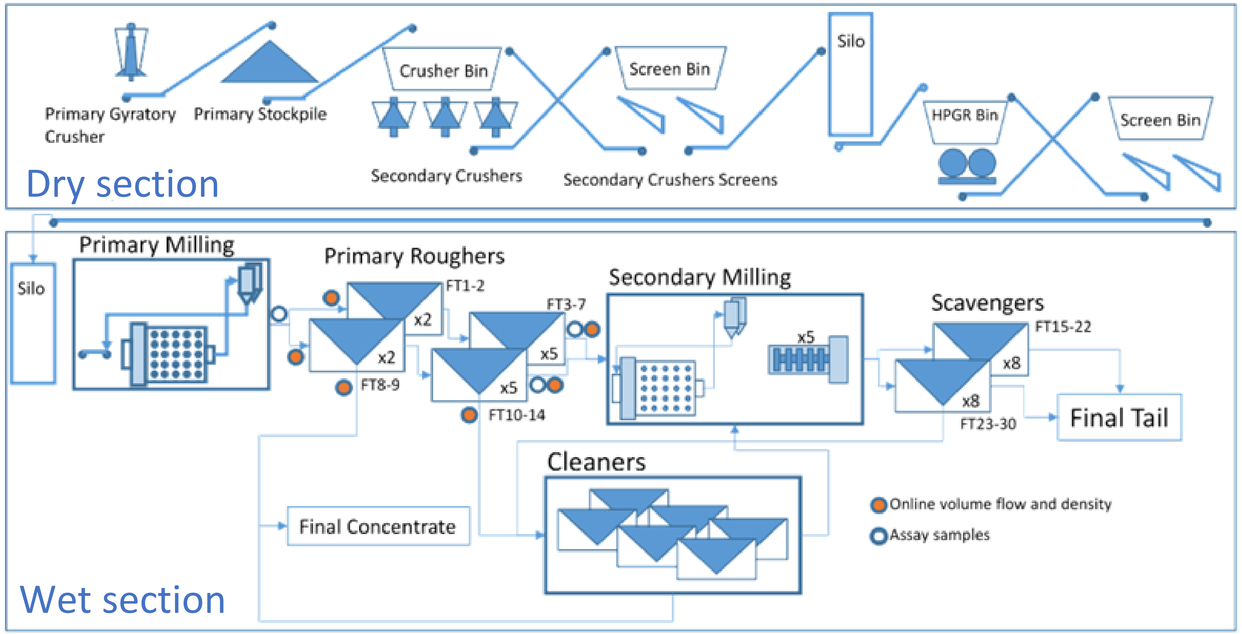
Overview

Value Generation in Industry: Mineral Processing Example

600 000 tons per month

Feed: 3 to 8 * (PGM g)/ton [$ 160 / ton]

Ore size: ~ 1 m



Concentrate: 200 to 2000 * (PGM g)/ton [$ 31 000 / ton]

Particle size: 10 to 100 microns

Mining company wide (concentrators, smelters, refineries)

Safety: Case frequency rates 2.6 per million hours

Environment: GHG emissions 4.5 M tons/year

Environment: Energy use 21 000 TJ/year

Environment: Water withdrawal 43 000 ML/year

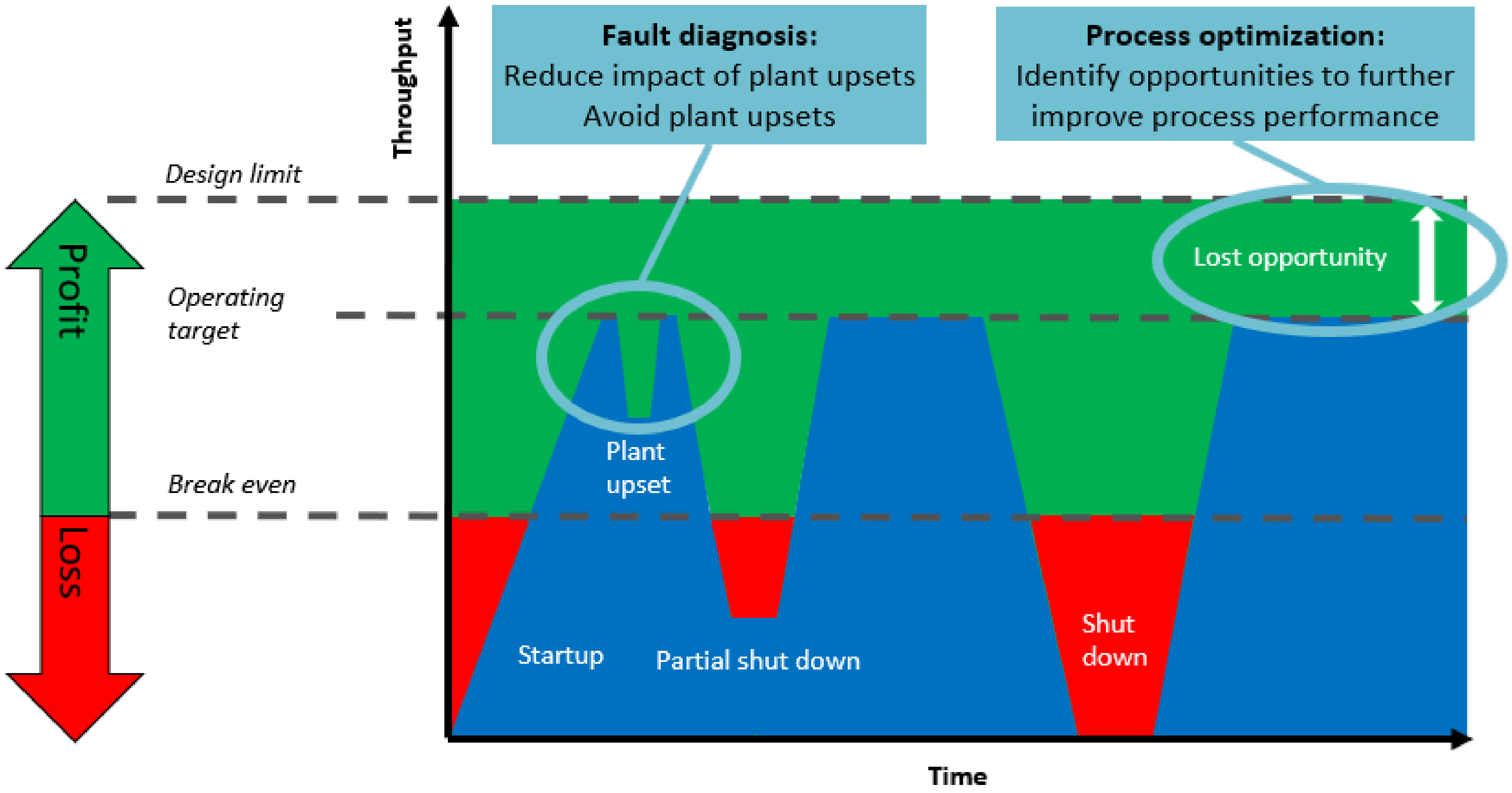
** Typical values - Mpinga et al., 2015. Direct leach approaches to PGM ores and concentrates: A review. Min Eng, 78.*

Steyn and Sandrock, 2021. Causal model of an industrial platinum flotation circuit. Con Eng Prac. 109, 104736.

Operational statistics from public financial results

Process Monitoring Opportunities

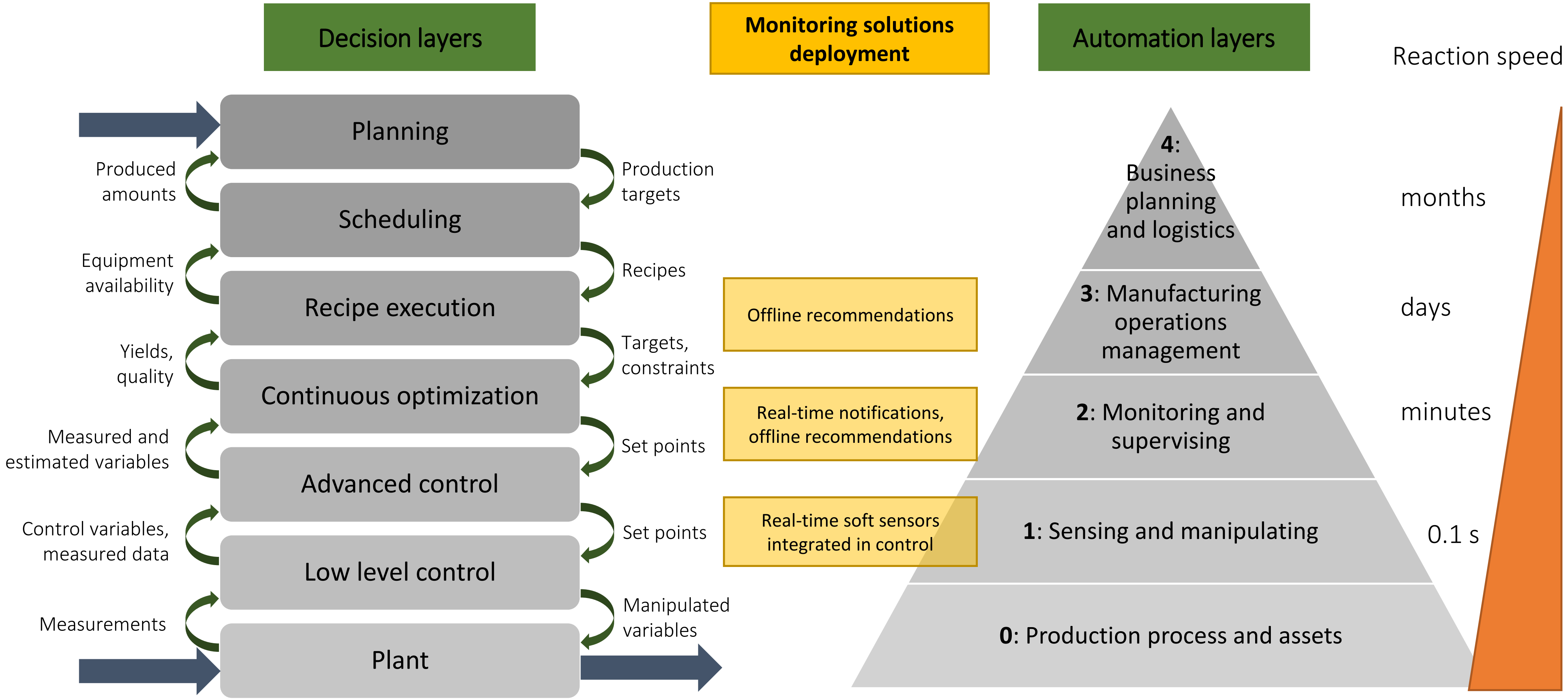
Fault diagnosis and optimization



Sand and Terwiesch, 2013. Closing the loops: An industrial perspective on the present and future impact of control. Euro J. Control. 19, 341-350.

Process Monitoring Opportunities

Automation hierarchy



Isaksson, Harjunkoski and Sand, 2018. The impact of digitalization on the future of control and operations. Comp Chem Eng. 114, 122-149.

Process Monitoring Tasks

Fault diagnosis

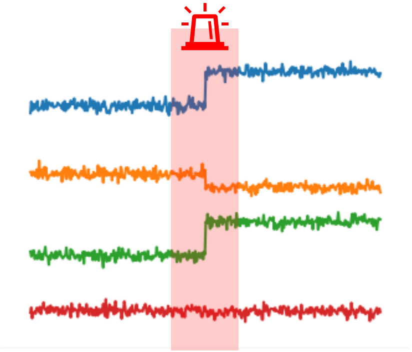
Fault detection
Determine whether fault has occurred

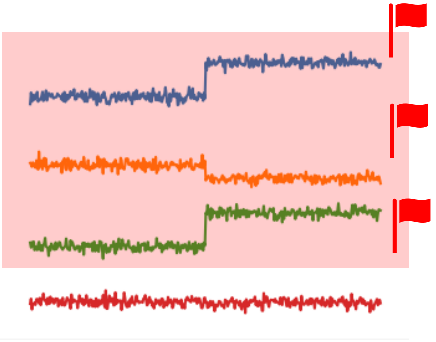
Yes / No

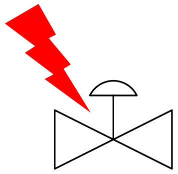
Fault identification
Identify most relevant observed measurements

Fault diagnosis
Determine fault occurrence (type, location, magnitude)

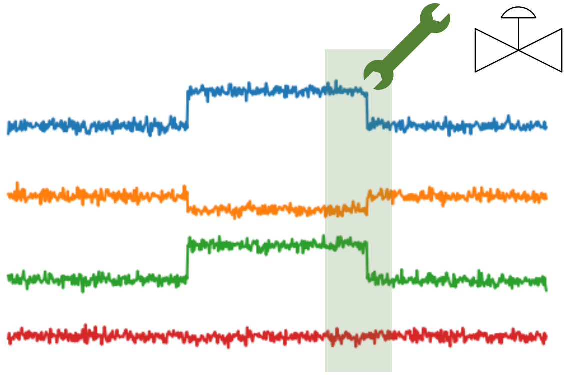
Process recovery
Applies intervention to remove fault effect







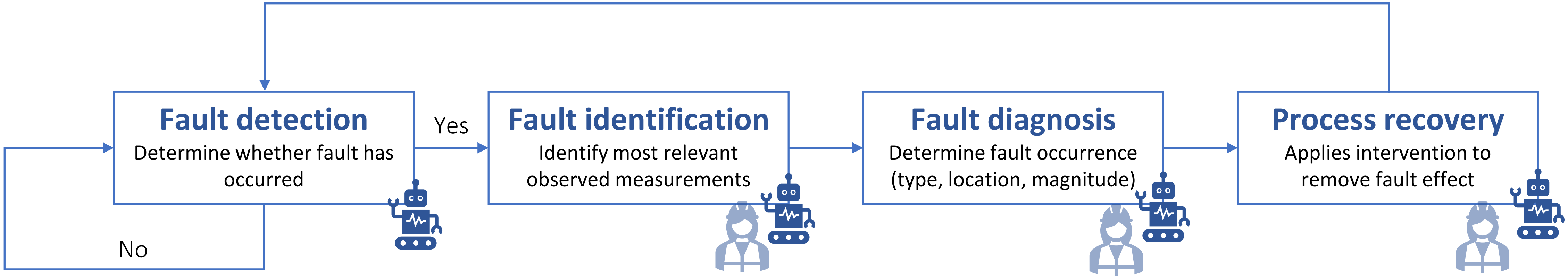
Valve 1 stuck
100% open



Performance metrics	False/missing alarm rates Detection delays	Identification accuracy *Not well-established*	Diagnosis accuracy *Not well-established*	Process key performance indicators *Not well-established*

Inspired by: Bi et al. (2022) One step forward for smart chemical process fault detection and diagnosis. Computers and Chemical Engineering.

Process Monitoring Tasks



End-to-end process monitoring (E2E-PM):
Complete set of <u>automated algorithms</u> to link process monitoring tasks and generate appropriate process recovery interventions exactly when required

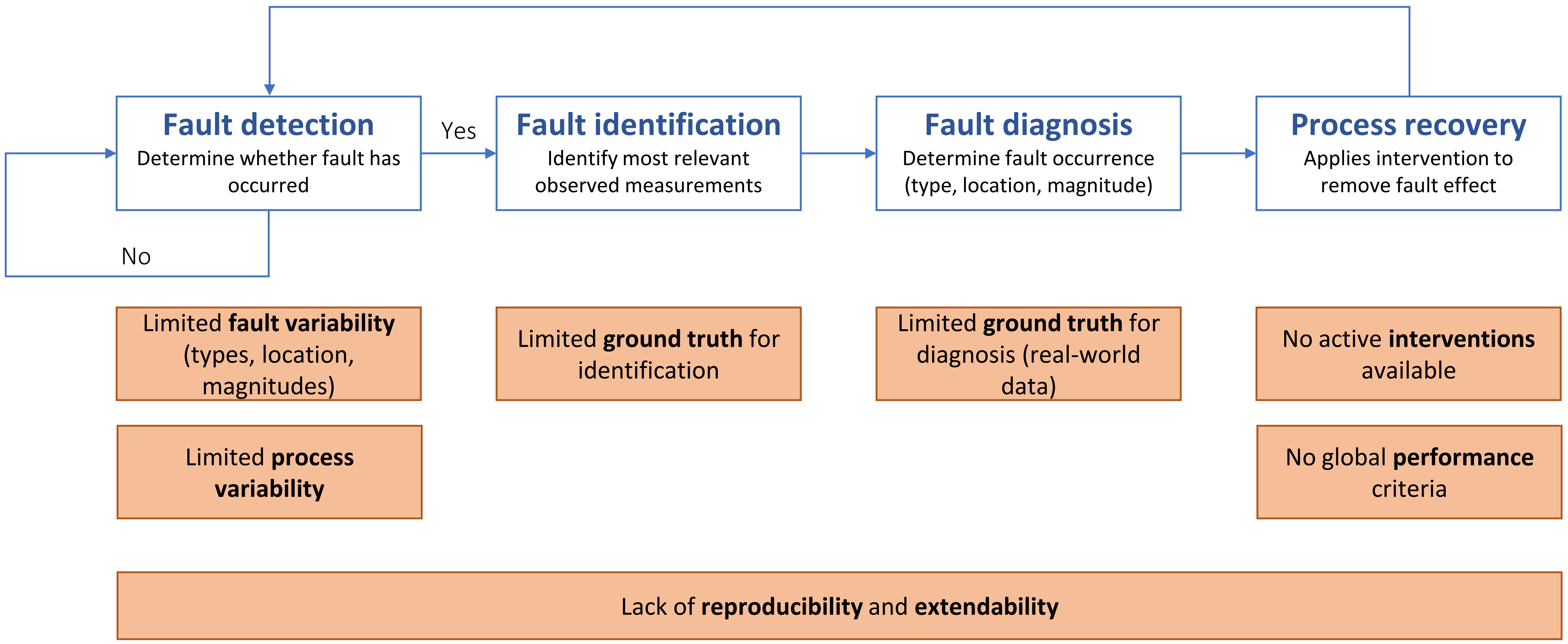
Related work:	Fault Tolerant Control + Data-Driven Process Monitoring	Prognostics and Health Management	Reinforcement Learning Industrial Task Suite

MacGregor & Cinar (2012). Monitoring, fault diagnosis, fault-tolerant control and optimization: Data-driven methods.

Zio, E. (2022). Prognostics and health management (PHM): Where are we and where do we (need to) go in theory and practice.

Chervonyi et al. (2022). Semi-analytical industrial cooling system model for reinforcement learning.

Process Monitoring Research Challenges



Process Data Challenges

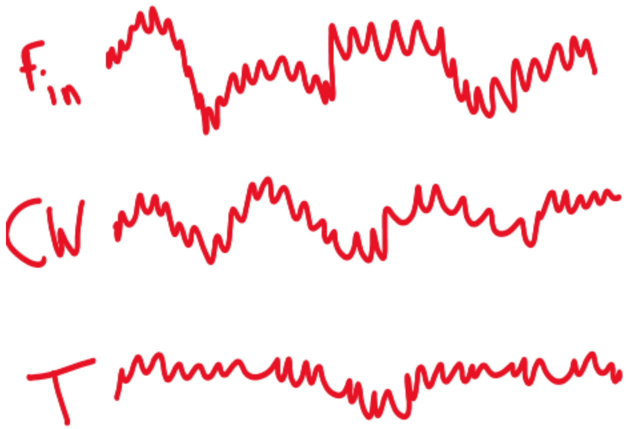
Process data characteristics

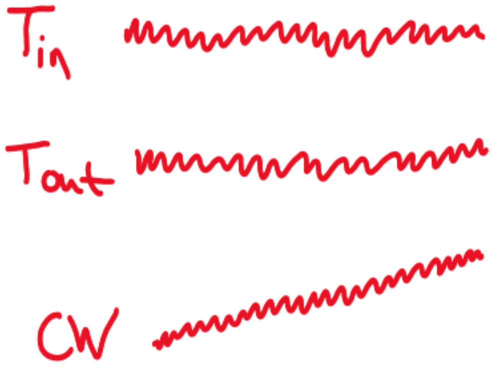
Dynamic
- Plant does not operate at fixed values
- Random and systematic disturbances

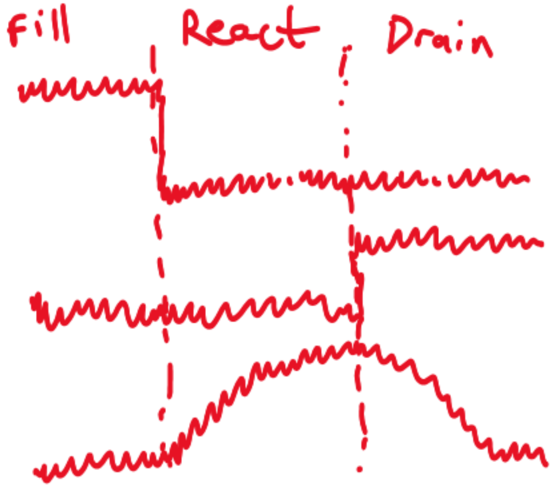
Time-varying
- Gradual changes in process parameters, e.g., due to degradation

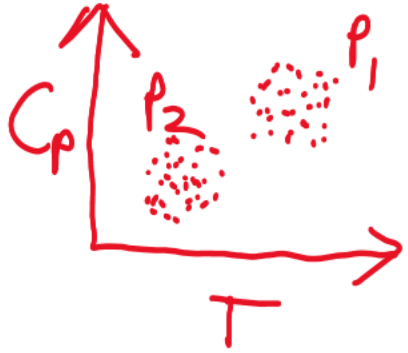
Batch vs continuous
- Batch process = recipe executed over time

Multimode
- Switching between recipes changes distribution of data









Kumar and Flores-Cerrillo (2022) – Machine Learning in Python for Process Systems Engineering. Achieve operational excellence using process data.

Process Data Challenges

Process data characteristics

Discrete/discontinuous

- Equipment switched on/off causing step changes

Nonlinear

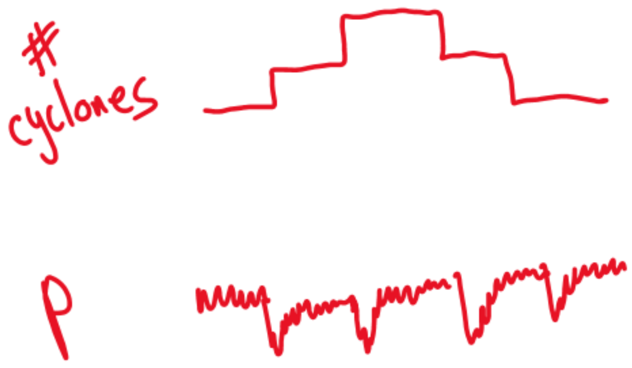
- Chemical and physical laws cause nonlinear relationships

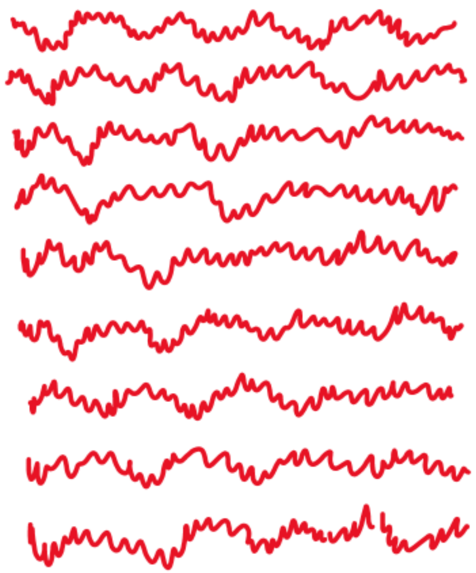
High dimensionality

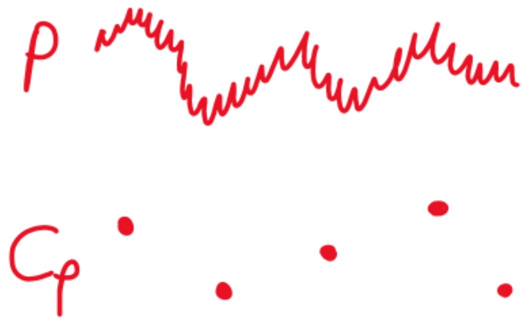
- Tens/hundreds/thousands of variables

Multi-rate sampling

- Sampling frequency of measurements differ (seconds to days)







Kumar and Flores-Cerrillo (2022) – Machine Learning in Python for Process Systems Engineering. Achieve operational excellence using process data.

Reproducibility Crisis

Data leakage and reproducibility crisis (in machine learning)

Kapoor and Narayanan (2022) found **329 papers** with machine learning application errors in 17 research fields, leading to widely optimistic conclusions

Lack of clean separation between training and test sets

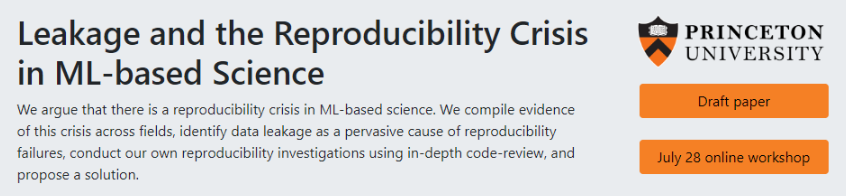
- No test set
- Pre-processing on training and test sets
- Feature selection on training and test sets
- Duplicates

Illegitimate features

- E.g., proxy for output variable included in inputs
- Domain-specific

Test set not representative

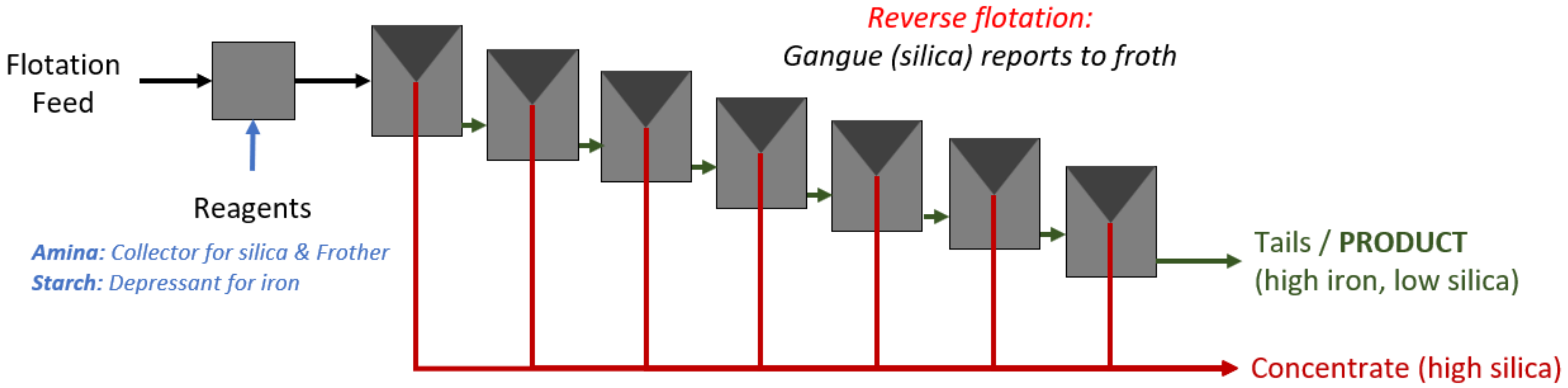
- Temporal leakage (shuffling)
- Nonindependence between train and test sets
- Sampling bias



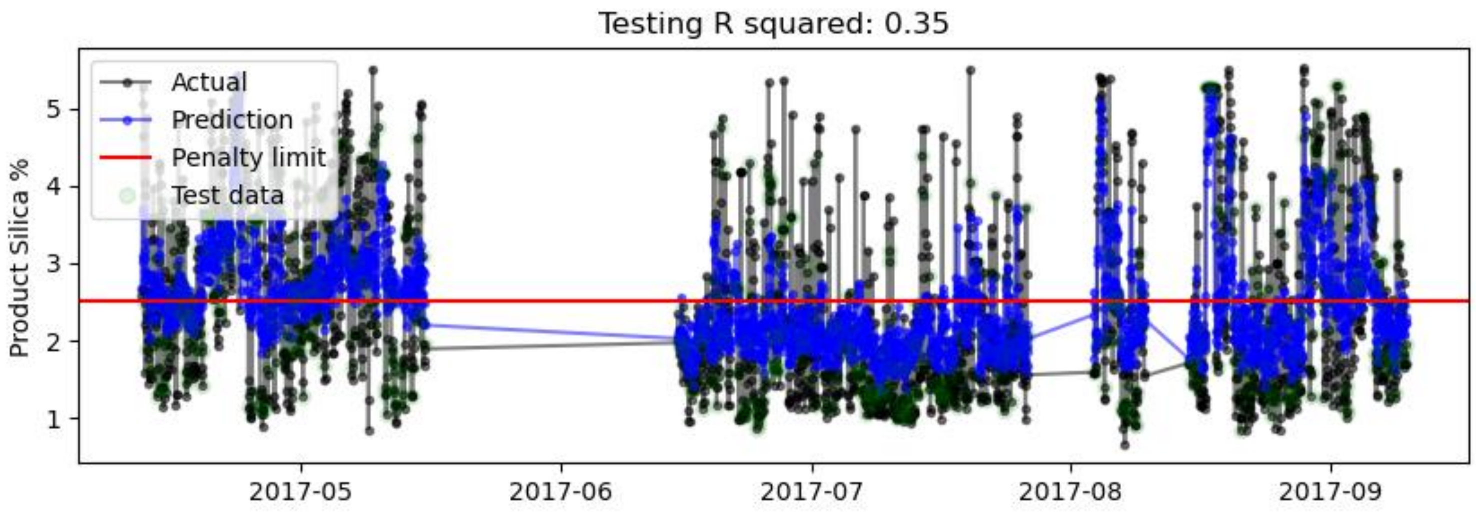
Kapoor and Narayanan (2022) Leakage and the reproducibility crisis in ML-based science. [*https://arxiv.org/abs/2207.07048*](https://arxiv.org/abs/2207.07048)

Reproducibility Crisis

Data leakage and reproducibility crisis (in machine learning)



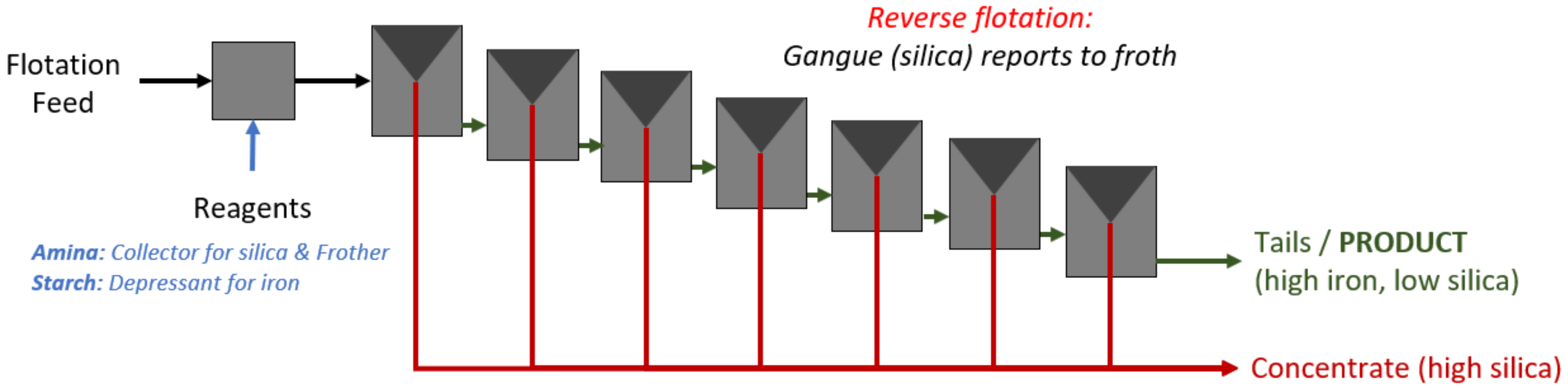
Shuffled training/testing data, gradient boosting model, 1H median KPI



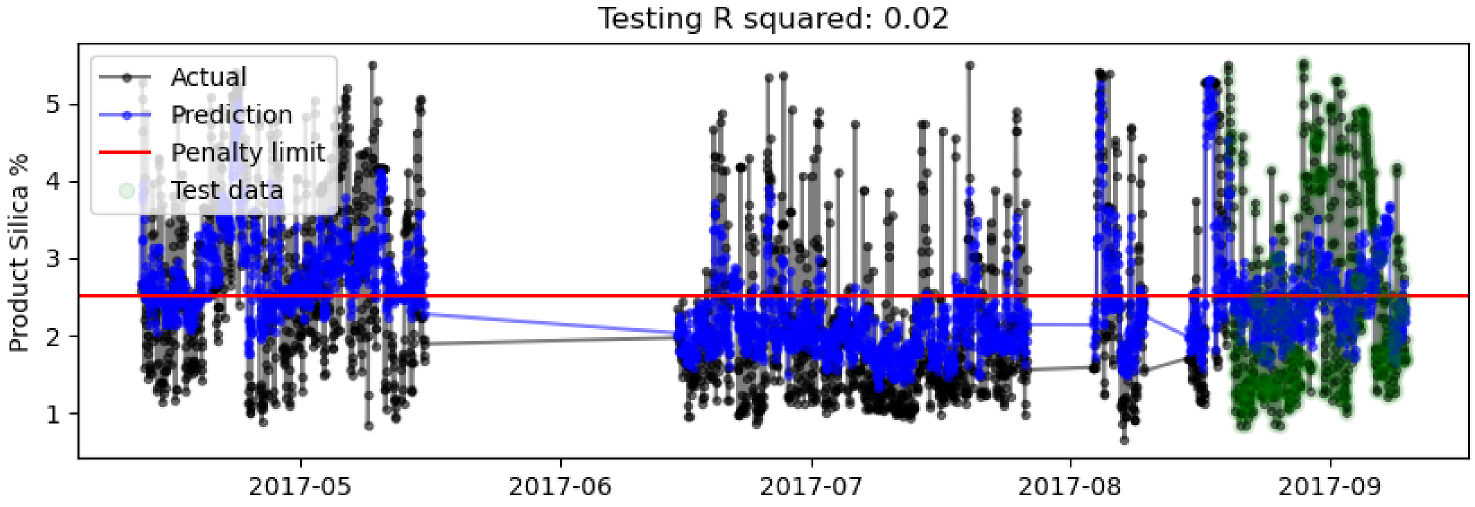
https://www.kaggle.com/edumagalhaes/quality-prediction-in-a-mining-process?select=MiningProcess_Flotation_Plant_Database.csv

Reproducibility Crisis

Data leakage and reproducibility crisis (in machine learning)



Unshuffled training/testing data, gradient boosting model, 1H median KPI



https://www.kaggle.com/edumagalhaes/quality-prediction-in-a-mining-process?select=MiningProcess_Flotation_Plant_Database.csv

Reproducibility Crisis

Publishing source code of case studies and algorithms to mitigate the crisis

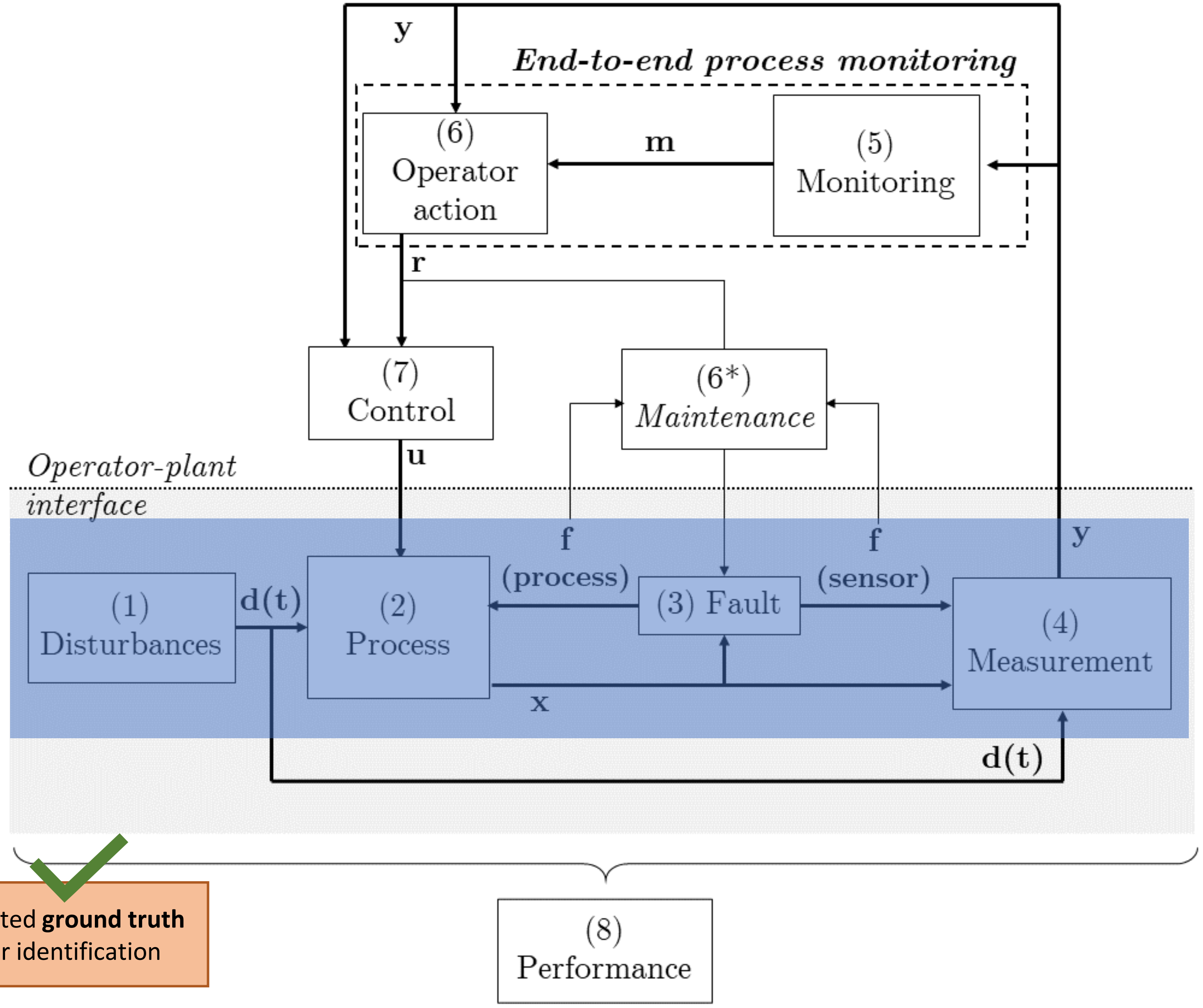
- Concerns:
  - Not achieving required code quality for publication
  - Intellectual property protection pressure
- Counter-arguments:
  - Even professional code is often badly documented, inconsistent, and poorly tested
  - If the code is good enough to produce results, it is good enough to share
  - Freely shared code does not require technical support (if the feedback is unhelpful, ignore it)
  - Value lies in (your) expertise

Barnes, N., 2010. Publish your computer code: it is good enough. Nature, October, Volume 467, p. 753–753.

End-to-End Process Monitoring Framework

Modular Case Study Design

- Test bed for E2E-PM solutions: Unbiased and effective evaluation
- 8 interacting modular components
- **Plant modules**
  - Disturbance module
    - Deterministic, stochastic, seasonal, etc.
  - Process module
    - Process dynamics (including actuation)
  - Fault module
    - Process, actuator, sensor faults
    - Condition/state-dependent, stochastic
    - Feedback to *process* and *sensors*
  - Measurement module
    - Effects of time delays, variable sampling rates, measurement noise, sensor faults



Limited **process variability**

Limited **fault variability** (types, location, magnitudes)

Limited **ground truth** for diagnosis (real-world data)

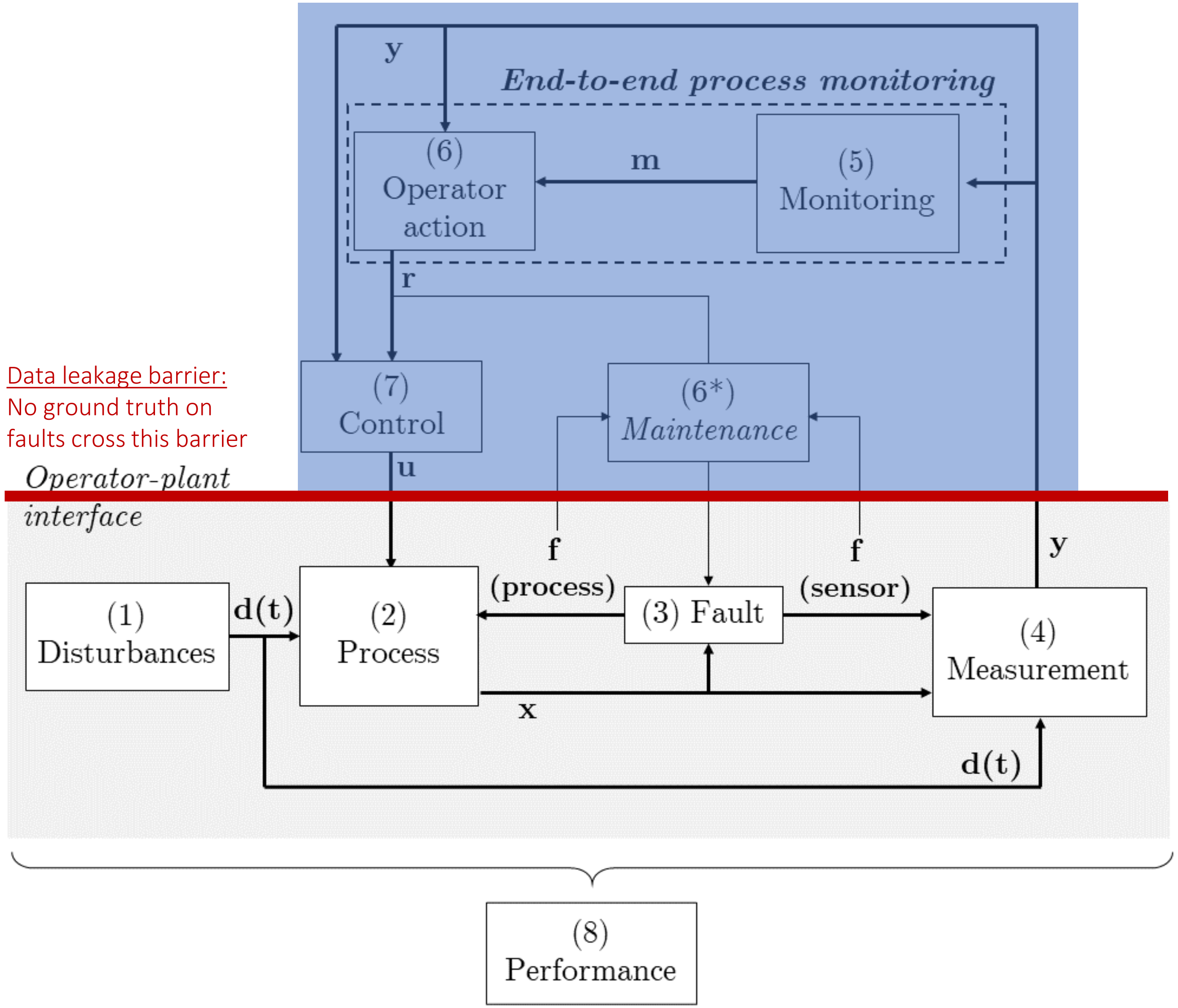
Limited **ground truth** for identification

End-to-End Process Monitoring Framework

Modular Case Study Design

- Test bed for E2E-PM solutions: Unbiased and effective evaluation
- 8 interacting modular components
- **Monitoring and intervention modules**
  - Monitoring module
    - Detection, identification, diagnosis
  - Operator action module
    - Follow-up interventions (start-up, shut down, set points adjustment, tuning, maintenance instructions)
  - Maintenance module
    - Direct interaction with fault states (inspection and replacement during plant shuts)
  - Control module
    - Automated interlocks, regulatory, supervisory, optimisation control

No active **interventions** available



Case Study

https://github.com/Stellenbosch-University-Process-Eng/End-to-end-process-monitoring

Toy Problem to Demonstrate Framework

Disturbance

Stochastically varying feed flow rate and feed concentration

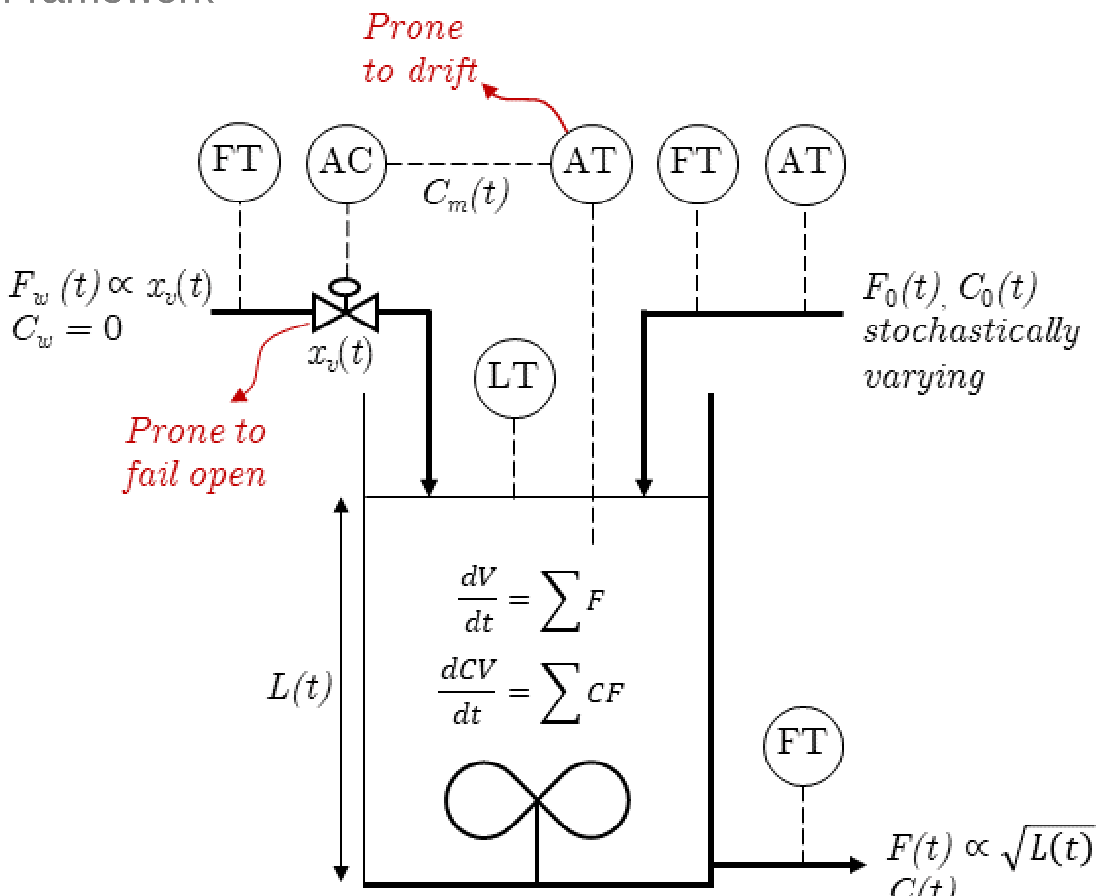
Process

Measurement

Control

Fault

1 = Control valve fail open
2 = Concentration measurement drift
Stochastic timing, bathtub curve



Monitoring

Principal Component Analysis
Training: First 7 days
Detection: Hotelling's T^2 and SPE
Diagnosis: Expert rules

Operator actions and maintenance

Different strategies:
- Planned maintenance
- Alarms inspire check, replace, maintenance timing

Performance

Profit ($/hr)
= (revenue from product on spec) – (costs from planned/unplanned maintenance)

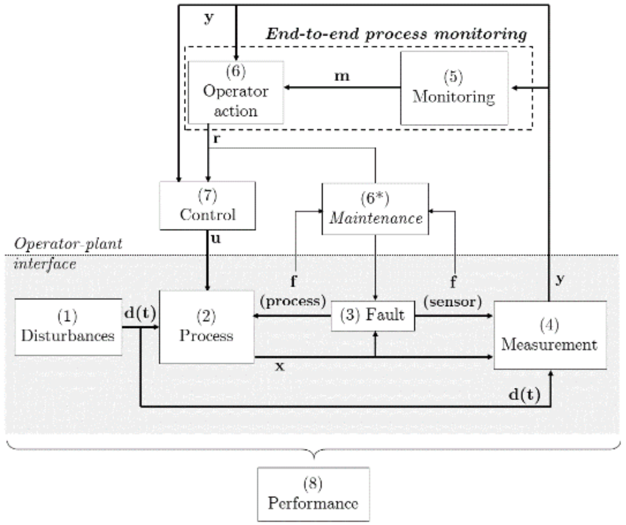
Setup and familiarization

Interactive session

Setup and familiarization

Goals and exercises

- Ensure MATLAB and/or Python is setup up
- Access repository code
  - Zip download and extraction
  - Using git
- Test running the code ‘BlendingTank_IFACWC.m’ (MATLAB) or ‘BlendingTank_IFACWC.py’ (Python)
- **Exercise**: Access simulation outputs (e.g., variables m or results DataFrame (Python)) and inspect values
- **Exercise**: Change the maximum duration of the simulation and rerun the experiment



Framework module details

Basic dynamic simulations

Disturbance model

Process model

Fault model

Measurement model

Basic dynamic simulations

Dynamic model simulation

- Typical open-loop (no control) simulation requirements for phenomenological models

Time
- Start time
- End time
- Time increments

Parameters
- Model parameters

Initial values
- Starting points
  - Output variables
  - Input variables

Inputs
- Complete specification
- Fixed for time range
- Function for time range

Dynamic model
- ODEs as function of parameters and inputs

Solver
- Numerical integration

Basic dynamic simulations

Dynamic model simulation

- Typical open-loop (no control) simulation requirements for phenomenological models

Time	Parameters	Initial values	Inputs	Dynamic model	Solver

• Start time
• End time
• Time increments
	• Model parameters	• Starting points
		  • Output variables
		  • Input variables
			• Complete specification
			• Fixed for time range
			• Function for time range
				• ODEs as function of parameters and inputs	• Numerical integration

$$\tau \frac{dy(t)}{dt} + y(t) = K_p x(t)$$

Time extent for which system should be simulated

Required: Start time, end time
Optional: Time increments result should be reported at

```python
# Time
t_span = [0,5]
t_sample = np.arange(t_span[0],t_span[1],0.1)
```

Basic dynamic simulations

Dynamic model simulation

- Typical open-loop (no control) simulation requirements for phenomenological models

Time	Parameters	Initial values	Inputs	Dynamic model	Solver

• Start time
• End time
• Time increments
	• Model parameters	• Starting points
		  • Output variables
		  • Input variables
			• Complete specification
			• Fixed for time range
			• Function for time range
				• ODEs as function of parameters and inputs	• Numerical integration

Constants used in dynamic model

$$\tau \frac{dy(t)}{dt} + y(t) = K_p x(t)$$

```python
# Parameters
tau = 0.5
K_p = 2.0
theta = [tau, K_p]
```

Basic dynamic simulations

Dynamic model simulation

- Typical open-loop (no control) simulation requirements for phenomenological models

Time	Parameters	Initial values	Inputs	Dynamic model	Solver

• Start time
• End time
• Time increments
	• Model parameters	• Starting points
		  • Output variables
		  • Input variables
			• Complete specification
			• Fixed for time range
			• Function for time range
				• ODEs as function of parameters and inputs	• Numerical integration

Starting point of simulation

May need steady-state balance ($\frac{dy}{dt} = 0$) to solve

$$\tau \frac{dy(t)}{dt} + y(t) = K_p x(t)$$

```
# Initial values
x_0 = 1.0
y_0 = [K_p*x_0]
```

Basic dynamic simulations

Dynamic model simulation

- Typical open-loop (no control) simulation requirements for phenomenological models

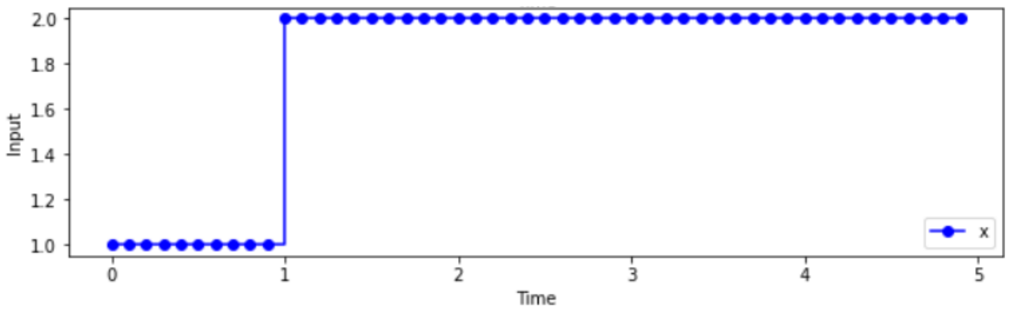
Time	Parameters	Initial values	Inputs	Dynamic model	Solver

• Start time
• End time
• Time increments
	• Model parameters	• Starting points
		  • Output variables
		  • Input variables
			• Complete specification
			• Fixed for time range
			• Function for time range
				• ODEs as function of parameters and inputs	• Numerical integration

Input / forcing function / independent variables trajectory
Open-loop simulation: Fixed trajectory
Closed-loop simulation: Controller output

$$\tau \frac{dy(t)}{dt} + y(t) = K_p x(t)$$

```
# Forcing function / input
x = np.ones(len(t_sample))*x_0
x_delta = 1.0
x[10:] = x_0+x_delta
```



Basic dynamic simulations

Dynamic model simulation

- Typical open-loop (no control) simulation requirements for phenomenological models

Time	Parameters	Initial values	Inputs	Dynamic model	Solver

• Start time
• End time
• Time increments
	• Model parameters	• Starting points
		  • Output variables
		  • Input variables
			• Complete specification
			• Fixed for time range
			• Function for time range
				• ODEs as function of parameters and inputs	• Numerical integration

$$\tau \frac{dy(t)}{dt} + y(t) = K_p x(t)$$

Ordinary differential equation(s) (ODE(s))
Required arguments: (t, y)
Optional arguments: (x, parameters)

```python
# Model
def firstOrderSystem(t,y,t_x,x,theta):
    # Determine input time
    t_ind = (np.abs(t_x-t)).argmin()
    # Determine input value
    x_now = x[t_ind]
    # Unpack parameters
    tau, K_p = theta
    # ODE
    dydt = (K_p/tau)*x_now - (1/tau)*y
    return dydt
```

Basic dynamic simulations

Dynamic model simulation

- Typical open-loop (no control) simulation requirements for phenomenological models

Time	Parameters	Initial values	Inputs	Dynamic model	Solver

• Start time
• End time
• Time increments
	• Model parameters	• Starting points
		  • Output variables
		  • Input variables
			• Complete specification
			• Fixed for time range
			• Function for time range
				• ODEs as function of parameters and inputs	• Numerical integration

$$\tau \frac{dy(t)}{dt} + y(t) = K_p x(t)$$

Numerical integration

For each time step, evaluates $\frac{dy}{dt}$

Next y value calculated based on $\frac{dy}{dt}$:

$$y(k+1) = y(k) + \Delta y$$

Next time step determined $t + \Delta t$

This may be iterative: varying Δt to ensure numerical accuracy

```
# Integration
result = solve_ivp(firstOrderSystem,
                   t_span, y_0,
                   args=(t_sample,x,theta),
                   t_eval=t_sample)
```

Basic dynamic simulations

Dynamic model simulation

- Typical open-loop (no control) simulation requirements for phenomenological models

Time	Parameters	Initial values	Inputs	Dynamic model	Solver

• Start time
• End time
• Time increments
	• Model parameters	• Starting points
		  • Output variables
		  • Input variables
			• Complete specification
			• Fixed for time range
			• Function for time range
				• ODEs as function of parameters and inputs	• Numerical integration

$$\tau \frac{dy(t)}{dt} + y(t) = K_p x(t)$$

Numerical integration

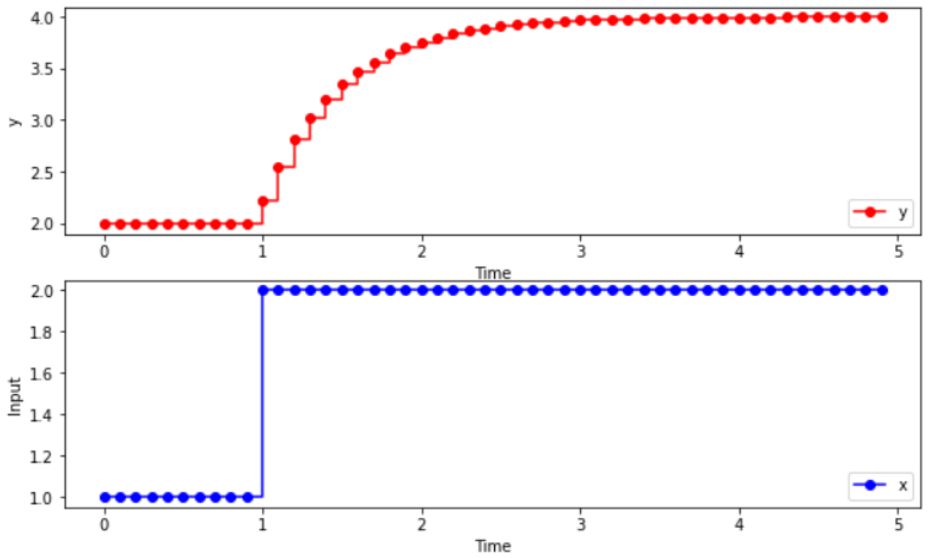
For each time step, evaluates $\frac{dy}{dt}$

Next y value calculated based on $\frac{dy}{dt}$:

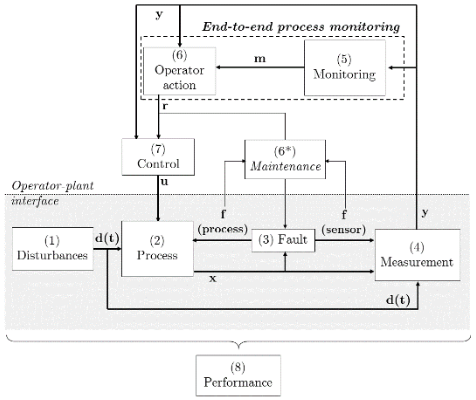
$$y(k+1) = y(k) + \Delta y$$

Next time step determined $t + \Delta t$

This may be iterative: varying Δt to ensure numerical accuracy

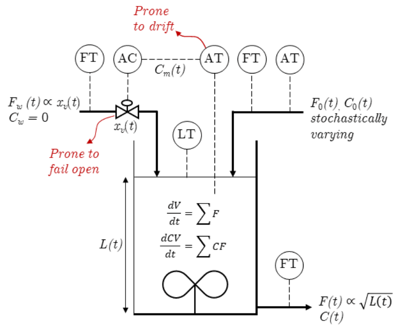


Simulation time



Motivation and details

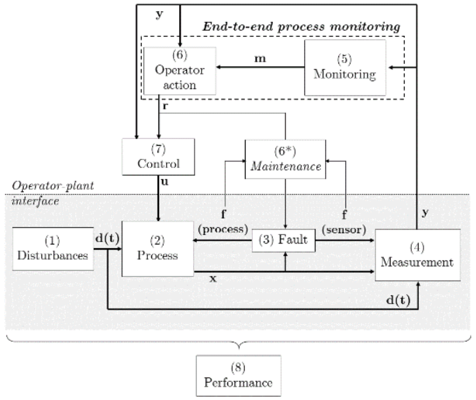
- Specification of simulation time
  - Time step Δt, ‘t[‘deltat’]’, e.g., 1 second
  - Simulation time t_{max}, ‘t[‘tmax’]’, e.g., 16 days in seconds
  - Maximum number of time steps N, ‘t[‘N’]’
  - Time vector $t = [N \times 1]$, ‘t[‘tvector’]
  - Current time index t_i, ‘t[‘i’]’

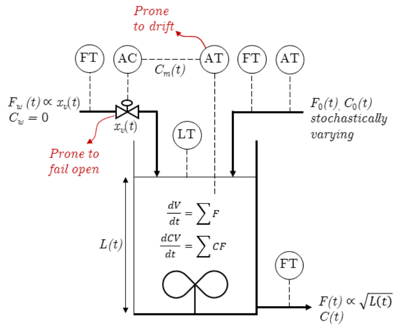


Disturbance module

Motivation and details

- Motivation:
  - Sufficient process variability is required to reflect the challenges of practical process monitoring
  - Real-world operations have varying disturbances, affecting the process and activating control systems
- Details:
  - A *disturbance model* generates suitable process disturbances $\boldsymbol{d}(t)$ over the required timescales
  - Disturbances are independent of the rest of the simulation
  - Various forms possible, e.g., deterministic, stochastic, seasonal

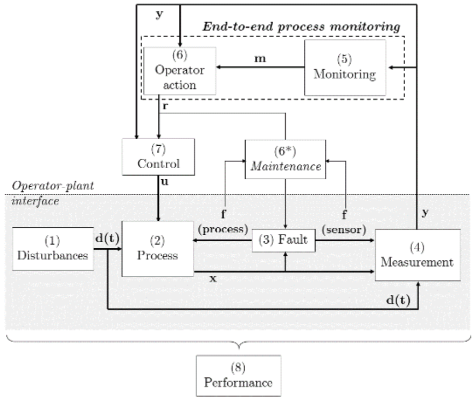


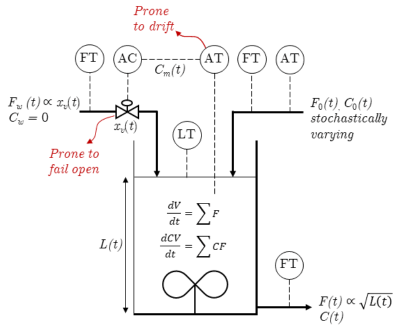


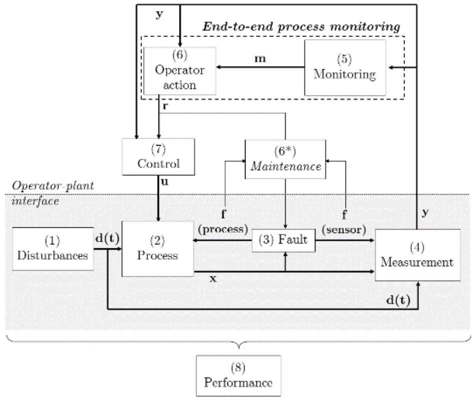
Disturbance module

Implementation

- Example of an autoregressive stochastic disturbances:
- $d_k = \phi d_{k-1} + (1-\phi)\mu + \sigma\sqrt{1-\phi^2}\,N(0,1)$
- Where:
  - d_i is the disturbance value at time step i,
  - ϕ and μ are autoregressive parameters
  - σ is a noise variance parameter
  - $N(0,1)$ is a standard normal variate



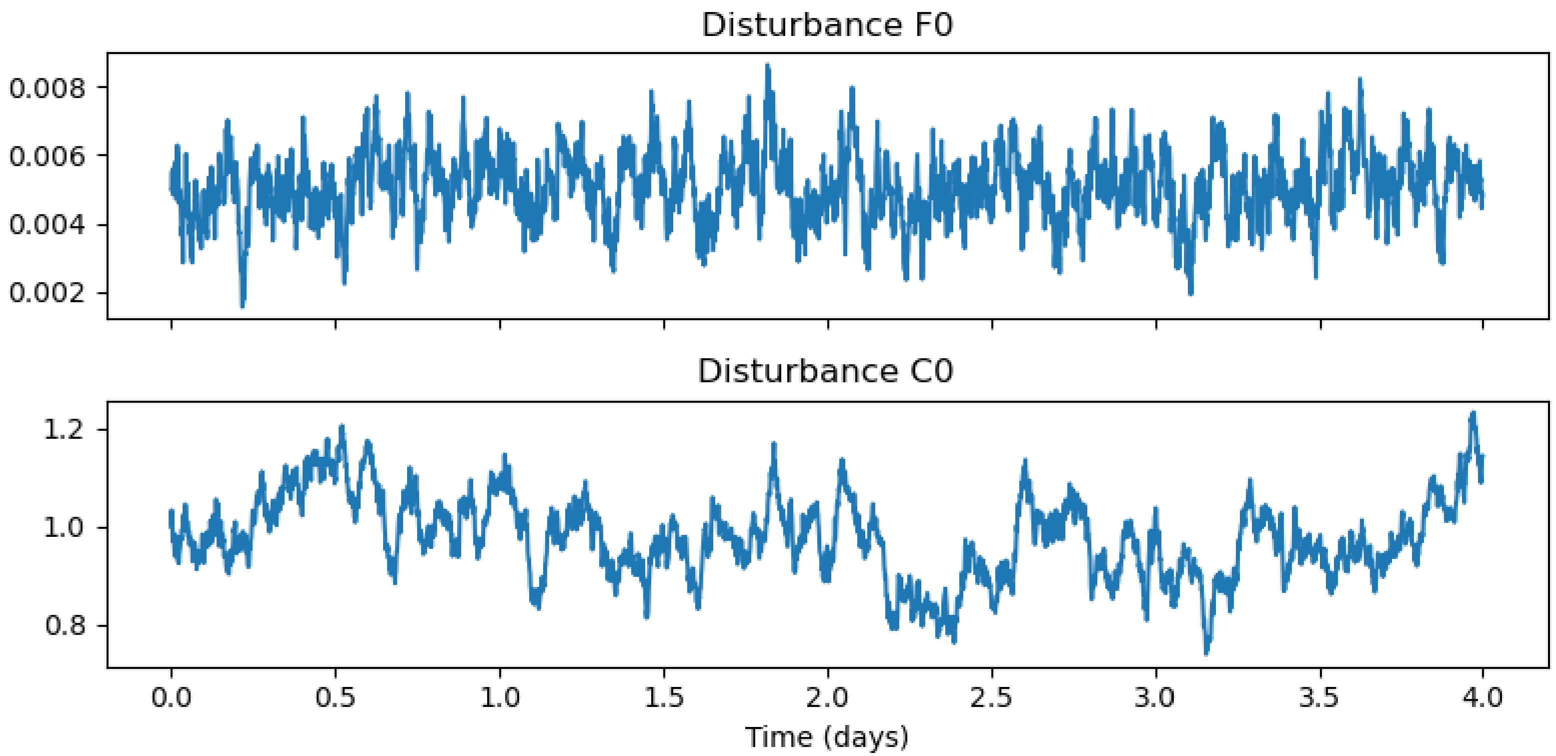


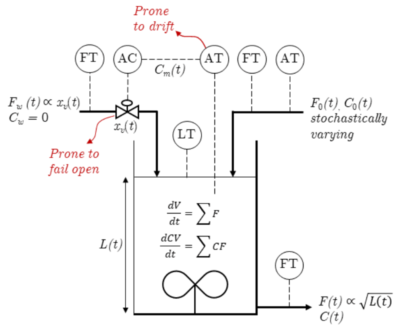


Disturbance module

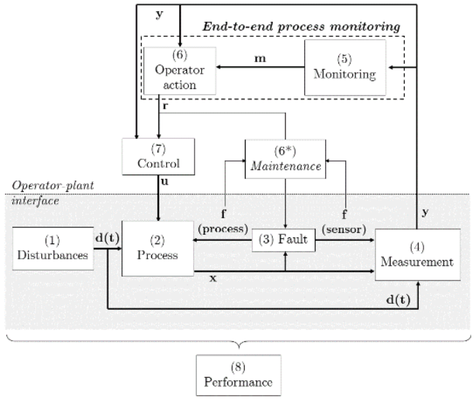
Implementation

- Example of an autoregressive stochastic disturbances:



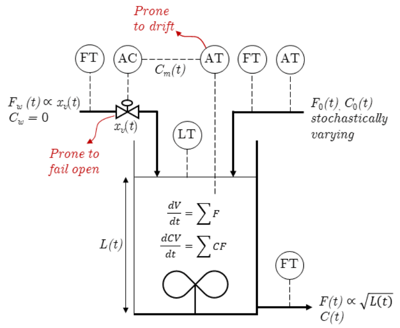


Process module

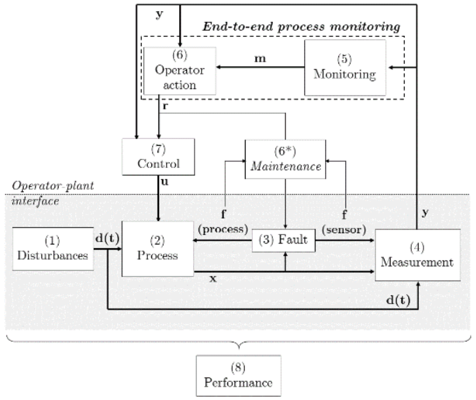


Motivation and details

- Motivation:
  - Real-world processes can have complex dynamics, potentially dependent on fault conditions
- Details:
  - A *process model* captures process dynamics
  - Inputs include disturbances $\boldsymbol{d}(t)$, control signals $\boldsymbol{u}(t)$, and fault conditions $\boldsymbol{f}(t)$
  - Outputs are intermediate variables and process states $\boldsymbol{x}(t)$
  - Considerations includes:
    - Dynamics and numerical integration
    - Initial conditions
    - Edge case handling

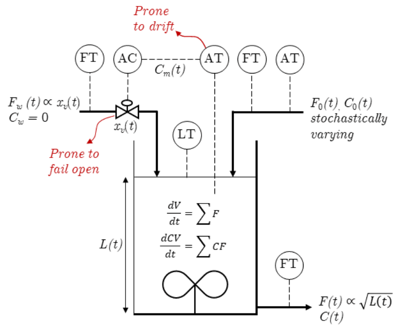


Process module

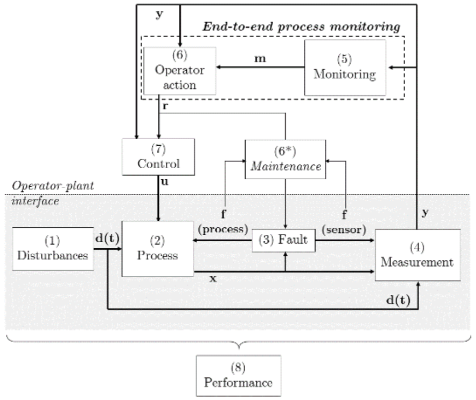


Implementation

- Process model is often a system of differential-algebraic equations
- Example of a dynamic process:
- Overall mass balance (assumption of constant density)
  - $\frac{dV}{dt} = F_0 + F_W - F$
- Component balance (assumption of similar density of solute and solute, no excess volume upon mixing)
  - $\frac{dCV}{dt} = C_0F_0 - CF$
- Valve dynamics
  - $\frac{dx_{F,v}}{dt} = \frac{1}{\tau_v}(x_{F,v,u} - x_{F,v})$
- Where:
  - V is the liquid volume in the tank
  - F_0, F_W, F are the feed inflow, water inflow, and outflow rates, respectively
  - C_0, C are the feed and product solute concentrations, respectively
  - $x_{F,v,u}, x_{F,v}$ are the control instruction and actual value of outflow valve opening, respectively
  - τ_v is the time constant for the valve dynamics



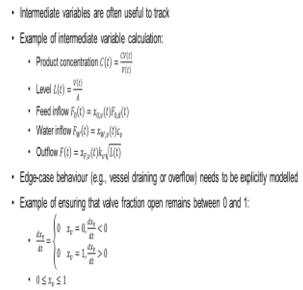
Process module



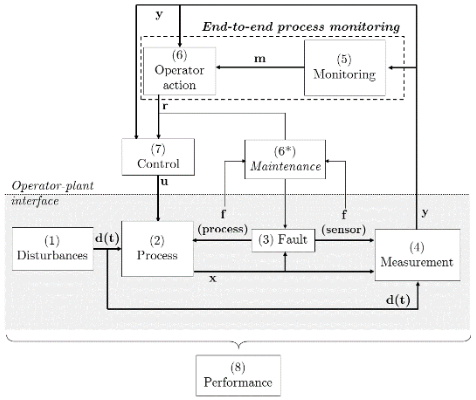
Implementation

- Intermediate variables are often useful to track
- Example of intermediate variable calculation:
  - Product concentration $C(t) = \frac{CV(t)}{V(t)}$
  - Level $L(t) = \frac{V(t)}{A}$
  - Feed inflow $F_0(t) = x_{0,v}(t)F_{0,d}(t)$
  - Water inflow $F_W(t) = x_{W,v}(t)c_v$
  - Outflow $F(t) = x_{F,v}(t)k_v\sqrt{L(t)}$
- Edge-case behaviour (e.g., vessel draining or overflow) needs to be explicitly modelled
- Example of ensuring that valve fraction open remains between 0 and 1:
  - $\frac{dx_v}{dt} = \begin{cases} 0 & x_v = 0, \frac{dx_v}{dt} < 0 \\ 0 & x_v = 1, \frac{dx_v}{dt} > 0 \end{cases}$
  - $0 \leq x_v \leq 1$

Note: Feed inflow and water inflow valves can be manipulated for start-up / shut down purposes

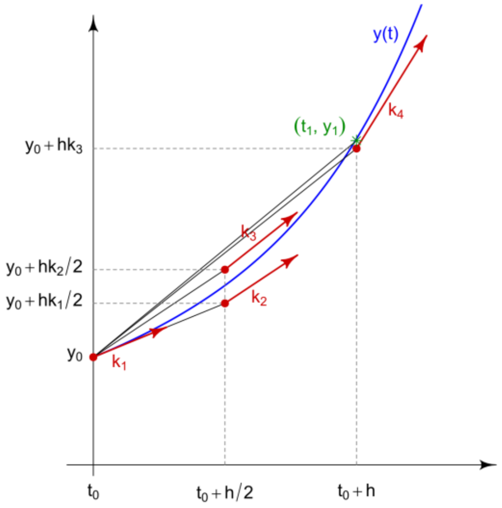

Intermediate variables are often useful to track
Example of intermediate variable calculation:
Product concentration
Level
Feed inflow
Water inflow
Outflow
Edge-case behaviour (e.g., vessel draining or overflow) needs to be explicitly modelled
Example of ensuring that valve fraction open remains between 0 and 1:

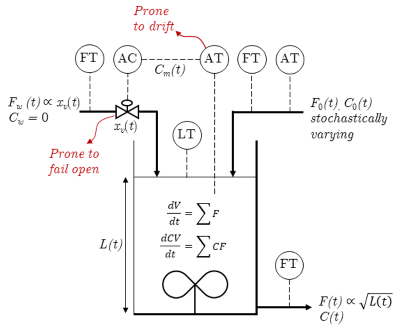
Process module



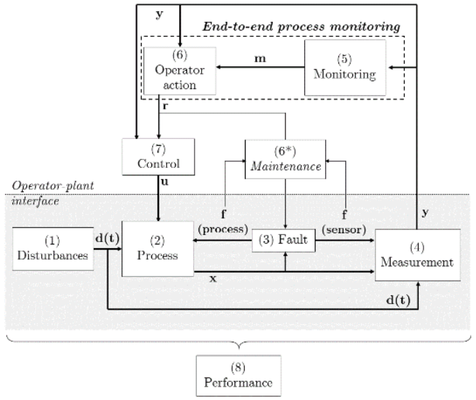
Implementation

- Process model is often a system of differential-algebraic equations
- State equations can be integrated sequentially to solve for the next time step
- E.g., fixed step Euler integration:
  - $x_{k+1} = x_k + \Delta x_k \Delta t$
  - $\Delta x_k = \left(\frac{dx}{dt}\right)_k$
  - Requires smaller step for stability
- E.g., Runge-Kutta fourth-order / fifth-order integration:
  - $x_{k+1} = x_k + h\sum_{i=1}^{4} b_i k_i$
  - $k_1 = f(t_k, x_k)$
  - $k_2 = f\left(t_k + \frac{h}{2}, x_k + h\frac{k_1}{2}\right)$
  - $k_3 = f\left(t_k + \frac{h}{2}, x_k + h\frac{k_2}{2}\right)$
  - $k_4 = f(t_k + h, x_k + hk_3)$
  - $b_i = \frac{1}{6}[1,2,2,1]$
  - Requires more calculations per step





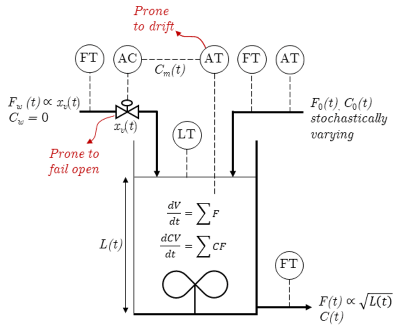
Process module

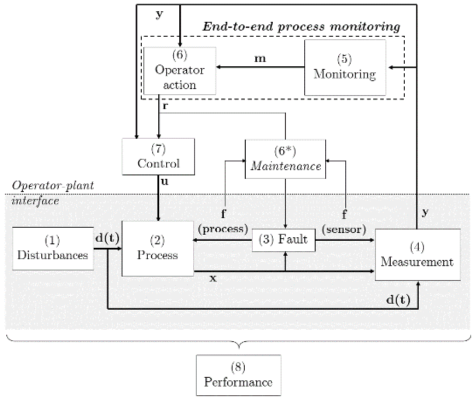


Implementation

- Solving for initial steady-state conditions is sometimes required for initialization of the process model, which may be challenging
- Example of initial steady-state calculation:
- Can define inputs $\boldsymbol{u}(t); \boldsymbol{d}(t)$, set points $\boldsymbol{x}(t); \boldsymbol{y}(t)$
- Accumulation = zero:
  - Valve dynamics
  - $0 = \frac{1}{\tau_v}\left(x_{F,v,u,ss} - x_{F,v,ss}\right); \therefore x_{F,v,ss} = x_{F,v,u,ss}$
  - Volume balance
  - $0 = F_{0,ss} + F_{W,ss} - F_{ss} = x_{0,v,ss}F_{0,d,ss} + x_{W,v,ss}c_v - x_{F,v,u,ss}k_v\sqrt{\frac{V_{ss}}{A}}$
  - Component mass balance
  - $0 = C_{0,ss}F_{0,ss} - C_{ss}F_{ss} = C_{0,d,ss}x_{0,v,ss}F_{0,d,ss} - C_{SP}x_{F,v,u,ss}k_v\sqrt{\frac{V_{ss}}{A}}$

Example: For desired C_{SP}, can calculate initial V_{SS} and $x_{W,v,ss}$ for fixed $x_{0,v,ss}$ and $x_{F,v,u,ss}$

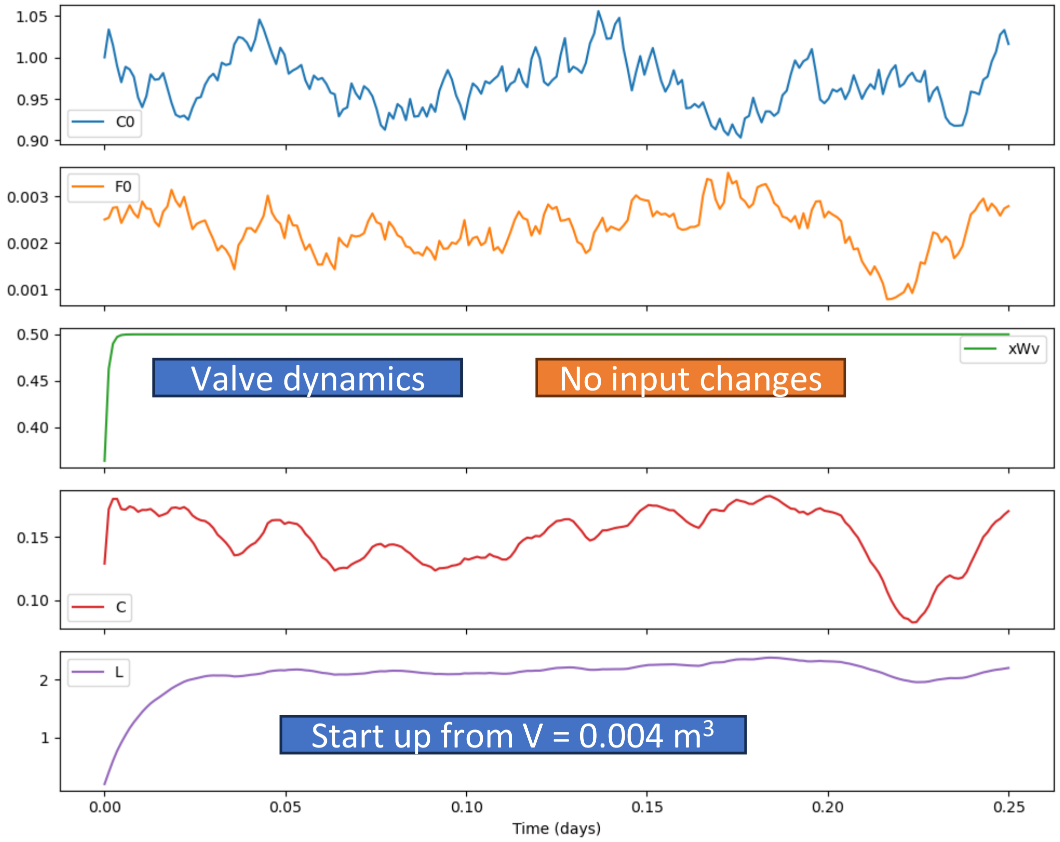




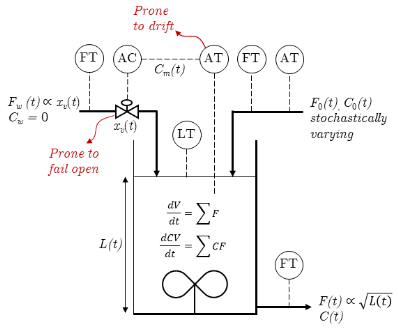
Process module

Implementation

- Alternative to initial conditions:
- Start from start-up conditions, and let regulatory control guide process to set points



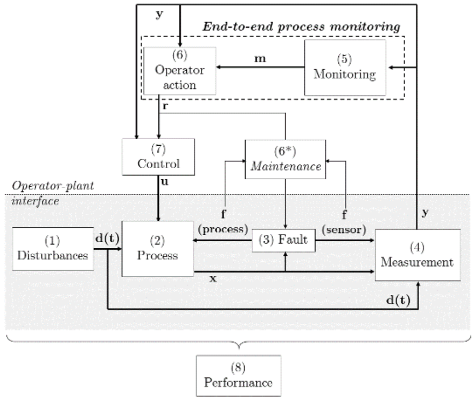
Disturbances

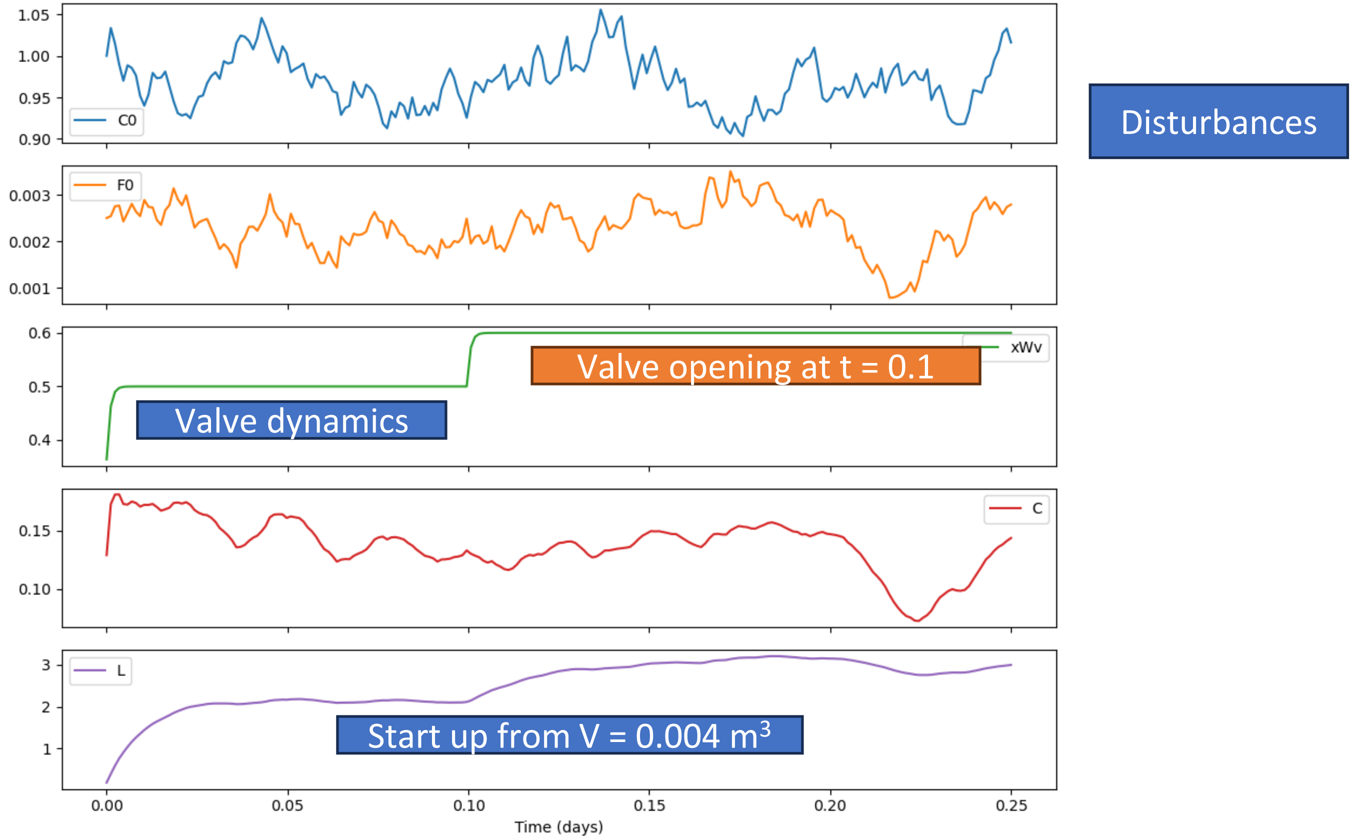


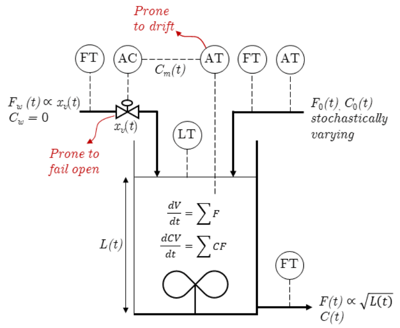
Process module

Implementation

- Alternative to initial conditions:
- Start from start-up conditions, and let regulatory control guide process to set points



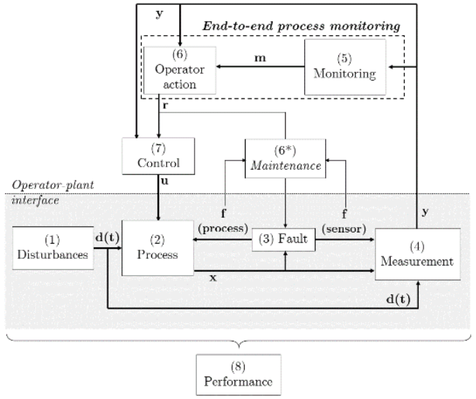


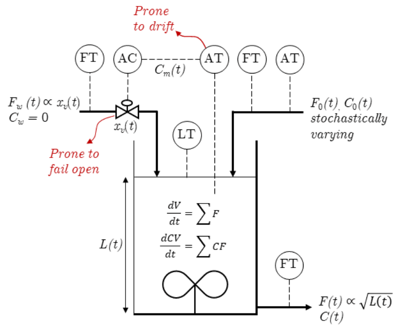


Fault module

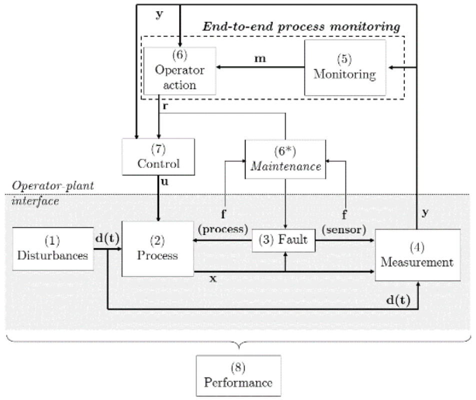
Motivation and details

- Motivation:
  - Real-world processes can have complex dynamics, potentially dependent on fault conditions
- Details:
  - A *fault model* simulates process, actuator and sensor faults
  - A fault signal $\boldsymbol{f}(t)$ captures the presence and nature of faults
  - Faults may depend on the current process state $\boldsymbol{x}(t)$
  - The fault signal serves as input to:
    - The *process model* – potentially affecting dynamics
    - The *measurement model* – potentially affecting sensor accuracy



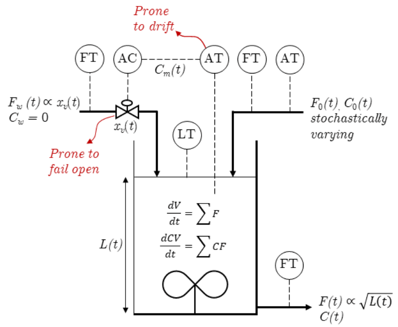


Fault module



Implementation

- The fault model typically contains a stochastic fault condition activation function, and the specification of how fault conditions affect the process and measurement modules
- Example of a fault model:
  - A *failure probability distribution* or *hazard rate* that incorporates the current run time of the component that can fail
  - A *fault state* update that evaluates the hazard function at each time step to check if a fault condition is activated
  - *Fault condition parameters* that are updated if the fault condition is activated

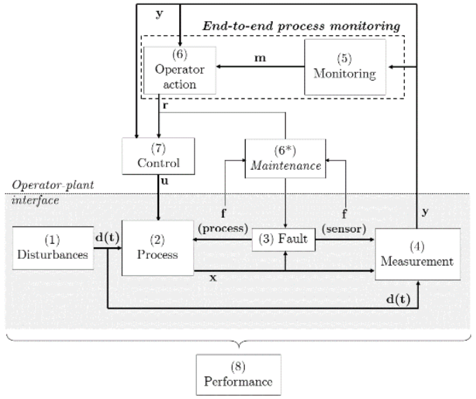


By Bathtub_curve.jpg: Wyattsderivative work: McSush (talk) - Bathtub_curve.jpg, Public Domain, https://commons.wikimedia.org/w/index.php?curid=7458336
Statistical models for bathtub curves: https://doi.org/10.1016/S0169-7161(01)20005-4

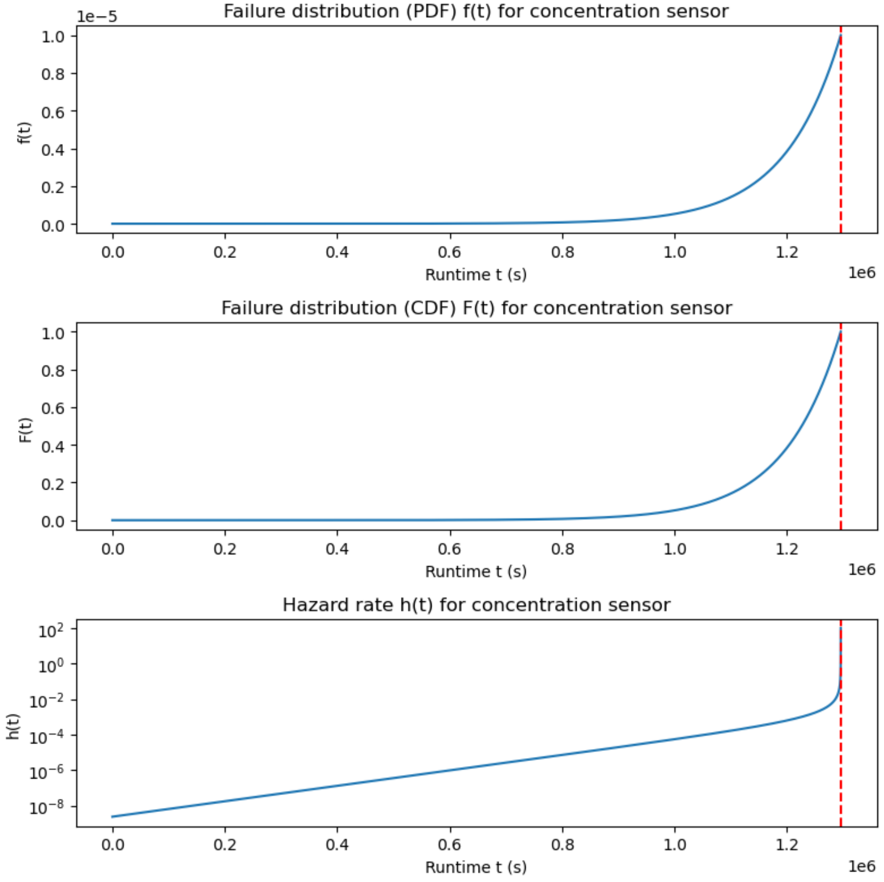
Fault module

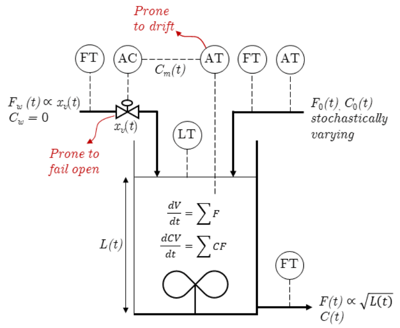
Implementation

- Example of a fault model:
  - Sensor *failure probability distribution* as a function of time

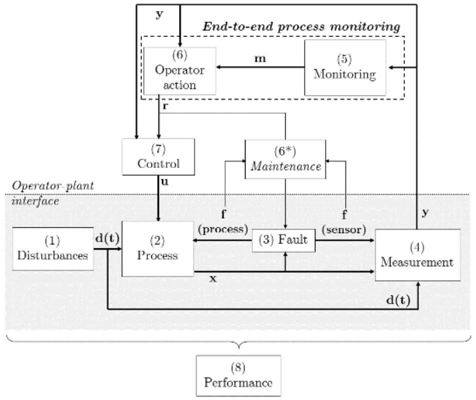


Runtime up to maximum lifetime



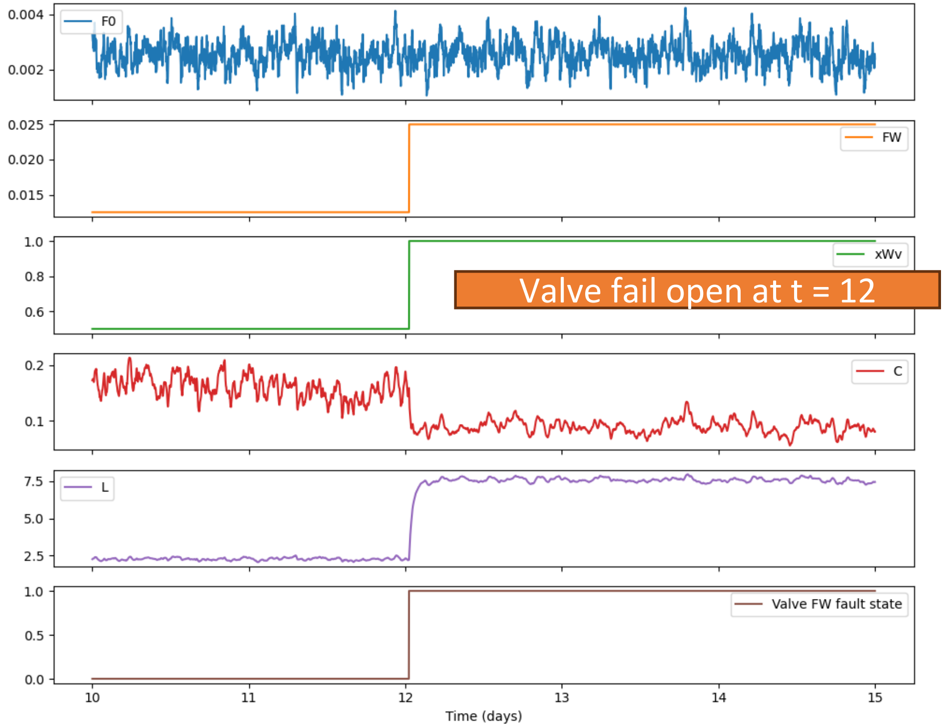


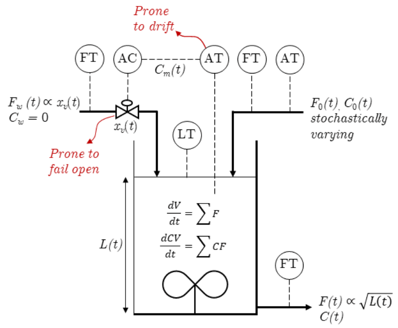
Fault module



Implementation

- Example of a fault model:
  - Different *fault states* and their potential effects as well as *fault condition parameters*:
    - Sensor stuck affects the **measurement model**, with measured value of sensor not updating with new measurements
    - Sensor bias affects the **measurement model**, with measured value offset by a constant bias from the real value
    - Sensor drift affects the **measurement model**, with measured value offset by a growing bias from the real value
    - Valve stuck affects the **process model**, with the actual valve opening (typically calculated through process dynamics) unaffected by control inputs or other instructions

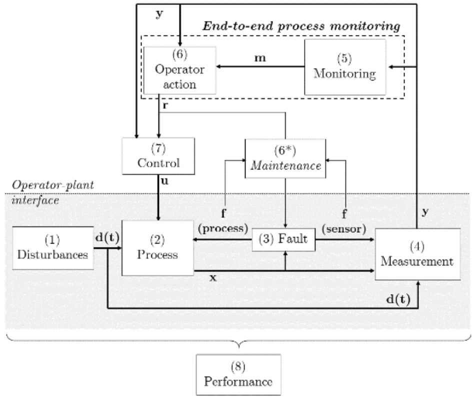


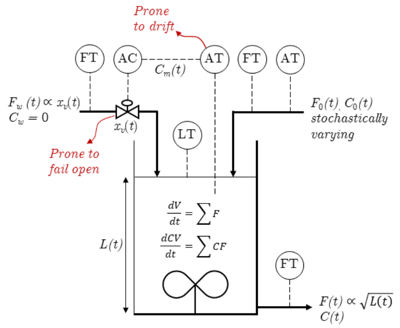


Measurement module

Motivation and details

- Motivation:
  - Measurements involve sensors, which may introduce time delays, variable sampling rates, measurement noise, and gross errors such as drifts
- Details:
  - A *measurement model* simulates measurement by sensors
  - Inputs to the measurement model are true process states $\boldsymbol{x}(t)$ and fault states $\boldsymbol{f}(t)$
  - Measured values $\boldsymbol{y}(t)$ are the outputs of the measurement model
  - The measured values serve as input to:
    - The *process model* – potentially affecting dynamics
    - The *measurement model* – potentially affecting sensor accuracy

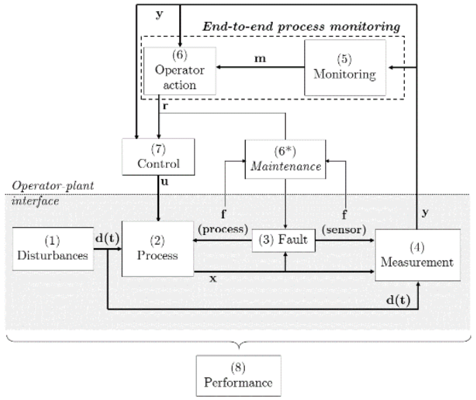


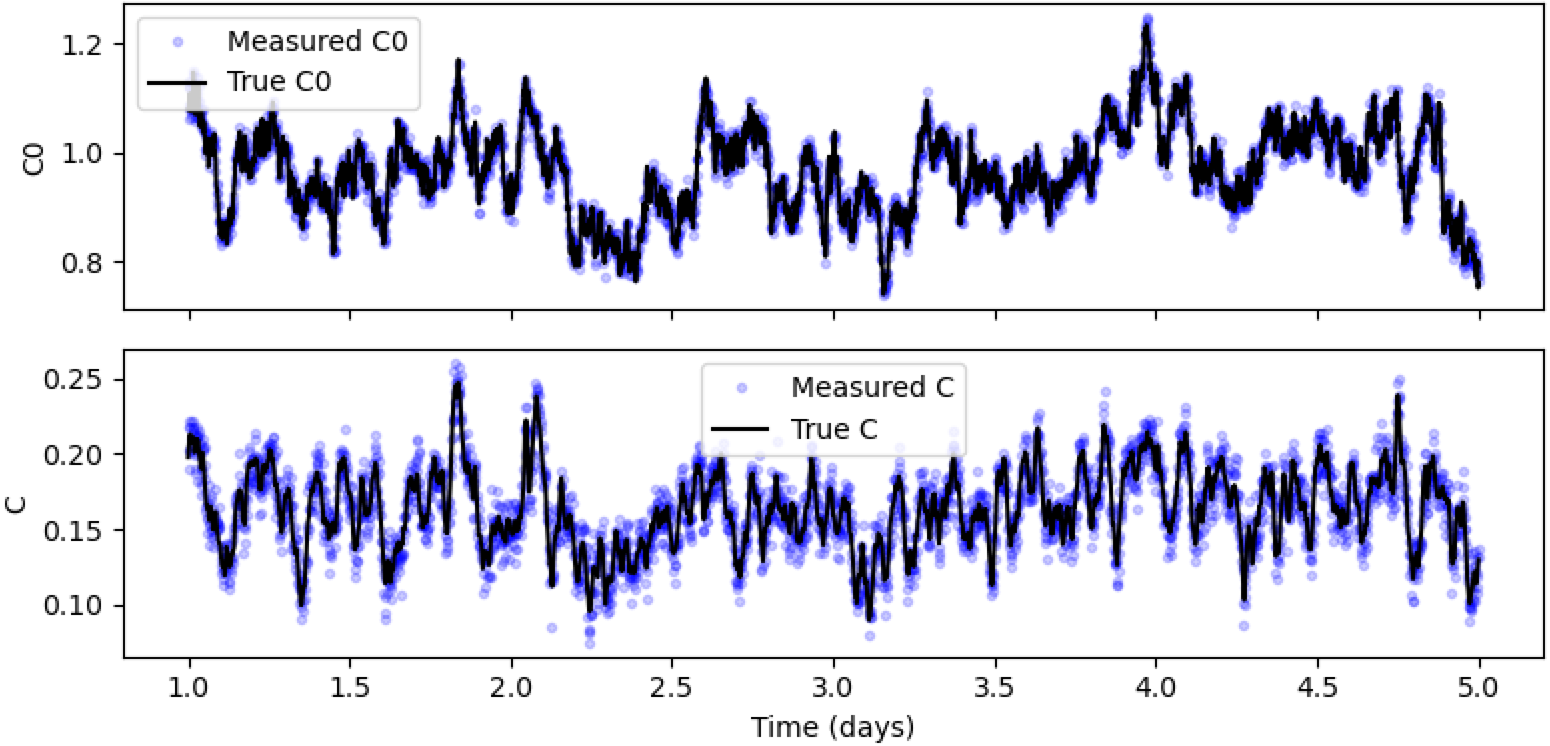


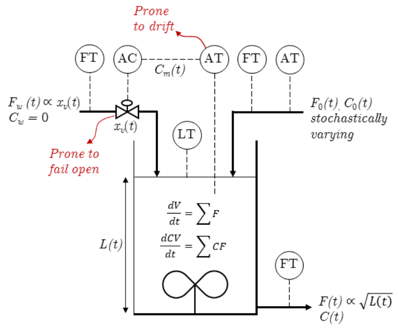
Measurement module

Implementation

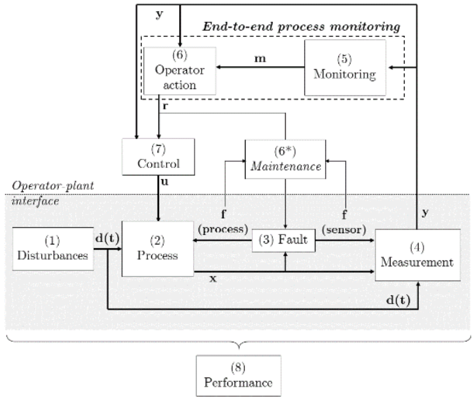
- Example of noise addition
  - $y_{k+1} = x_k + N(0, \sigma_{noise})$





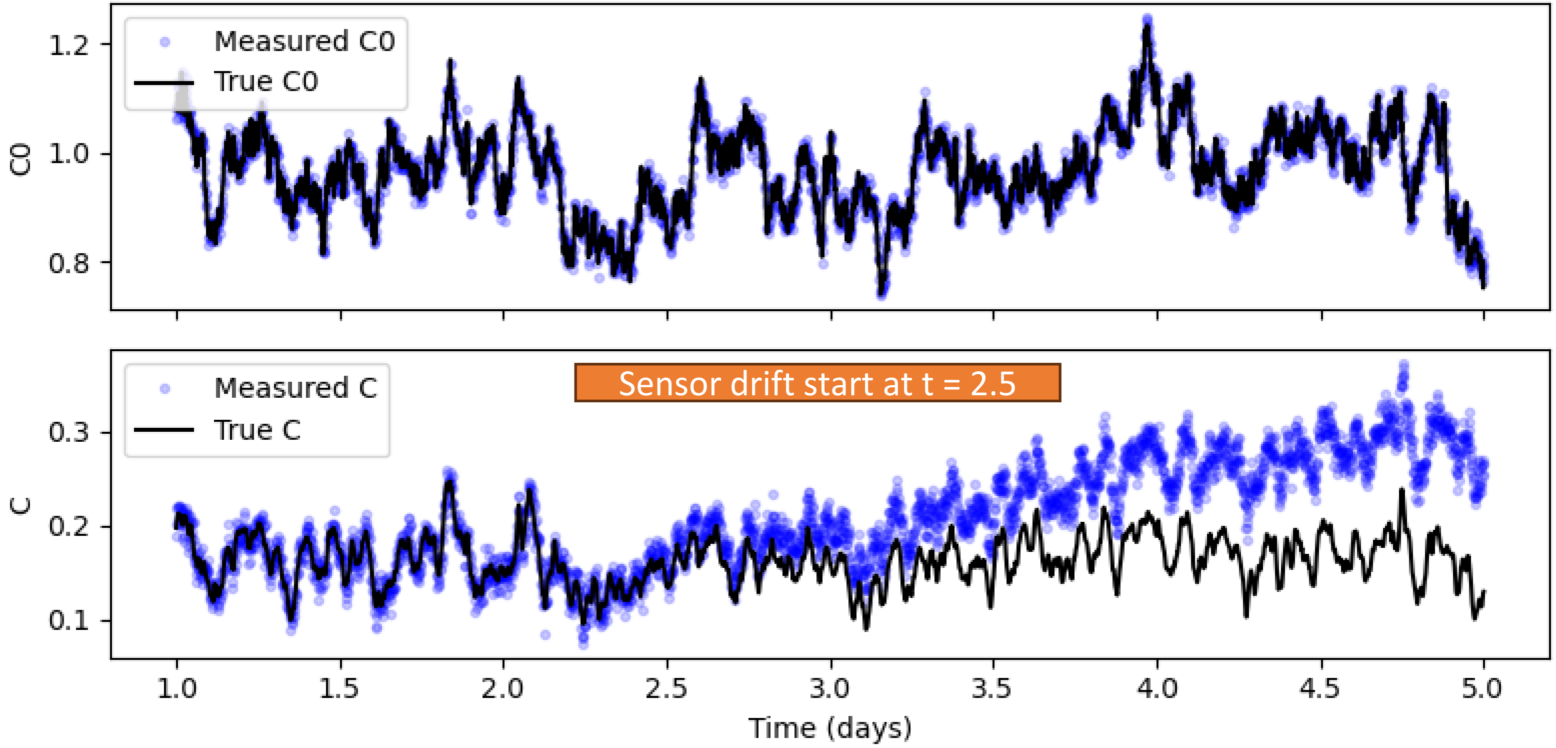


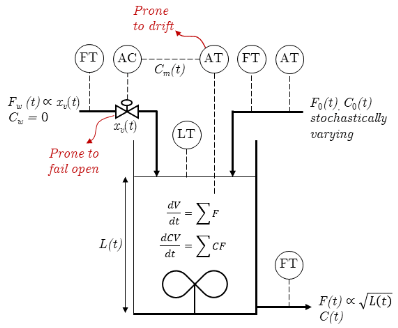
Measurement module



Implementation

- Effect of different sensor faults on measurement:
  - Stuck sensor: $y_{k+1} = y_k$
  - Sensor bias: $y_{k+1} = x_k + N(0, \sigma_{noise}) + \beta_{bias}$
  - Sensor drift: $y_{k+1} = x_k + N(0, \sigma_{noise}) + \Delta_{drift,k}; \Delta_{drift,k+1} = \Delta_{drift,k+1} + \delta_{drift}\Delta t$





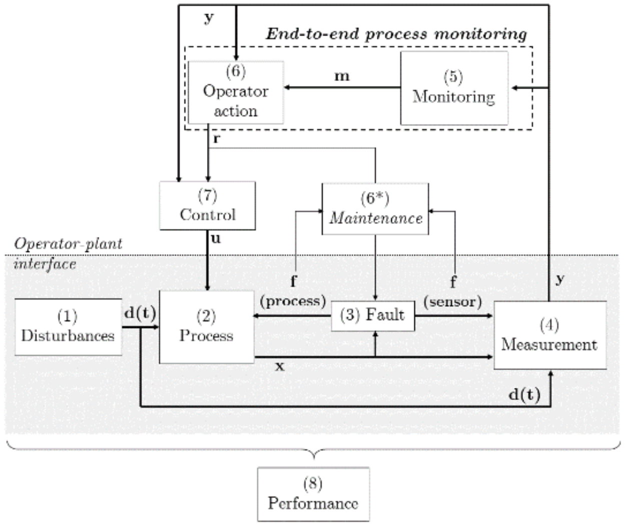
Framework familiarization and experiments

Interactive session

Setup and familiarization

Goals and exercises

- Run the illustrative examples in Python (optional) for the following models:
    - Disturbance model
    - Process model
    - Fault model
    - Measurement model
- **Exercise**: Change the disturbance model parameters and observe the effect
- **Exercise**: Double the outflow valve coefficient of the process model and observe the effect
- **Exercise**: Try different random seeds for the fault model illustration and observe the effect
- **Exercise:** Try different parameters for the measurement model (and measurement faults) and observe the effect



Framework module details

Regulatory control model

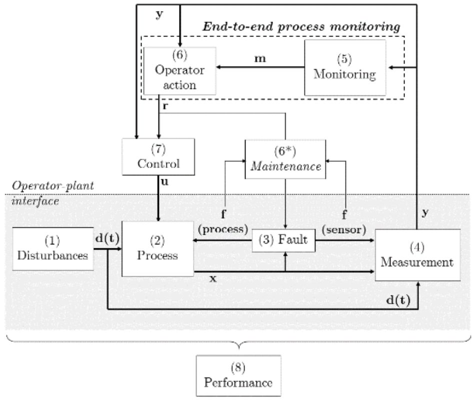
Monitoring model

Supervisory control model

Maintenance model

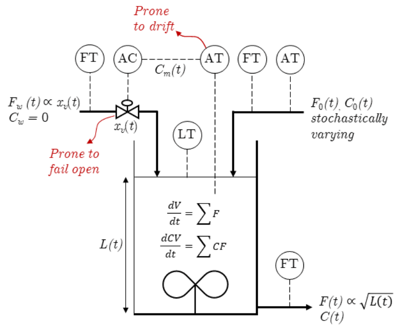
Economic performance model

Regulatory control module



Motivation and details

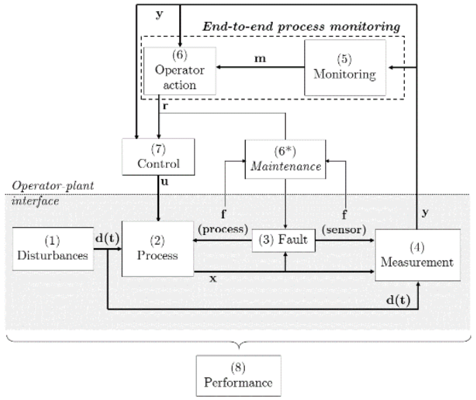
- Motivation:
  - Control is required to keep important process variables at their setpoints
  - Control equipment (actuators and sensors) are subject to faults, which deteriorate control and process performance
  - Through control output and process variable interaction (process responses and control responses), correlations between process variables may become unintuitive (from a process response perspective)
- Details:
  - A *regulatory control model* simulates control output calculated from setpoints, measurements, and control algorithms
  - Inputs to the control model are measurement values $\boldsymbol{y}(t)$, control mode (automatic or manual), setpoints (part of $\boldsymbol{r}(t)$), and current inputs $\boldsymbol{u}(t)$
  - The output of the control model is the control outputs $\boldsymbol{u}(t)$ (e.g., valve position instructions)
  - The control outputs serve as input to:
    - The *process model*

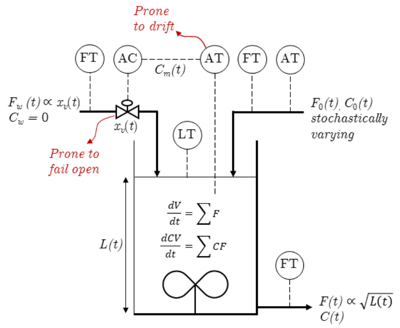


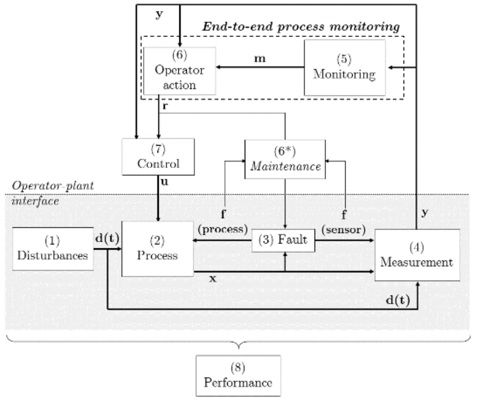
Regulatory control module

Implementation

- Automatic control, e.g., PI algorithm:
  - Automatic control mode indicated by supervisory control flag (e.g., ‘valve position’ = -1)
  - $u(t) = -K \times \left(e(t) + \frac{\Delta t}{\tau_I} \sum e(t) \right) + \bar{u}$
  - Tuning constants: K (absolute value; sign indicates direct or reverse acting); τ_I
  - Controller bias: $\bar{u}$
- Manual control:
  - Manual control indicated by supervisory control flag (e.g., ‘valve position’ ≠ -1)
  - Specific value for control output (e.g., valve position) indicated, e.g., 0.5 (halfway open)



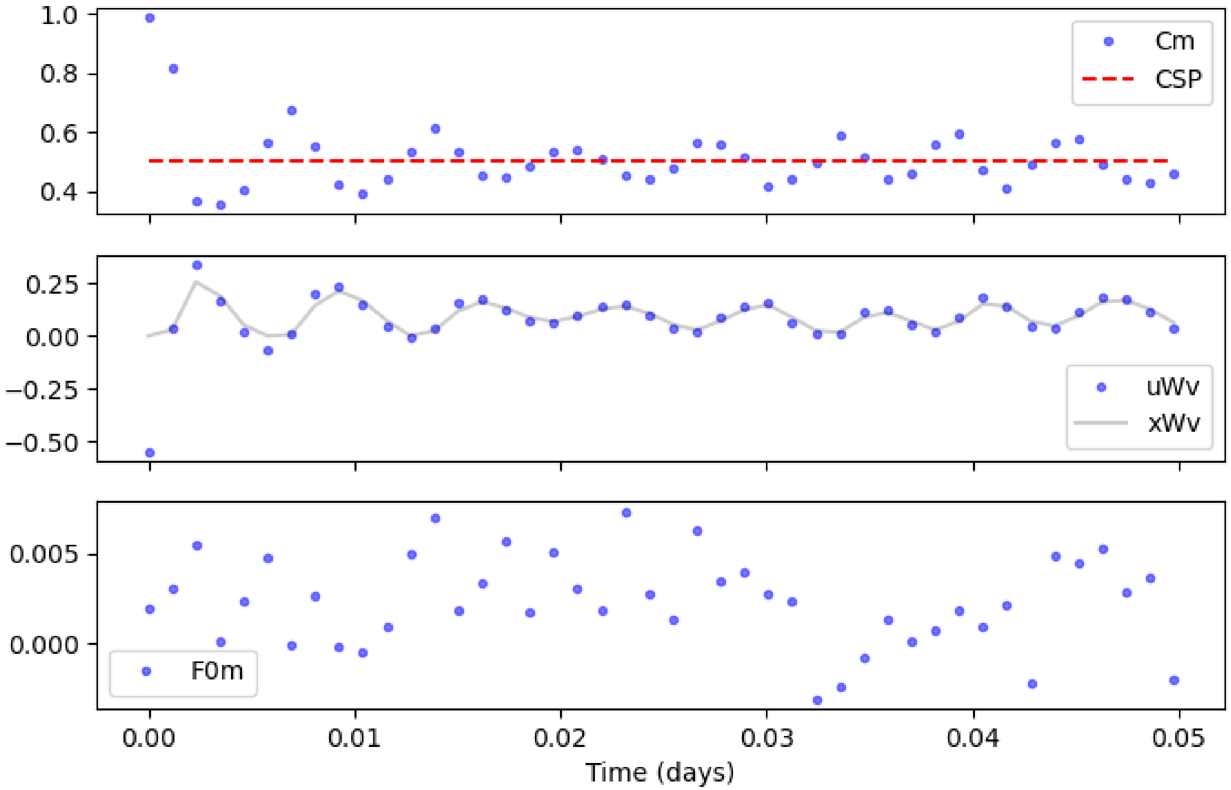


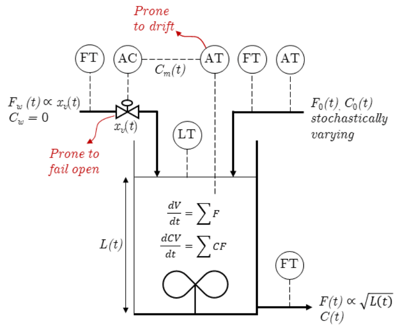


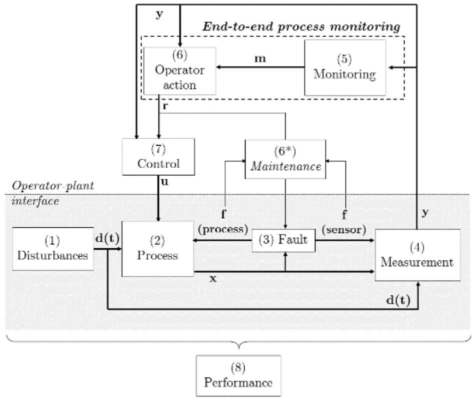
Regulatory control module

Implementation

- Example: Outflow concentration $C(t)$ control through PI control affecting feed water valve $u_{Wv}(t)$
  - Noise from measurement $C_m(t)$ propagates to control output $u_{Wv}(t)$
  - Valve dynamics (in *process model*) results in filtered actual valve position $x_{Wv}(t)$



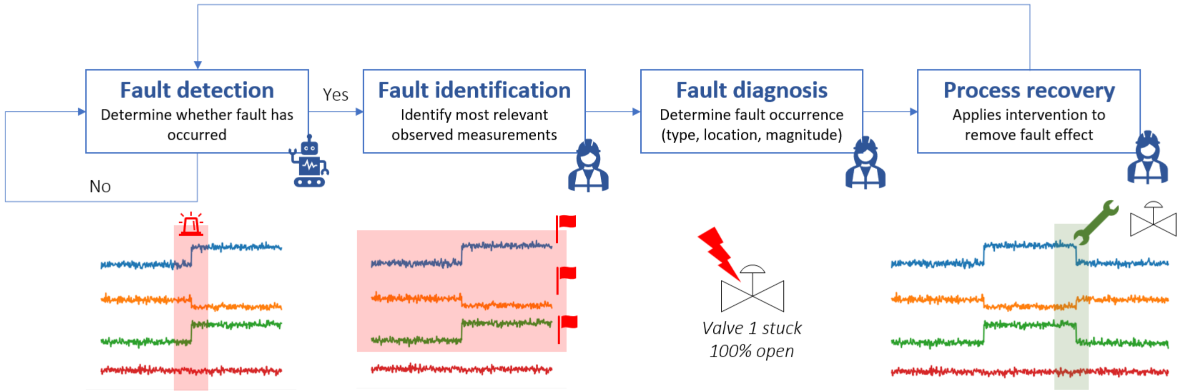


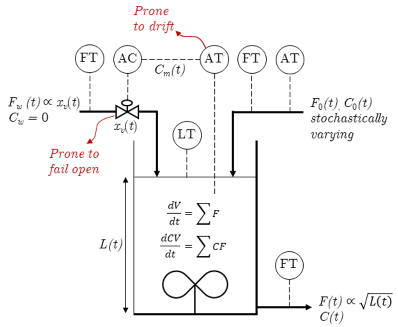


Monitoring module

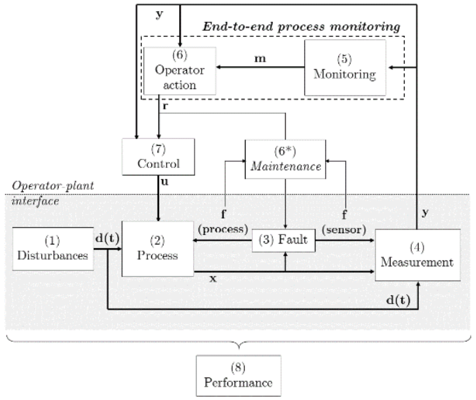
Motivation and details

- Motivation:
  - Monitoring is required to detect, identify, diagnose and rectify faults
- Details:
  - A *monitoring model* simulates monitoring activities (detection, identification, diagnosis)
  - Inputs to the monitoring model are measurement values $\boldsymbol{y}(t)$
  - The output of the monitoring model $\boldsymbol{m}(t)$ include warnings, alarms, and fault diagnosis information
  - The monitoring outputs serve as input to:
    - The *operator actions: supervisory control model*



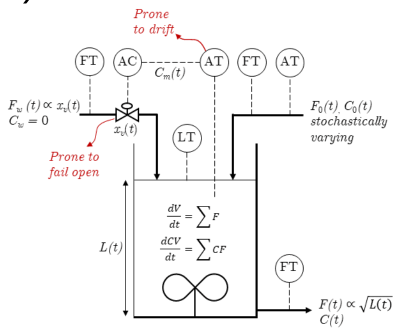


Monitoring module

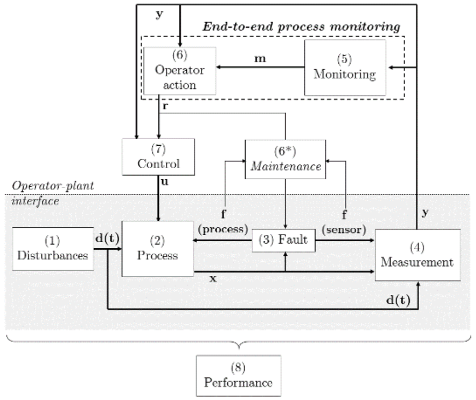


Implementation

- The monitoring model involves a **training** and an **application** phase
- During **training**:
  - Training measurement data is collected from $\boldsymbol{y}(t)$ for a time span $t_{trainingStart}$ to $t_{trainingEnd}$
    - $t_{trainingStart}$ should allow for the initial start up phase to be complete before normal operating conditions data is collected
    - $t_{trainingEnd}$ should allow sufficient representative NOC data (or faults, in the case of supervised approaches)
  - Monitoring model parameters are estimated (e.g., loadings in PCA monitoring)
  - This phase may include hyperparameter optimization (e.g., number of components to be retained in PCA monitoring)
- During **application**:
  - Online measurement data is collected from $\boldsymbol{y}(t)$ at each time step
  - The trained monitoring model is applied to generate monitoring statistics (e.g., scores and reconstruction errors calculated, modified Hotelling's T^2 and SPE statistics compared to thresholds)
  - Typically, expert rules are applied to translate monitoring statistics and contribution plots to fault diagnosis outputs

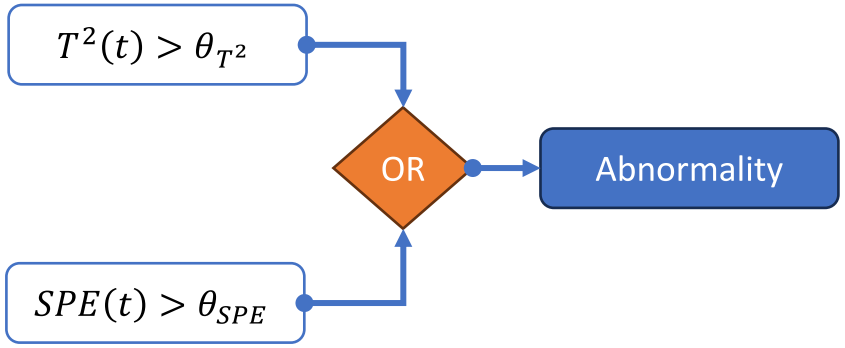


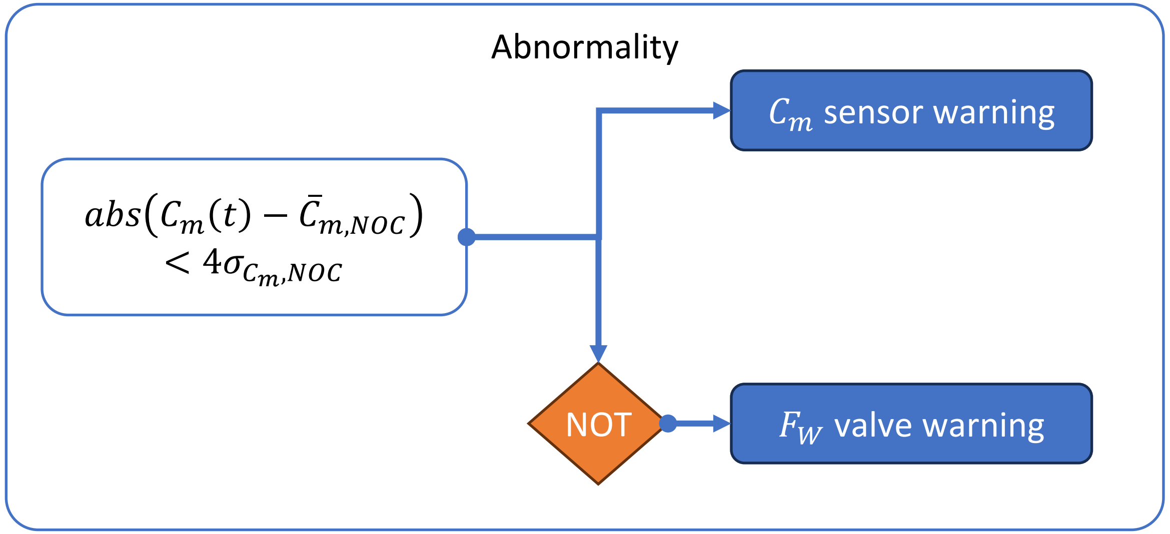
Monitoring module

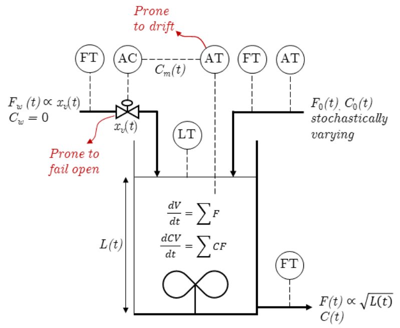


Implementation

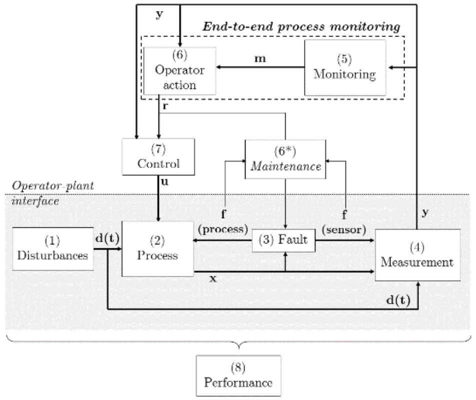
- Example of expert rules for fault diagnosis:
- A selection of components that can fail is pre-defined, e.g., all valve components and all sensor components
- For each component, an expert rule is applied to determine if a **warning** is logged at a specific time step
- E.g., warning for composition sensor component or warning for feed water valve component:







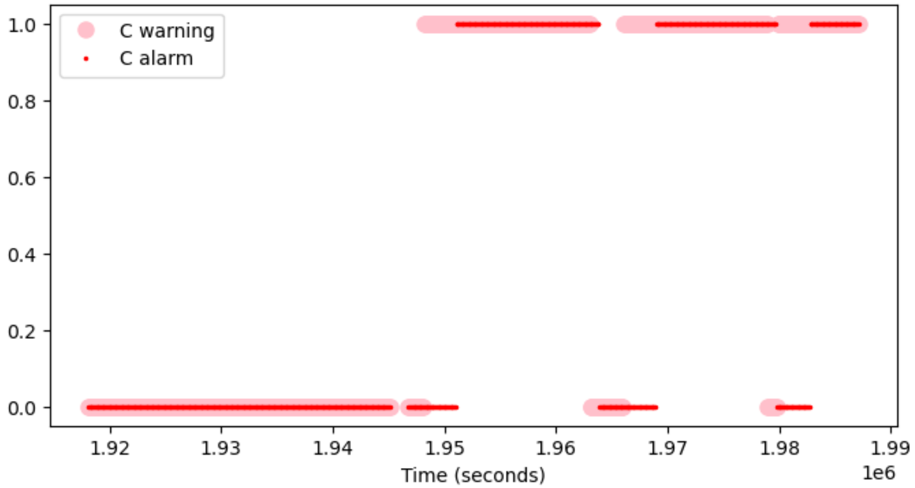
Monitoring module

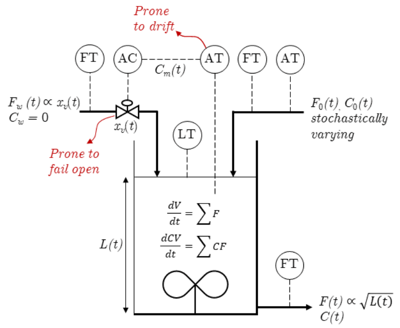


Implementation

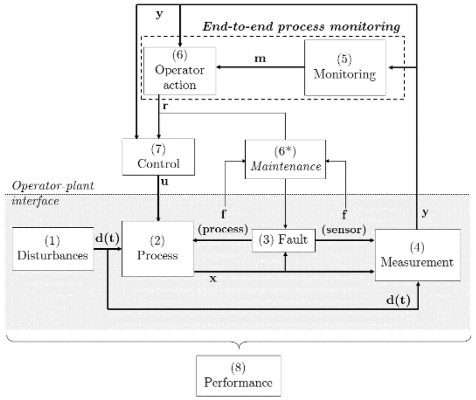
- Example of expert rules for fault diagnosis:
- Time-series of **warning** signal is aggregated to an **alarm** signal
  - E.g., warning signal of 1 (warning) or 0 (no warning)
  - Aggregation of warning to alarm, with a_f fraction of warnings for alarm, a_w window length

$$alarm(i) = \begin{cases} 1 \quad if \displaystyle\sum_{i=t-a_w}^{t-1} warning(i) > (a_f \times a_w) \\ 0 \qquad\qquad otherwise \end{cases}$$



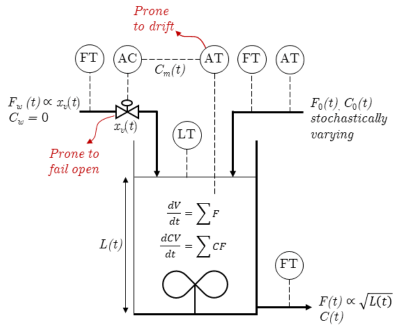


Operator actions: supervisory control module



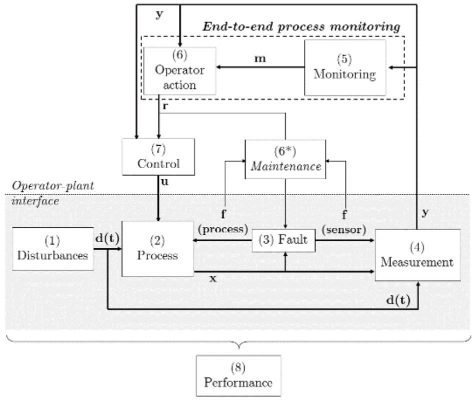
Motivation and details

- Motivation:
  - Faults may cause abrupt changes to processes, requiring shutdown and unplanned maintenance
  - Fault tolerant control actions may include changes to controller setpoints
- Details:
  - Supervisory control specifies the current operating regime (startup, running, shutdown, shut), triggered by:
    - Interlocks based on measurements $\mathbf{y}(t)$
    - Alarms and diagnosis from the outputs of the monitoring system $\boldsymbol{m}(t)$
  - Setpoints $\boldsymbol{r}(t)$ for controllers are provided
  - Tracking of components (valve and sensors) $\boldsymbol{r}(t)$ to serve as inputs during maintenance



Operator actions: supervisory control module

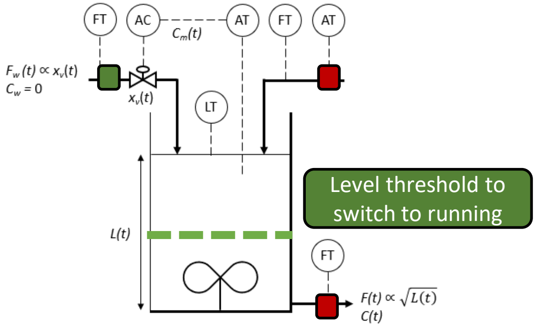
Motivation and details

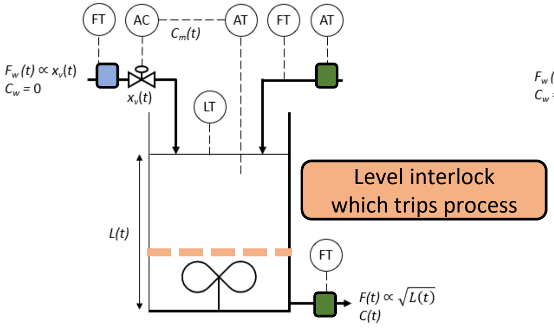


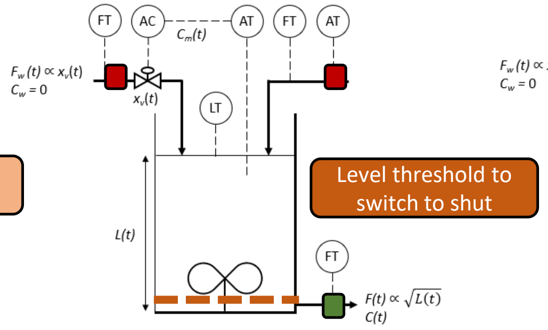
Shutdown triggers:
Level interlock
Planned maintenance (scheduled)
Unplanned maintenance (alarm triggered from **monitoring**)

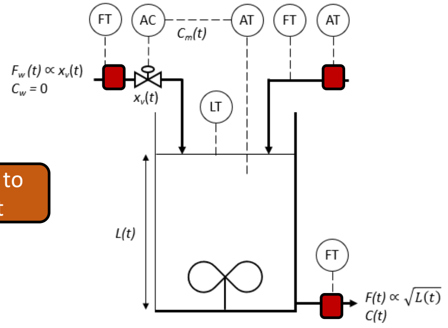
Initial startup	Running	Shutdown	Shut	Startup	Running
F_0, F valves closed F_W valve fully open When $L_{startup}$ threshold reached, switch to “Running” regime	F_0, F valves opened F_W controlled When $L_{interlock}$ threshold reached, switch to “Shutdown” regime	F_0, F_W valves closed F valve fully open When $L_{shutdown}$ threshold reached, switch to “Shut” regime	All valves closed Maintenance strategy applied, e.g.: valves/sensors/all checked, replaced	Similar to initial startup	See previous description

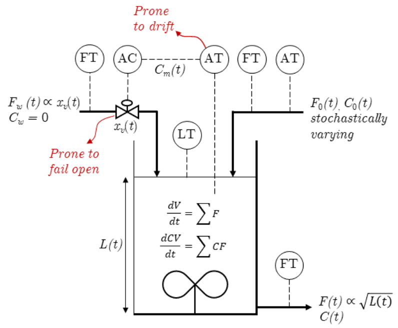
Time



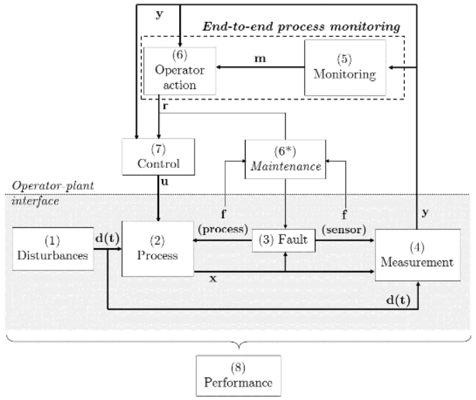






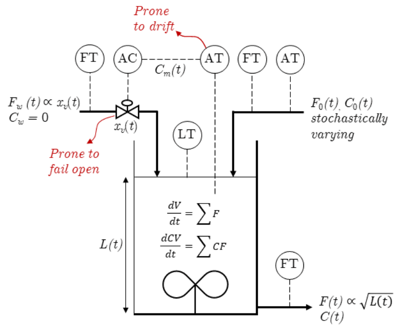


Operator actions: supervisory control module

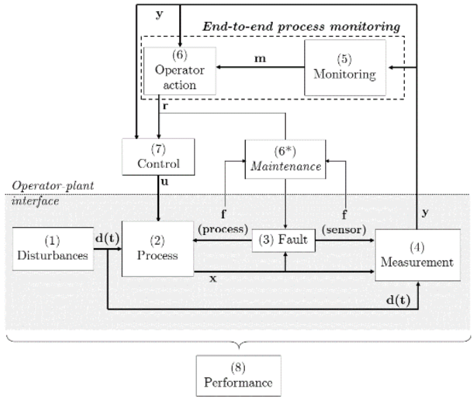


Implementation

- The *supervisory control model* considers each regime (running, shutdown, shut, startup) and provides the following for each of the regimes:
  - Valve positions / valve control type
  - Setpoints for regulatory controllers
  - Switching to next regime
- **Shut** regime example:
  - Valves for F_0, F_W closed, F open (all manual control)
  - No setpoint for composition controller
  - Switch to **Startup** regime
  - Update time for next planned shut in case current shut was not unplanned
- **Startup** regime example:
  - Valves for F_0, F closed, F_W open (all manual control)
  - No set point for composition controller
  - Switch to **Running** regime if startup level (high) threshold reached

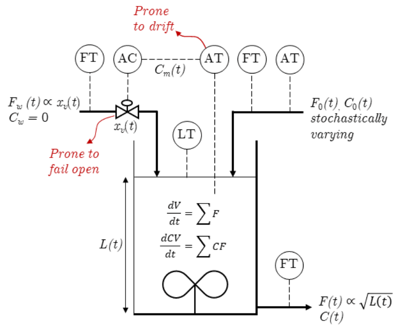


Operator actions: supervisory control module

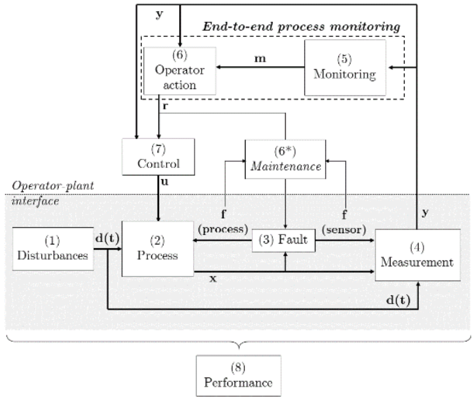


Implementation

- The *supervisory control model* considers each regime (running, shutdown, shut, startup) and provides the following for each of the regimes:
  - Valve positions / valve control type
  - Setpoints for regulatory controllers
  - Switching to next regime
- **Running** regime example:
  - Valves for F_0, F open (all manual control), valve for F_W under automatic control
  - Setpoint for composition controller as configured upfront
  - Switch to **Shutdown** regime if:
    - <u>Level high interlock activated</u>: unplanned shutdown,
all components considered faulty
    - <u>Planned maintenance due</u>:
components to be checked depends on maintenance cycle
    - <u>Alarm raised by monitoring model</u>, and maintenance strategy triggers unplanned maintenance,
flagged components considered faulty

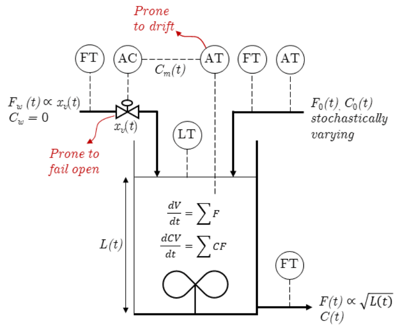


Operator actions: supervisory control module



Implementation

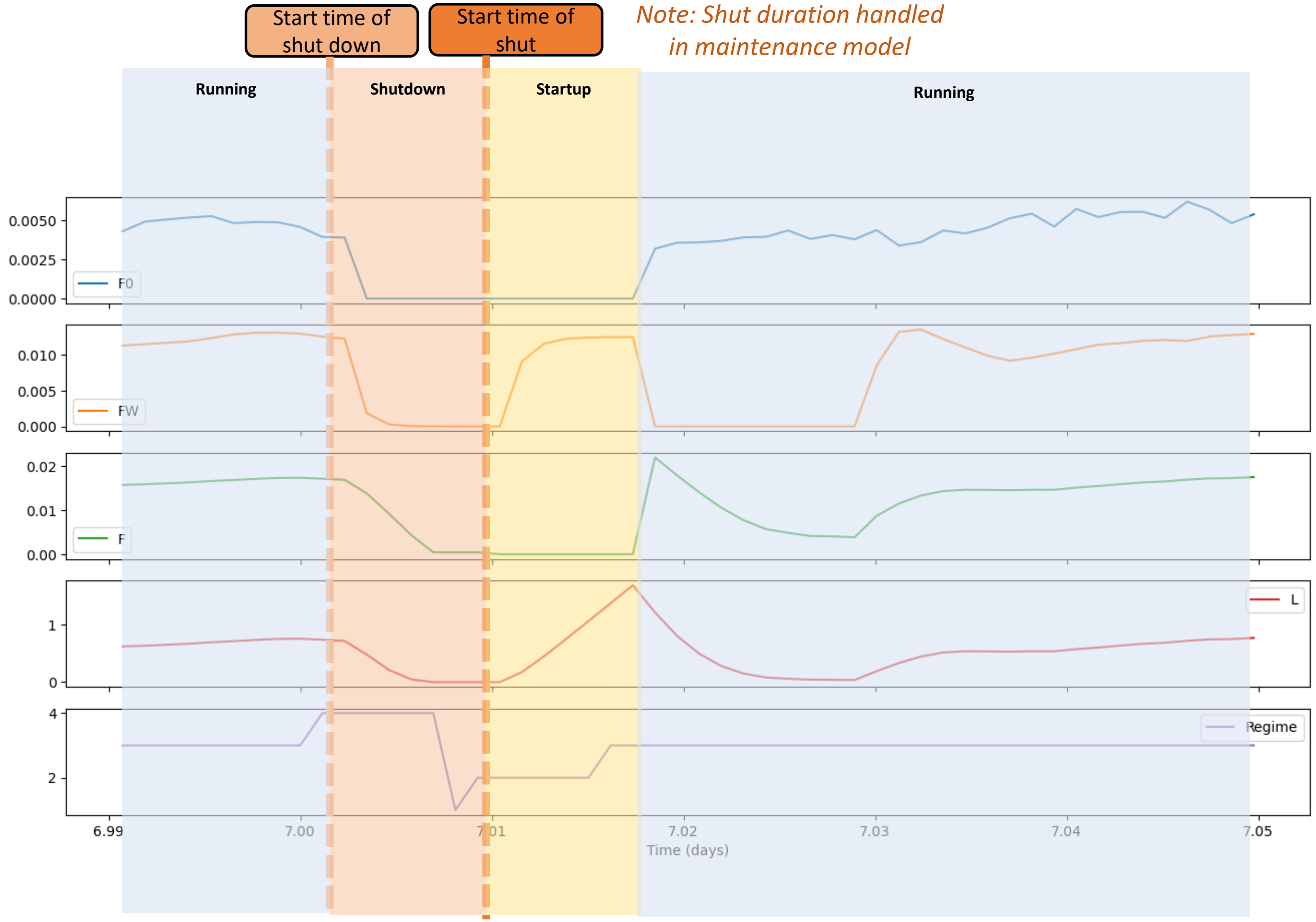
- The *supervisory control model* considers each regime (running, shutdown, shut, startup) and provides the following for each of the regimes:
  - Valve positions / valve control type
  - Setpoints for regulatory controllers
  - Switching to next regime
- **Shutdown** regime example:
  - Valves for F_0, F_W closed, F open (all manual control)
  - No setpoint for composition controller
  - Switch to **Shut** regime if regime if shutdown level (low) threshold reached

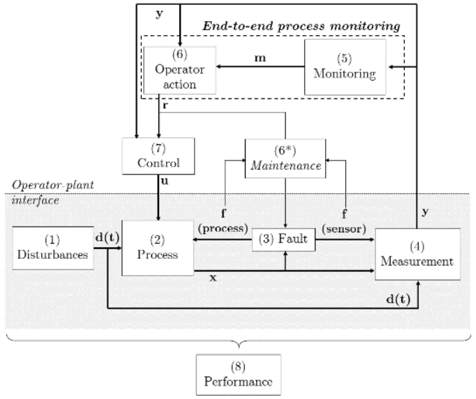


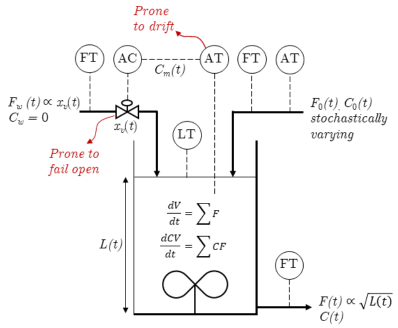
Operator actions: supervisory control module

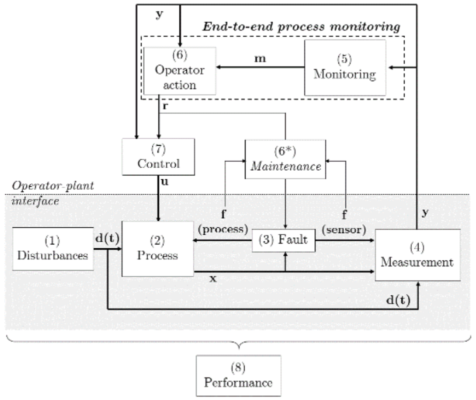
Implementation

- Example of scheduled shutdown





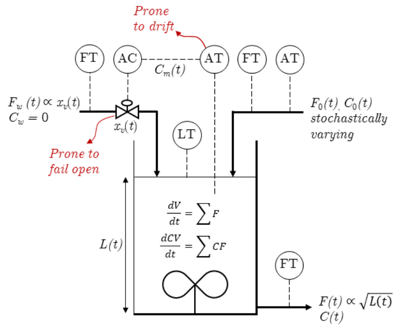




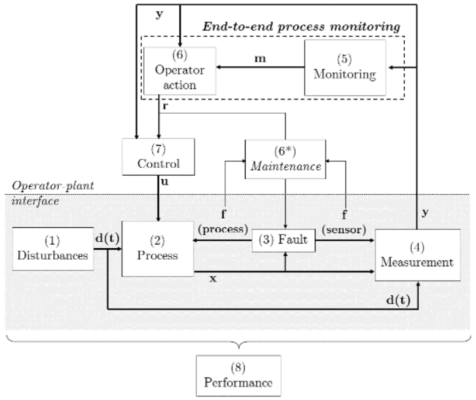
Maintenance module

Motivation and details

- Motivation:
  - Process recovery after fault diagnosis often involves servicing or replacing physical components (e.g., valves, sensors)
  - Maintenance reduces the time a process is running (reducing revenue), and incurs additional maintenance costs
  - Periods of planned maintenance is typically part of process operations
  - Faulty components need maintenance to prevent sub-optimal control and process performance
  - Critical component maintenance might be required outside planned maintenance, as part of unplanned maintenance
  - Finding a balance between planned and unplanned maintenance is a challenging problem
- Details:
  - The *maintenance model* simulates the checking and replacement of faulty components
  - The inputs to the maintenance model are:
    - Operating regime (indicating whether a shut is occurring),
    - Monitoring outputs (indicating which components – valves and/or sensors – have been flagged for unplanned maintenance)
    - Fault status of component (only available during maintenance)
  - The outputs of the maintenance model are:
    - Updated time (representing the passage of time of a maintenance shut)
    - Updated fault status of serviced / replaced components

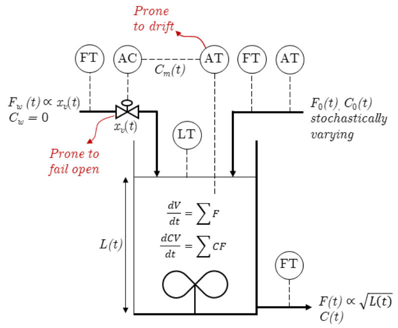


Maintenance module



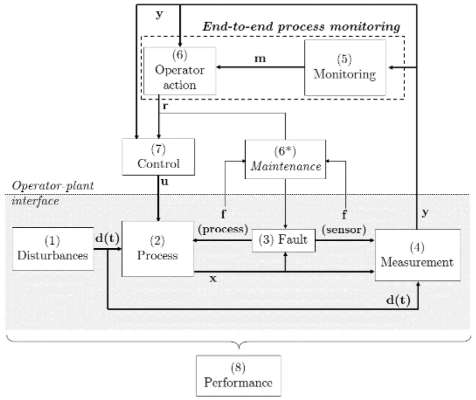
Implementation

- During maintenance, a component is check and potentially replaced/fixed if:
  - The maintenance shut is for **all** components
  - The maintenance shut is for the **indicated type of component**
  - The component is **flagged** by the monitoring model and a maintenance shut has been initiated by operator action
- A **replaced/fixed** component is updated in terms of:
  - Fault state set to ‘none’
  - Fault flag removed
  - In case of a sensor drift fault: drift reset to zero
- The total **maintenance time** is calculated based on:
  - The **minimum maintenance duration**
  - The sum of the time to **check** all flagged/scheduled components
  - The sum of the time to **replace/fix** all checked components that were found to be faulty
- The maintenance module moves the simulation time forward to the time before shut plus the maintenance duration



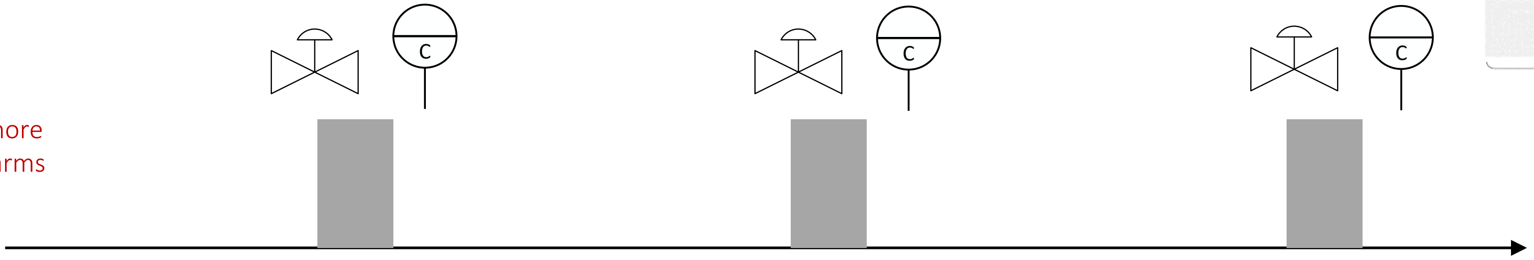
Maintenance module

Examples of different maintenance strategies (integrated with supervisory control)



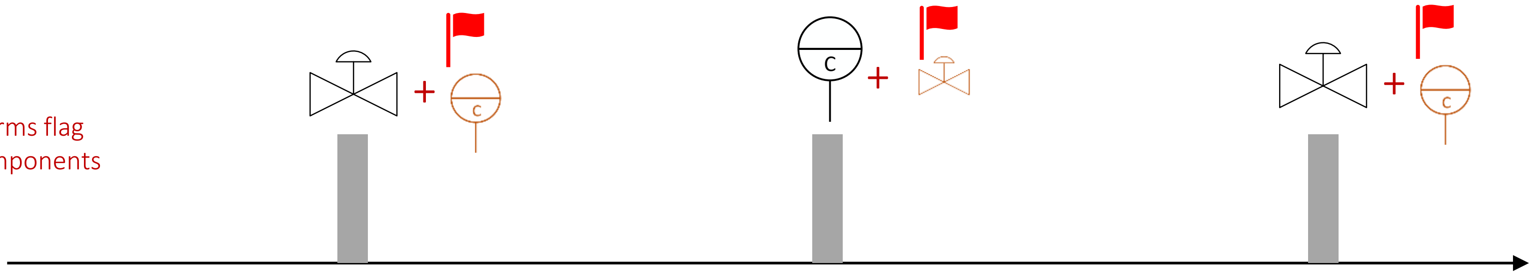
Case 1

Ignore alarms



Case 2

Alarms flag components

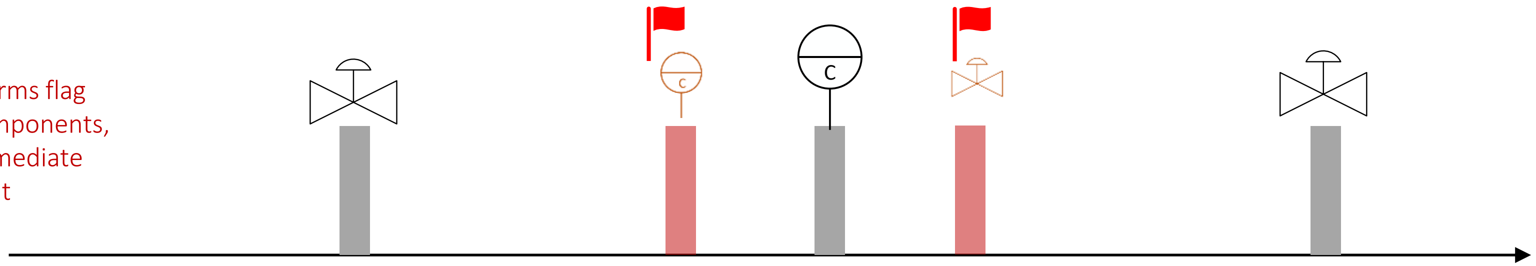


Planned maintenance

+ flagged components

Case 3

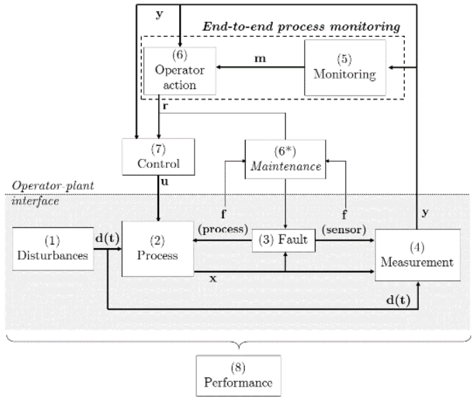
Alarms flag components, immediate shut



Planned maintenance

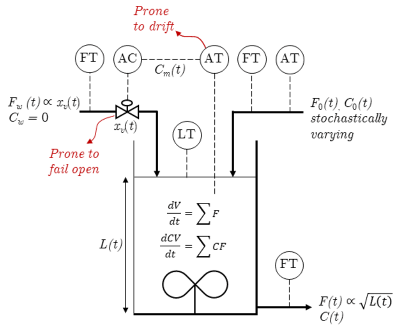
Unplanned/ immediate maintenance

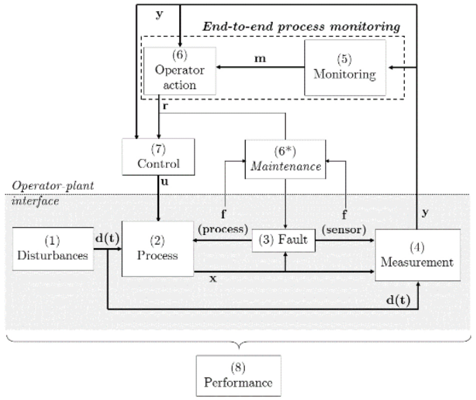
Performance module



Motivation and details

- Motivation:
  - The goals of plant operation is safe, environmentally friendly, and profitable outcomes
  - End-to-end process monitoring design should target the same goals as plant operation
  - Tracking plant performance for extended periods (typical of planning, scheduling, corporate reporting cycles) ensures a representative summary of normal and abnormal condition handling
- Details:
  - The *performance model* simulates the checking and replacement of faulty components
  - The inputs to the performance model are:
    - The true operating condition and outputs of the plant (e.g., process states, fault condition, maintenance actions, etc.)
  - The outputs of the maintenance model are:
    - A time series of performance key performance indicators

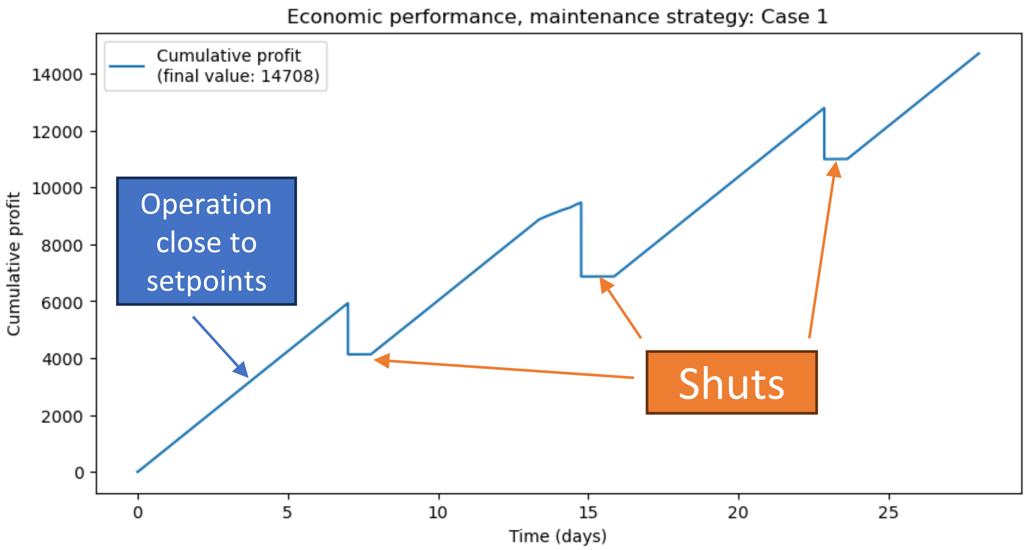


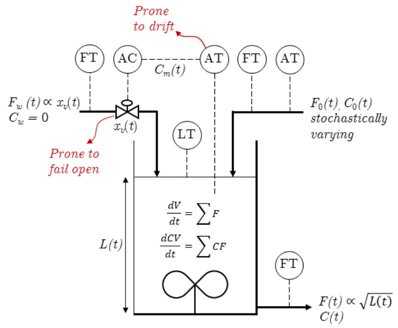


Performance module

Implementation

- A simple example of economic performance is considered, with **revenue** and **cost** items
- **Costs** that occur during maintenance can be explicitly considered, as a measure of the impact of incorrect interventions (overly sensitive fault detection and incorrect diagnosis)
  - E.g., *planned* maintenance costs (250 monetary units per hour of shut) and *unplanned* maintenance costs (350 monetary units per hour of shut) – unplanned shuts are typically costlier
- **Revenue** is earned through producing products on specification, typically per produced unit (penalties may be incurred for off-spec product, and discrete product quality price groups may limit the benefit of maximizing product quality)
  - E.g., $R = \exp(-40(C - C_{SP})^2)$ monetary units per Δt under regimes that are not shuts





Case Study

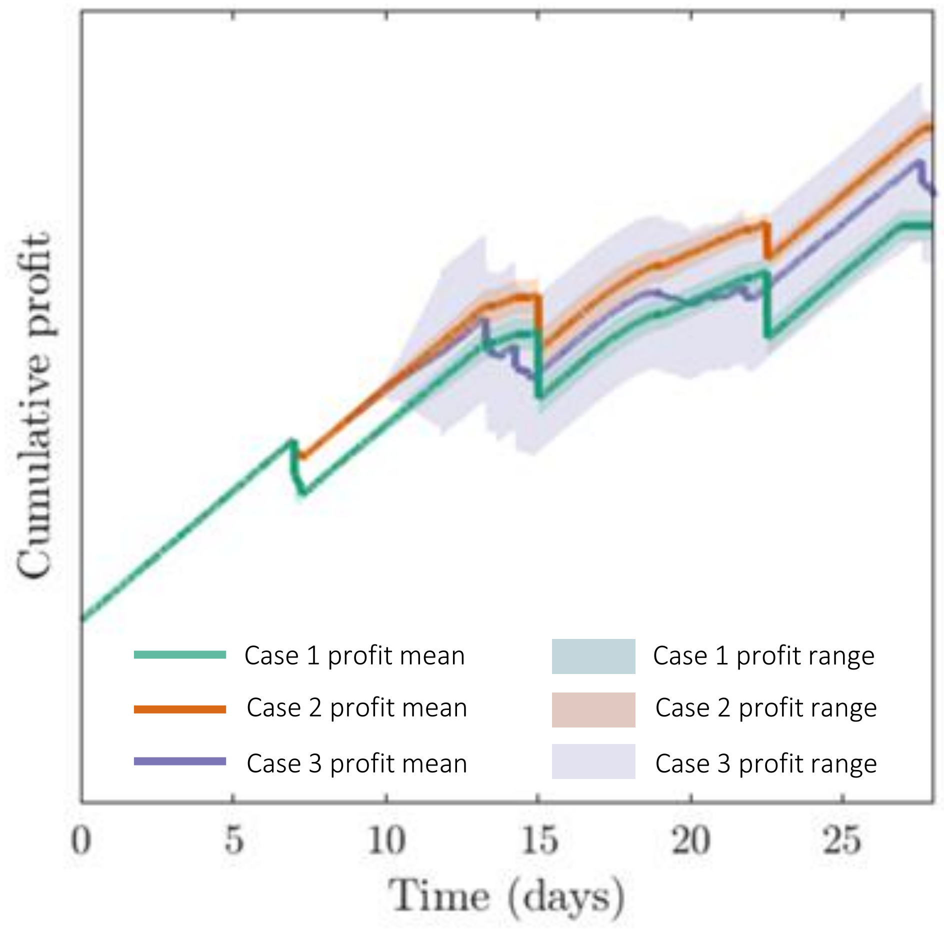
Repeated simulation results

- 50 repeats
- Each repeat:
  - 28 simulation days
  - Different stochastic disturbances
  - Different monitoring training data
  - Different fault manifestations

Case 1 — Ignore alarms — Planned maintenance

Case 2 — Alarms flag components — Planned maintenance + flagged components

Case 3 — Alarms flag components, immediate shut — Planned maintenance, Unplanned/ immediate maintenance



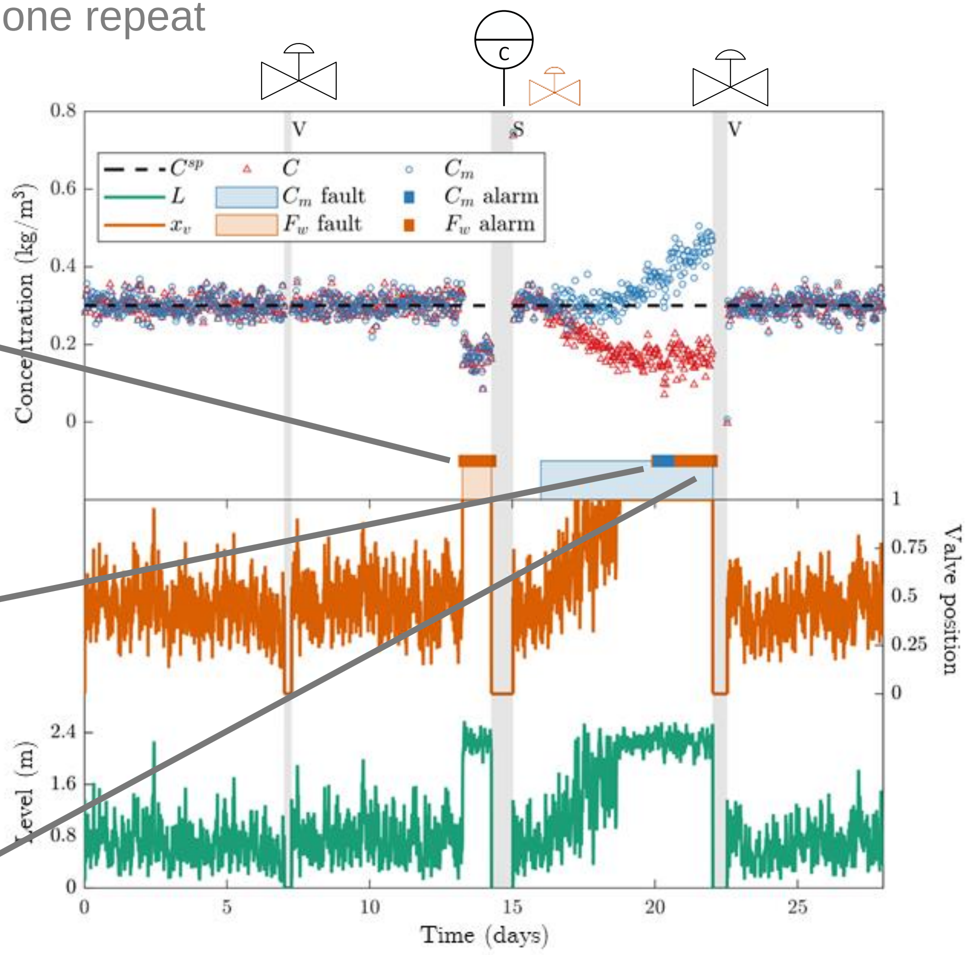
Case Study

Example of one repeat

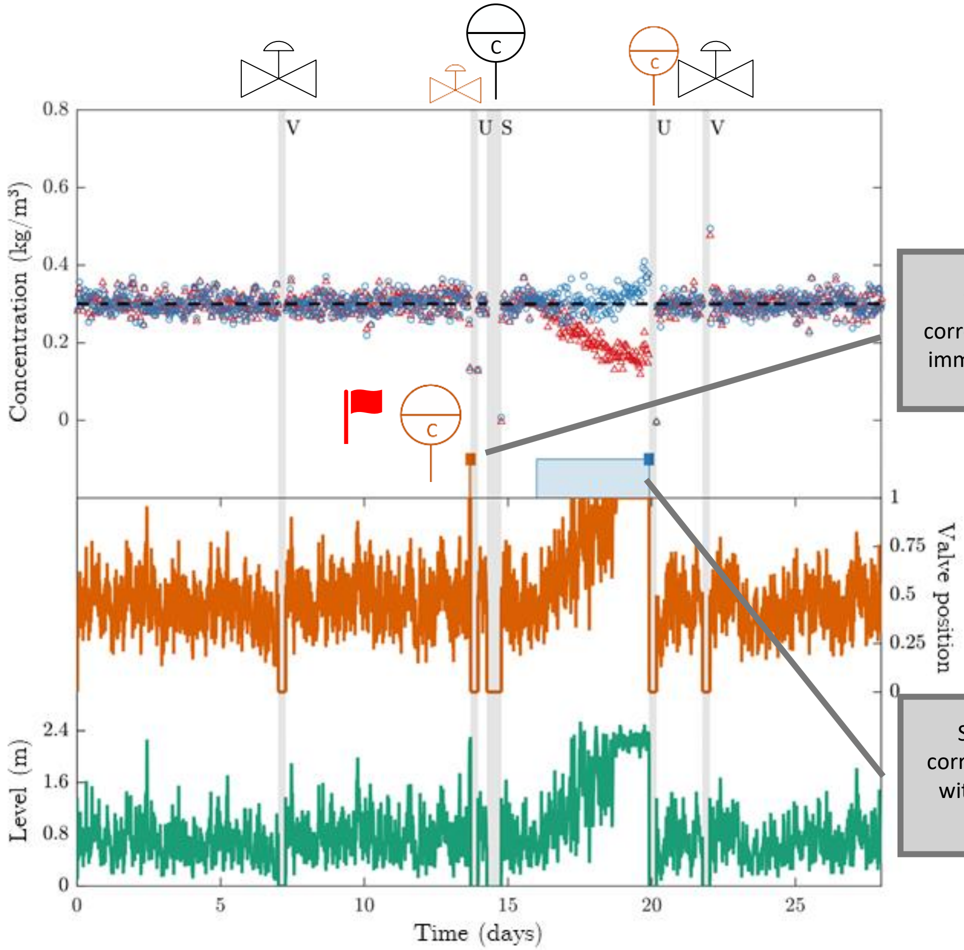
Valve fault correctly detected, with short delay

Sensor fault correctly detected with long delay

Valve fault incorrectly detected



Case 2	
Planned maintenance	+ flagged components



Valve fault correctly detected, immediately fixed

Sensor fault correctly detected with long delay, then fixed

Case 3	
Planned maintenance	Immediate maintenance

Framework familiarization and experiments

Interactive session

Setup and familiarization

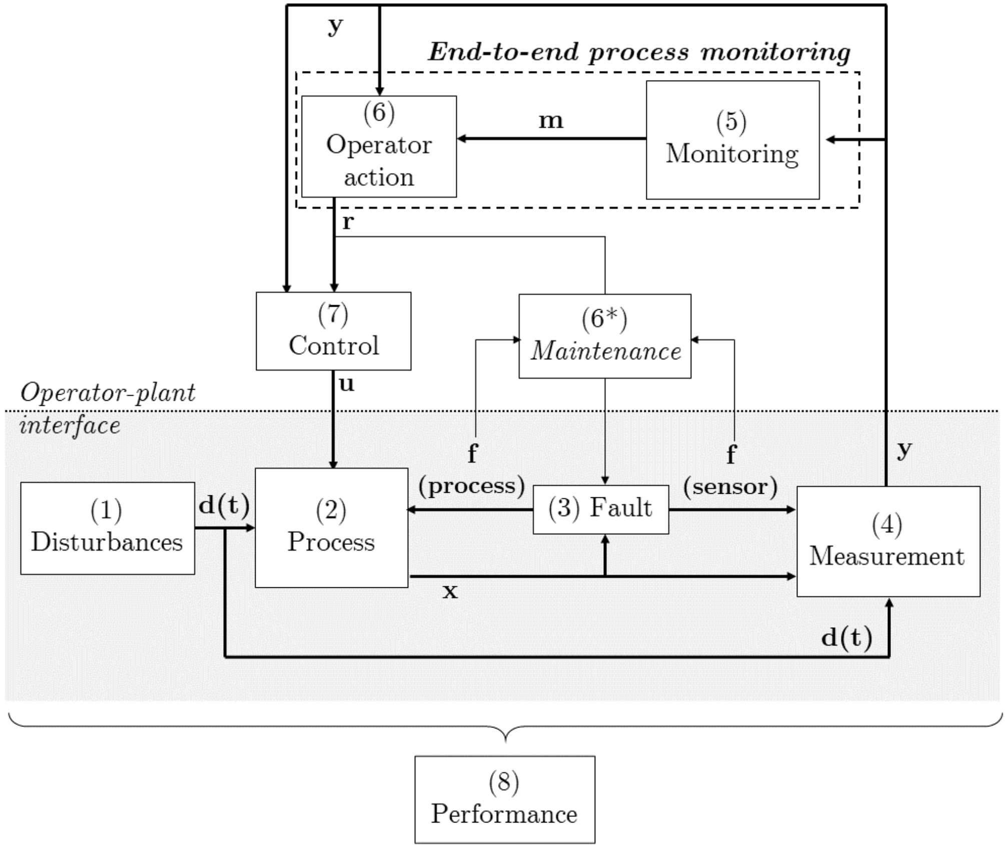
Goals and exercises

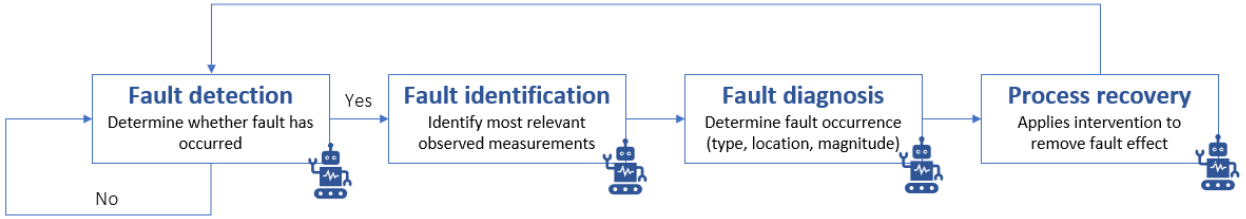
- Run the illustrative examples in Python (optional) for the following models:
  - Regulatory control model
  - Monitoring model
  - Supervisory control model
  - Maintenance model
- **Exercise**: Change the PI tuning constants and observe the effect
- **Exercise**: Change the thresholds for the monitoring model and observe the effect
- **Exercise:** Switch between different maintenance strategies (case 1, 2 and 3) and observe the effect
- **Challenge**: Propose and test an improved monitoring model and maintenance strategy

Conclusions

End-to-End Process Monitoring Framework

- Process Monitoring is only effective when interventions are implemented and improves financial, environmental, and safety indicators
- E2E-PM framework improves process monitoring testing:
  - Holistic evaluation
  - Reproducibility and extendability
  - Global performance criteria
  - Active interventions available
  - Fault and process variability
- Challenges:
  - Implementation effort
  - Scaling to complex, integrated processes
- Future work:
  - Expansion of case studies and monitoring approaches
  - Exploitation by reinforcement learning





The Future of Work. Now.

www.stonethree.com

+27 21 851 3123

info@stonethree.com

24 Gardner Williams Avenue

Paardevlei Somerset West

South Africa 7130

Additional details

Fault modelling with failure rates

- **Failure rate** λ is the frequency with which a component fails
  - The failure rate is the total number of failures within an item population, divided by the total time expended by that population, during a particular measurement interval under stated conditions
- Failure rate can vary over the life cycle of a component
- Failure rate is thought of as the probability that a failure occurs in a specified interval given no failure before time t
- BUT: failure rate is not a probability (its value can exceed 1)
- **Failure distribution** $F(t)$ is a cumulative distribution function that describes the probability of failure up to and including time t
  - $P(T \leq t) = F(t)$ where T is the failure time
- **Reliability function** $R(t)$ (also known as survival function) is the probability of no failure before time t
  - $R(t) = 1 - F(t)$
- **Hazard function/rate** $h(t)$ is the failure rate over small intervals of time (as $\Delta t \to 0$)
  - $h(t) = \lim_{\Delta t \to 0} \frac{R(t) - R(t + \Delta t)}{\Delta t R(t)}$