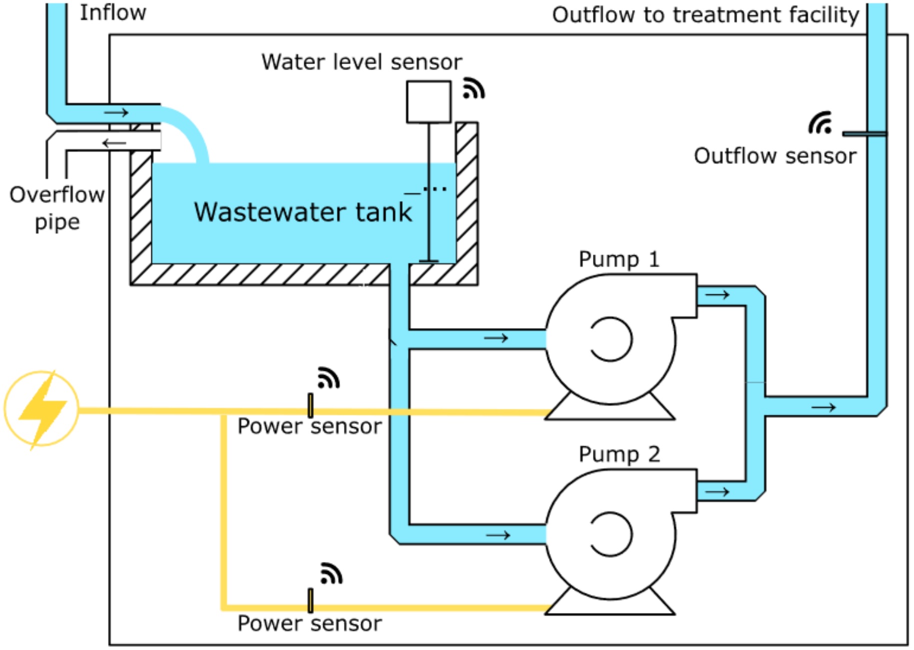
Industrial IoT for Digitization of Electronic Assets

Model Predictive Control to Minimize CO_2 Emission of Wastewater Pumping Station

Objectives

- Understand and analyse the key system parameters and model the interrelation
- Develop a pump control strategy using Model Predictive Control
- Reduction of the overall CO_2 emission of the wasterwater pumping station



Development

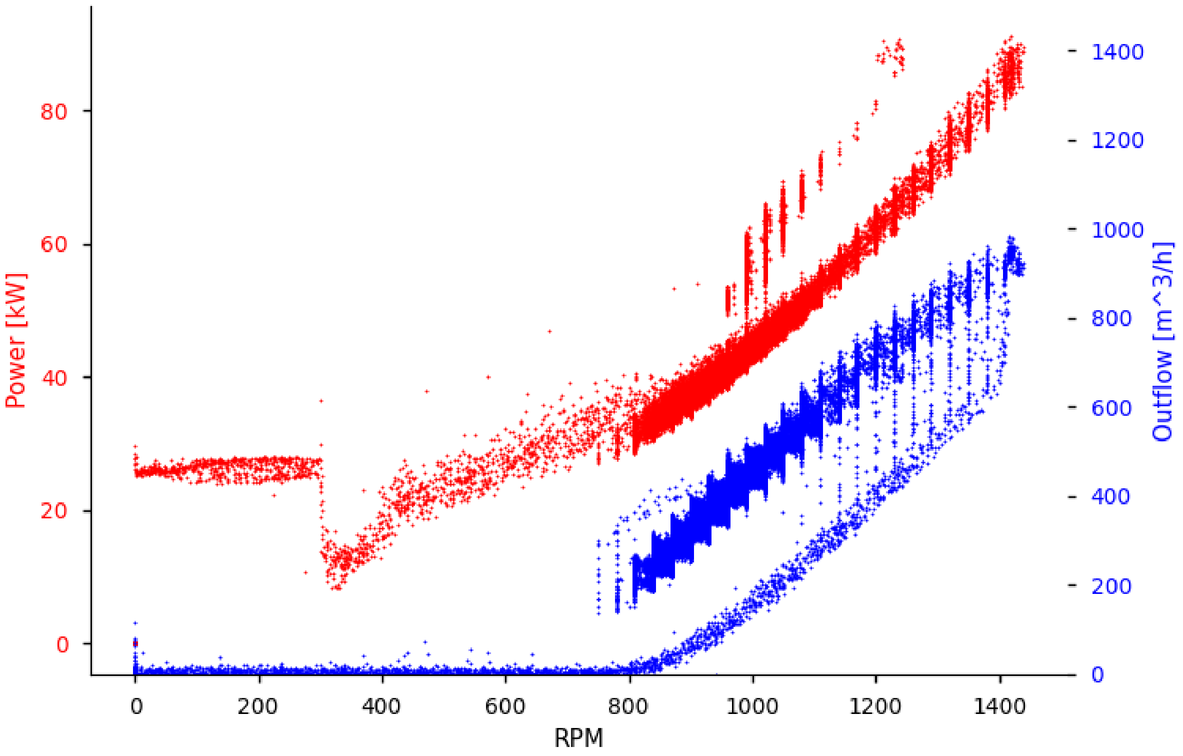
Data Collection
- Pump RPM, power and outflow
- Water level (height) in the reservoir
- CO_2 emission data from energinet API

Data Processing
- ARX model for RPM and power
- ARX model for RPM and outflow
- Inflow estimation using Kalman filter

Model Predictive Control
- Optimize pump operation
- Pump control according to cost function
- Minimize CO_2 emission

Data Filtration

- Raw file of pump station data consists of two pumps P1 and P4 ➔ only Pump 4 is considered
- Selection of data:
  - **Pump1_rpm = 0 rpm** and **Pump4_rpm > -1**
  - Negative speeds of Pump4 removed
- The Pump4_power vs Pump4_rpm remotely follows quadratic relation
- The outflow vs Pump4_rpm relation follows shifted ramp
- Outlier removal is applied on the subset of the raw data based on the above filters



The Auto Regressive eXogenous Model estimations

ARX model for Outflow Estimation

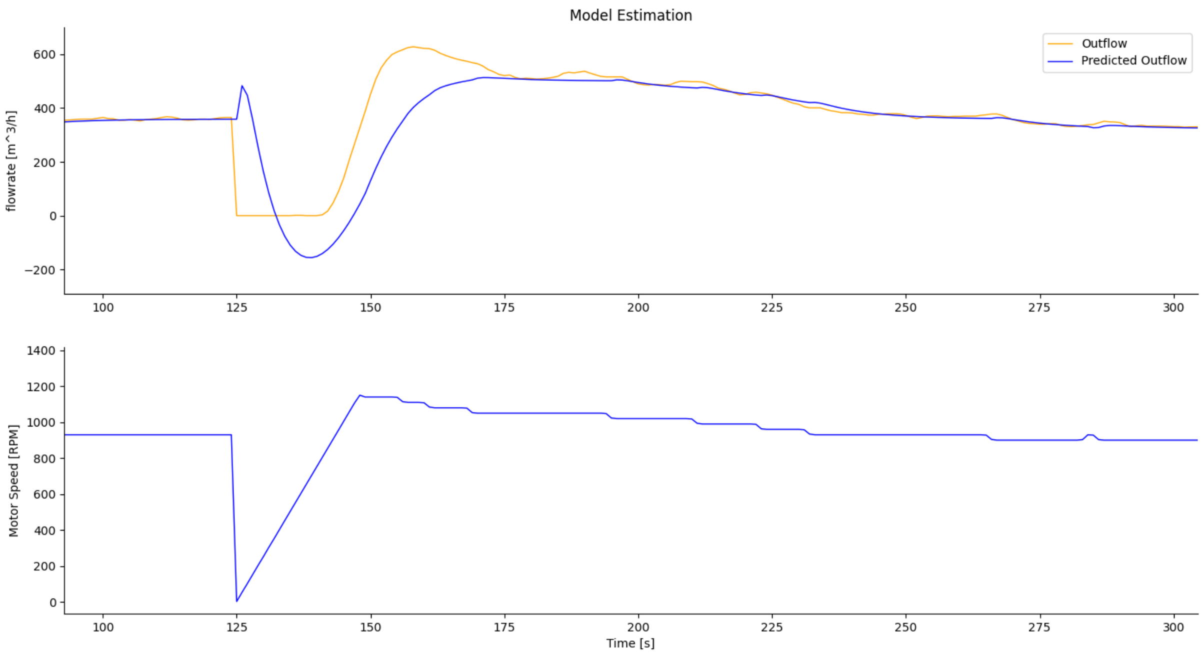
```python
# Building the model
basis_function = Polynomial(degree=1)
model = FROLS(
    order_selection=True,
    n_info_values=10,
    extended_least_squares=False,
    ylag=5,
    xlag=5,
    info_criteria="aic",
    estimator="least_squares",
    basis_function=basis_function, )

# Select One Day of data to build the model
test_data = df[["pump4_rpm", "outflow"]].loc['2023-03-01':'2023-03-02']
train_data = df[["pump4_rpm", "outflow"]].loc['2023-02-27':'2023-03-01']
```

ARX model for Power Estimation

```python
# Building the model
basis_function = Polynomial(degree=1)
model = FROLS(
    order_selection=True,
    n_info_values=10,
    extended_least_squares=False,
    ylag=3,
    xlag=3,
    info_criteria="aic",
    estimator="least_squares",
    basis_function=basis_function, )

# Select One Day of data to build the model
test_data = df[["pump4_rpm", "pump4_power"]].loc['2023-03-01':'2023-03-02']
train_data = df[["pump4_rpm", "pump4_power"]].loc['2023-02-27':'2023-03-01']
```

The Auto Regressive eXogenous Model estimations

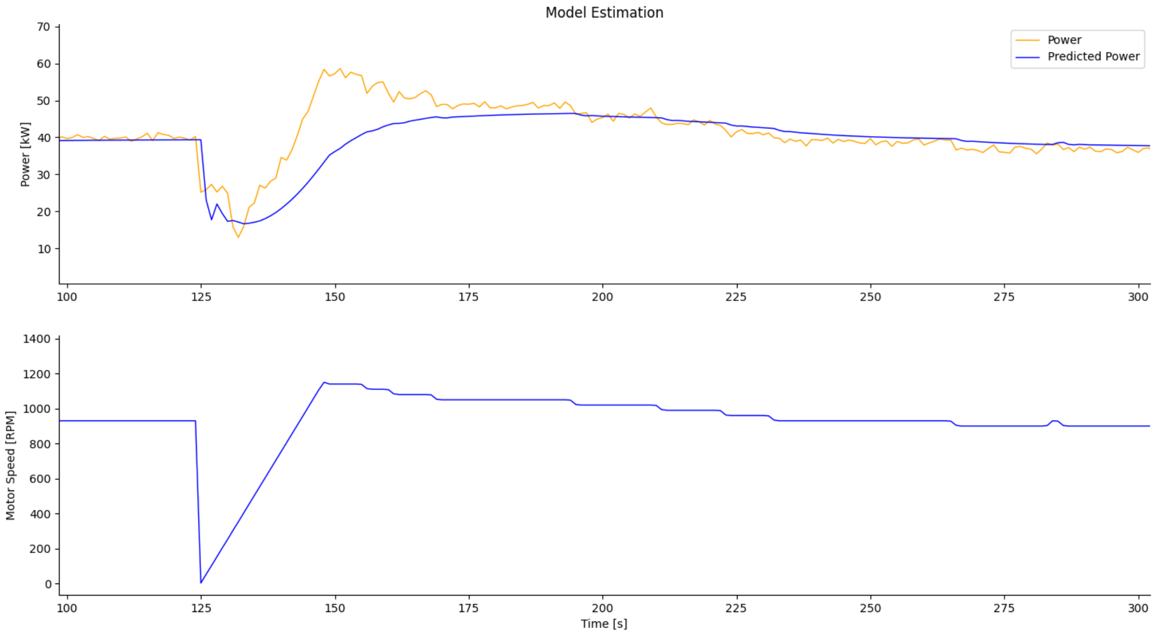
ARX model for Outflow Estimation



Errors ➔

```
MAE = 12.07615235729746
MSE = 539.7235214538877
RMSE = 23.231950444460914
RRSE = 0.3099301963566659
```

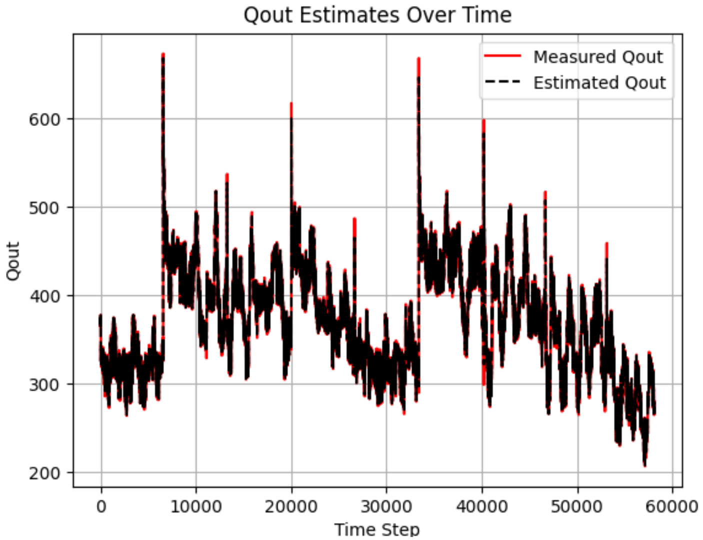
ARX model for Power Estimation

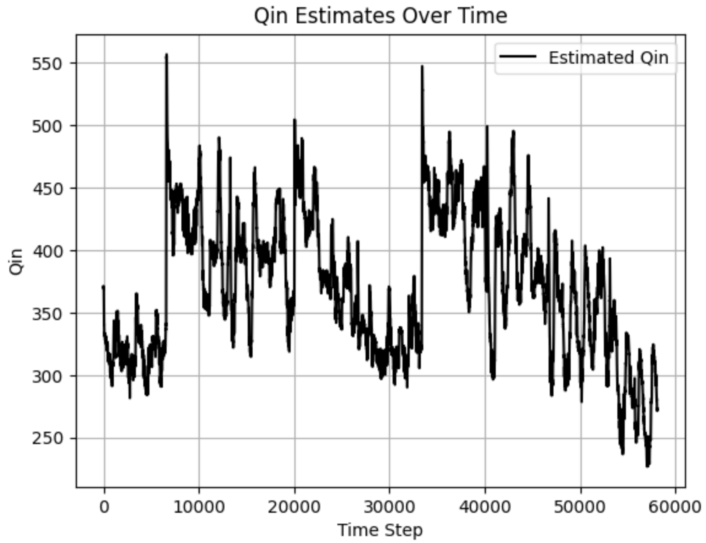


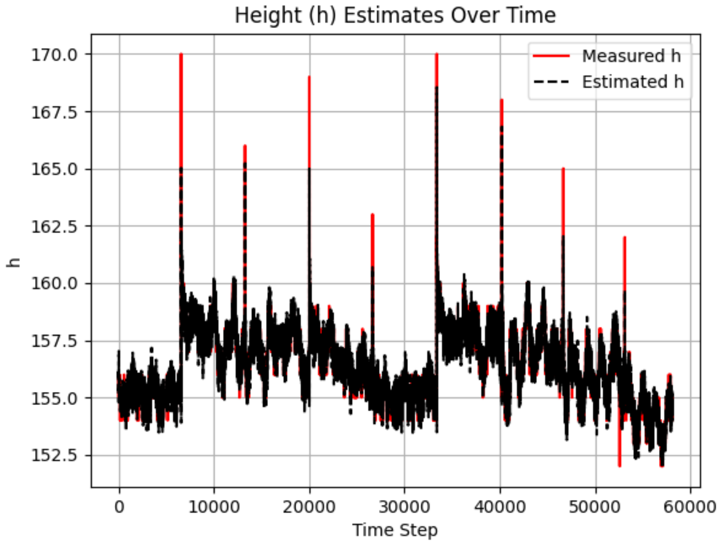
Errors ➔

```
MAE = 1.0309321749722542
MSE = 3.0204616133895
RMSE = 1.7379475289372086
RRSE = 0.4283954521718611
```

Inflow Estimation using Kalman Filter







- System Model:

$$\dot{h} = \frac{1}{A}(Q_{in} - Q_{out})$$

- Estimate inflow indirectly using a Kalman filter.
- State variables:

$$\boldsymbol{x} = [Q_{in} \quad Q_{out} \quad h]^T$$
$$\boldsymbol{y} = [Q_{out} \quad h]^T$$

  - Height h
  - Inflow rate Q_{in}
  - Outflow rate Q_{out}

- State Transition Matrix:

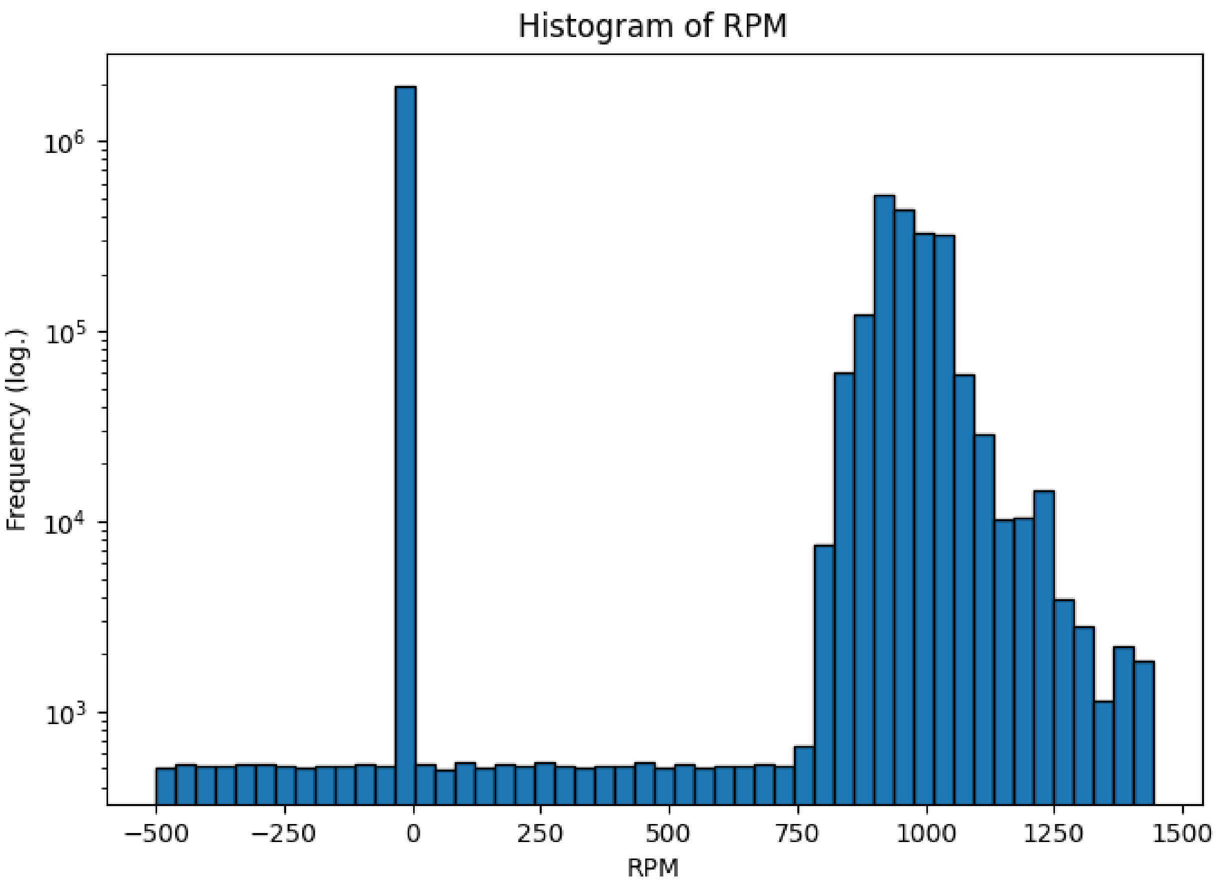
$$\boldsymbol{F} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{Ts}{A} & -\frac{Ts}{A} & 1 \end{bmatrix}$$

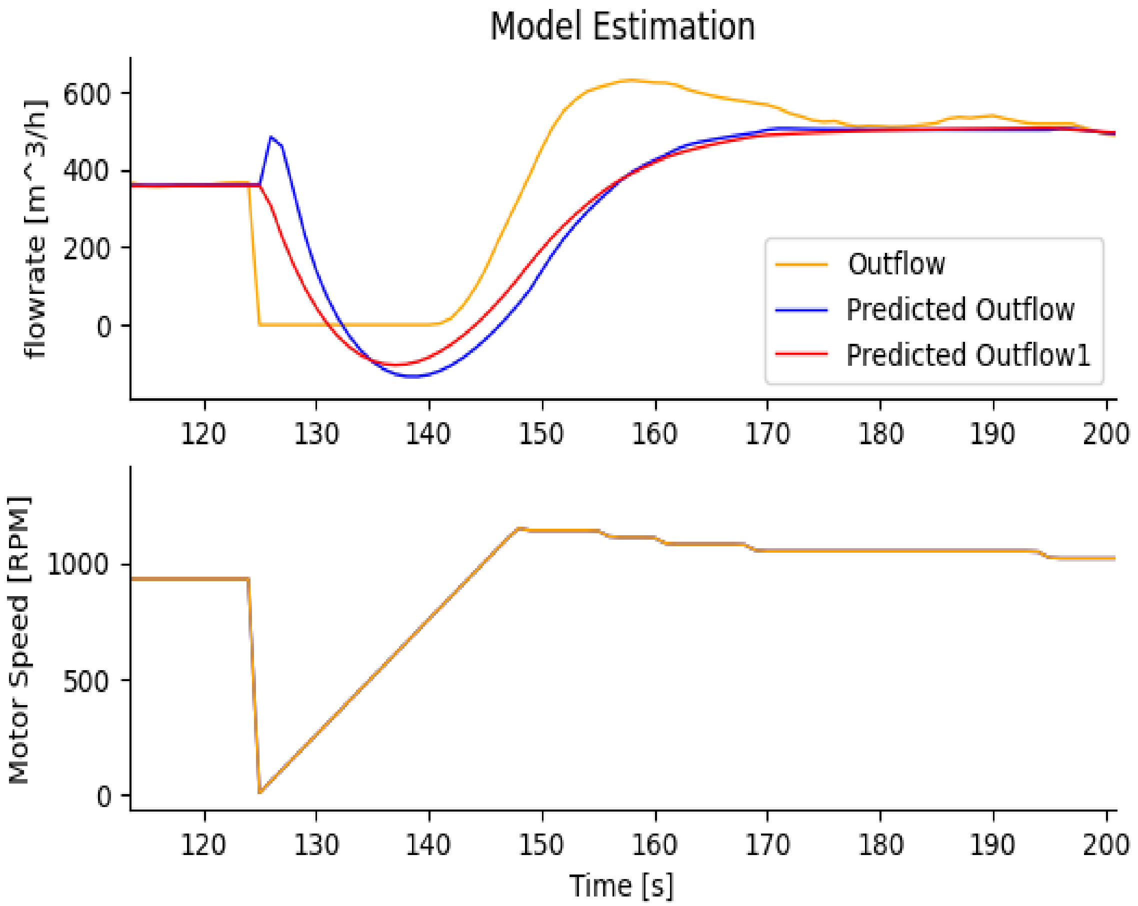
- Observation Matrix:

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Observed problems with the generated models

- Uneven data set distribution ➔The data points for Pump4_rpm < 750 is much smaller than the data points for pump4_rpm > 750 rpm ➔ System ARX model is ill-defined for lower speeds of the Pump





Observed problems with the generated models

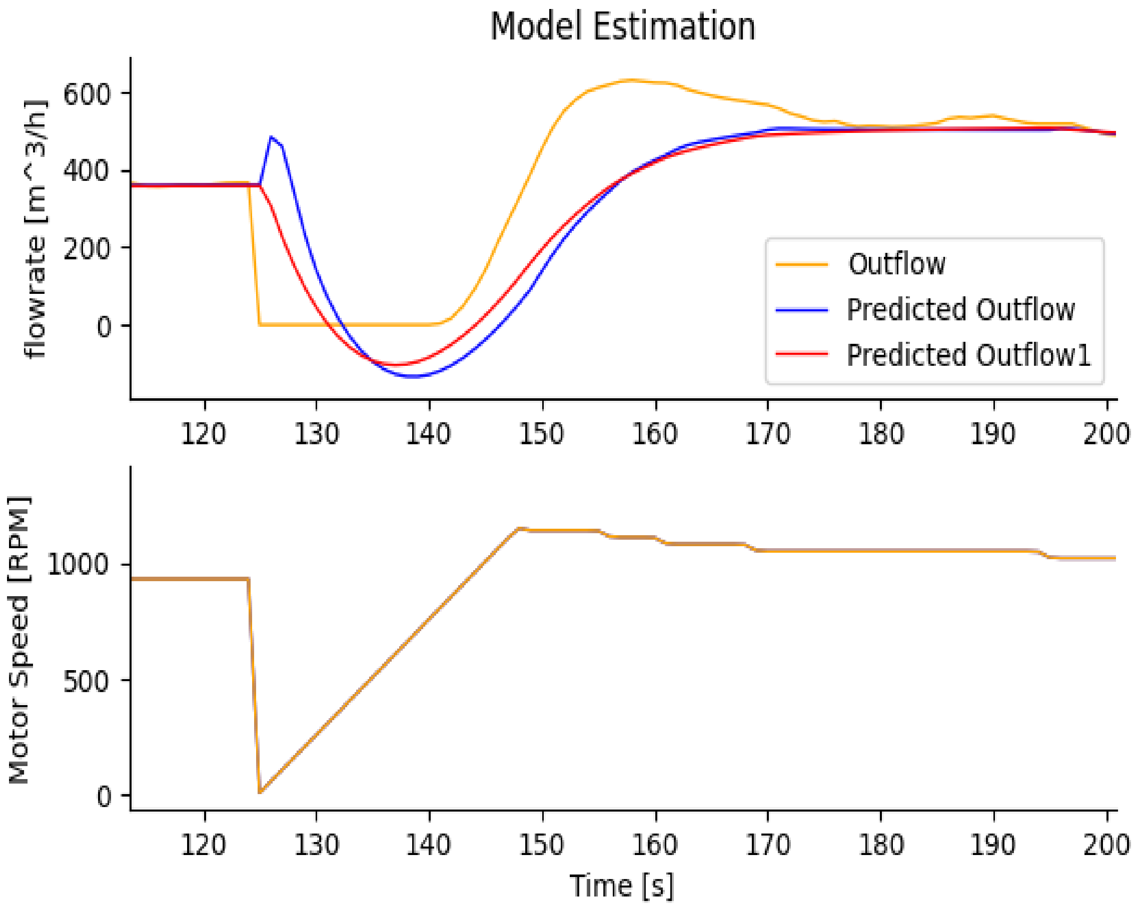
- Uneven data set distribution ➔The data points for Pump4_rpm < 750 is much smaller than the data points for pump4_rpm > 750 rpm ➔ System ARX model is ill-defined for lower speeds of the Pump

➔The time plot shows the model prediction for two different data sets

Case 1: The previous ARX model with actual ill-defined data.

Case 2: The outflow is set to zero for Pump4_rpm <750

The dynamics during change of speed is removed in the 2nd model – red graph in the figure



Piecewise Linear Function

- Another possible solution to improve model to improve the fit in low speed dynamics :

- Implement the identified ARX model when speed is > 750 RPM else set the outflow $Q_{out} = 0$ in the MPC controller
- This can be applied to avoid manual manipulation of the raw collected data set.

$$(\omega > 750 \ \rightarrow Q_{out} = Q_{out}) \land (\omega \leq 750 \ \rightarrow Q_{out} = 0)$$

The flowrate-height equation in MPC modelling can be written as:

```
m.Equation(h.dt() == 1/18 *(Qin - m.if3(w-750,0,Qout))/3600*100)
```

Conflicts:

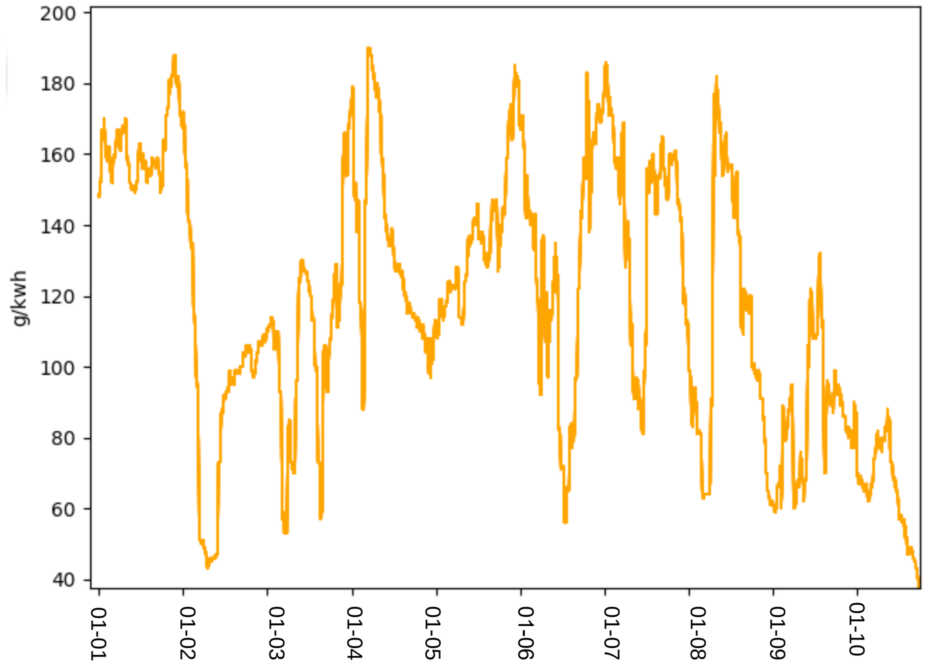
This approach introduces Binary/Integer variables; the problem becomes mixed integer data set which drastically increases complexity and computational time of the solver

Note: The speeds in this code is defined as continuous values ranging from 0 rpm to 1500 rpm rather than intergers ➔ This reduces the computation time of the solver drastically.

CO_2 Emission Prognosis

- Datasource: Energinet API
  https://api.energidataservice.dk/dataset/CO2EmisProg

- Co2 Emission data is selected for the filters:
  - From: 2024-01-01 To: 2024-01-17
  - Price Area DK2 (East of Great Belt)



CO_2 Emission Prognosis

Prognosed CO_2 emission [g/kwh] is available in 5 min resolution during the period creates two **conflicts**

- During a time horizon = 300s, there is no change in the CO_2 Emissions data
- From the obtained data from Energinet API, a minimum of 4-hour time horizon is needed to see significant changes
- The inflow estimation cannot be assumed constant for a 4-hour time horizon which is inapplicable.

- Pump control loop needs high Sampling Frequency (Ts = 1s) to react to changes in inflow
- Optimizing over 4 hour time horizon with 1s resolution is computationally unfeasible

➔ A **solution** is to accelerate CO2 Prognosis to exhibit significant changes in 2 min horizon by compressing 5 min resolutions of Co2Emis data to 1s time step

➔ For a time horizon of 2 minutes, we need 10 hours of CO_2 Emissions data.

Model Predictive Control

- Assumptions:
  - Single Pump
  - Q_{in} const. over optimization horizon N
  - Accelerated CO2 Emission Prog. C_k

  - Continous Pump Speed
    - Avoids Mixed Integer Optimization

  - Sample Time: $T_s = 1\ s$
  - Horizon: $N = 120\ s$ -> 10 h CO_2 Data

$$\underset{\omega \in [0,1500]}{\arg\min} \sum_{k=1}^{N} C_k * P_k + w_h(h) h_k$$

$$
\begin{aligned}
& s.t. \\
& P_k = f_P(\boldsymbol{\omega}) \\
& \dot{h} = \frac{1}{A}(\hat{Q}_{in} - Q_{out}) \\
& Q_{out,k} = f_{Qout}(\boldsymbol{\omega}) \\
& \hat{Q}_{in} = KF(\boldsymbol{F}, \boldsymbol{H}, h, Q_{out}, \omega)
\end{aligned}
$$

Model Predictive Control

1. Solver Setup

```
m = GEKKO(remote=False)
m.options.SOLVER = 1 # APOPT solver
m.options.IMODE = 6 # control
m.TIME_SHIFT = 1

# time horizon: 2min -> 10h CO2
Thor = int(2*60) #sec
Ts = 1 #sec
n = Ts*Thor
m.time = np.linspace(0,int(Thor-1),n)
```

2. Variables

```
# Manipulated variable
w = m.MV(value=0, lb = 0, ub = 1500,integer=False)
w.STATUS = 1    # allow optimizer to change
w.DCOST = 0.07  # Penalize Changes in Pump Speed

# Controlled Variable
Qout = m.CV(value=0)
P = m.CV(value=0)
h = m.CV(value=170)

# Parameters: CO2 and Qin
c = m.Param()
Qin = m.Param()
```

3. Objectives

```
# Objective Function (parts of it)
m.Minimize(c*P)

m.options.CV_TYPE = 1 # Linear error with deadband

# eH = h-200
eH = m.CV(value=0)
eH.SPHI=0
eH.WSPHI=10000    # Penalty on exceeding 200cm
eH.WSPLO=0        # Penalty on losing to 200cm
eH.STATUS =1

# eL = h-120
eL = m.CV(value=0)
eL.SPLO=0
eL.WSPHI=2        # Penalty on exceeding to 120cm
eL.WSPLO=15       # Penalty on falling below 120cm
eL.STATUS = 1
```

Model Predictive Control

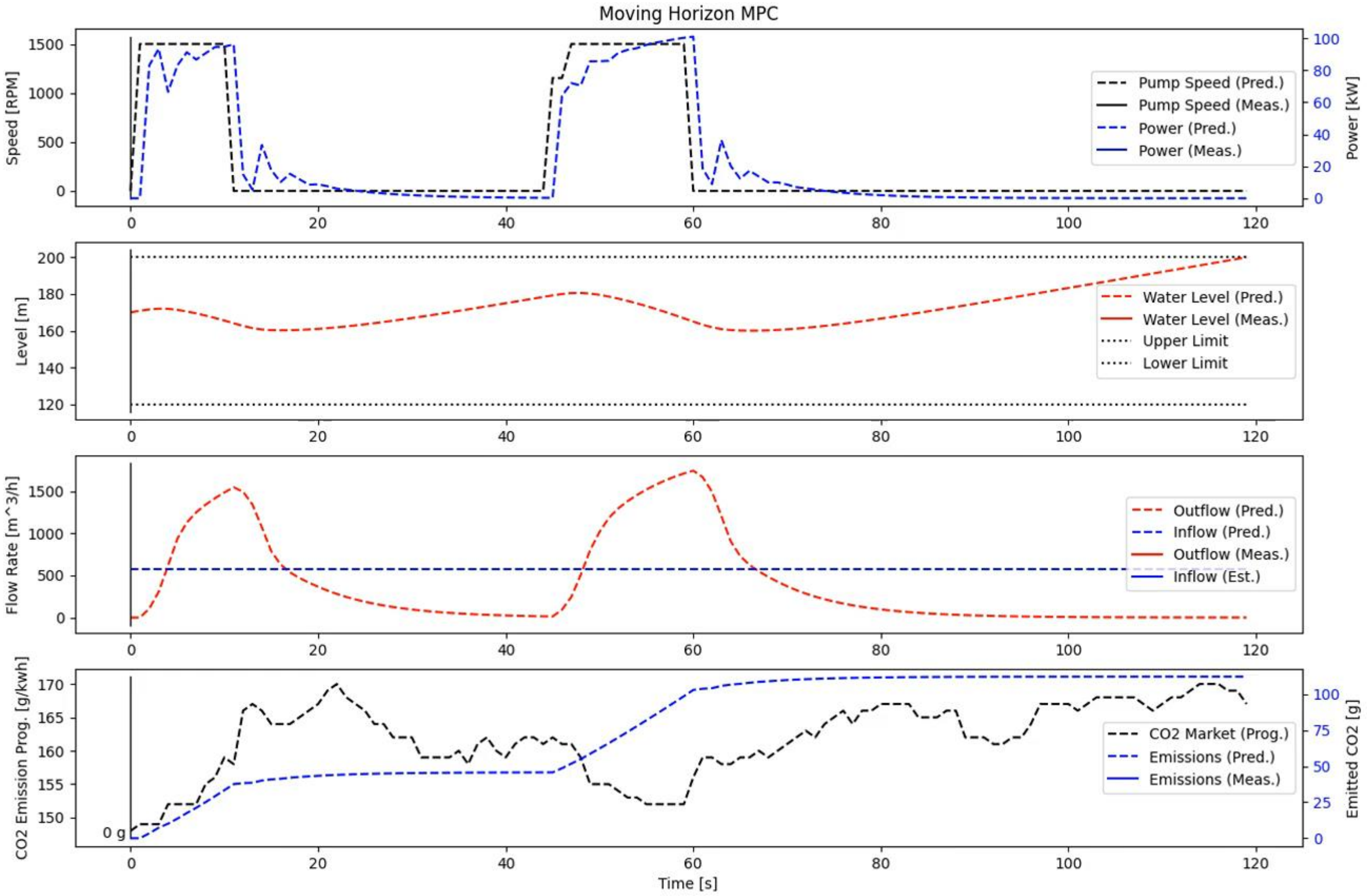
4. System Dyamics

```python
p_power = {'a':A_power,'b':B_power,'c':C_power}
p_outflow = {'a':A_outflow,'b':B_outflow,'c':C_outflow}

# Creating Arx Models
m.arx(p_power,P,w)       # Power vs. RPM
m.arx(p_outflow,Qout,w)  # Qout vs. RPM

# System Equations
m.Equation(h.dt() == 1/18 *(Qin - Qout)/3600*100)
emissions = m.Var() # Cumulated CO2 Emissions
m.Equation(emissions == m.integral(c/3600*P))
m.Equations([eH==h-200,eL==h-120]) # Errors
```

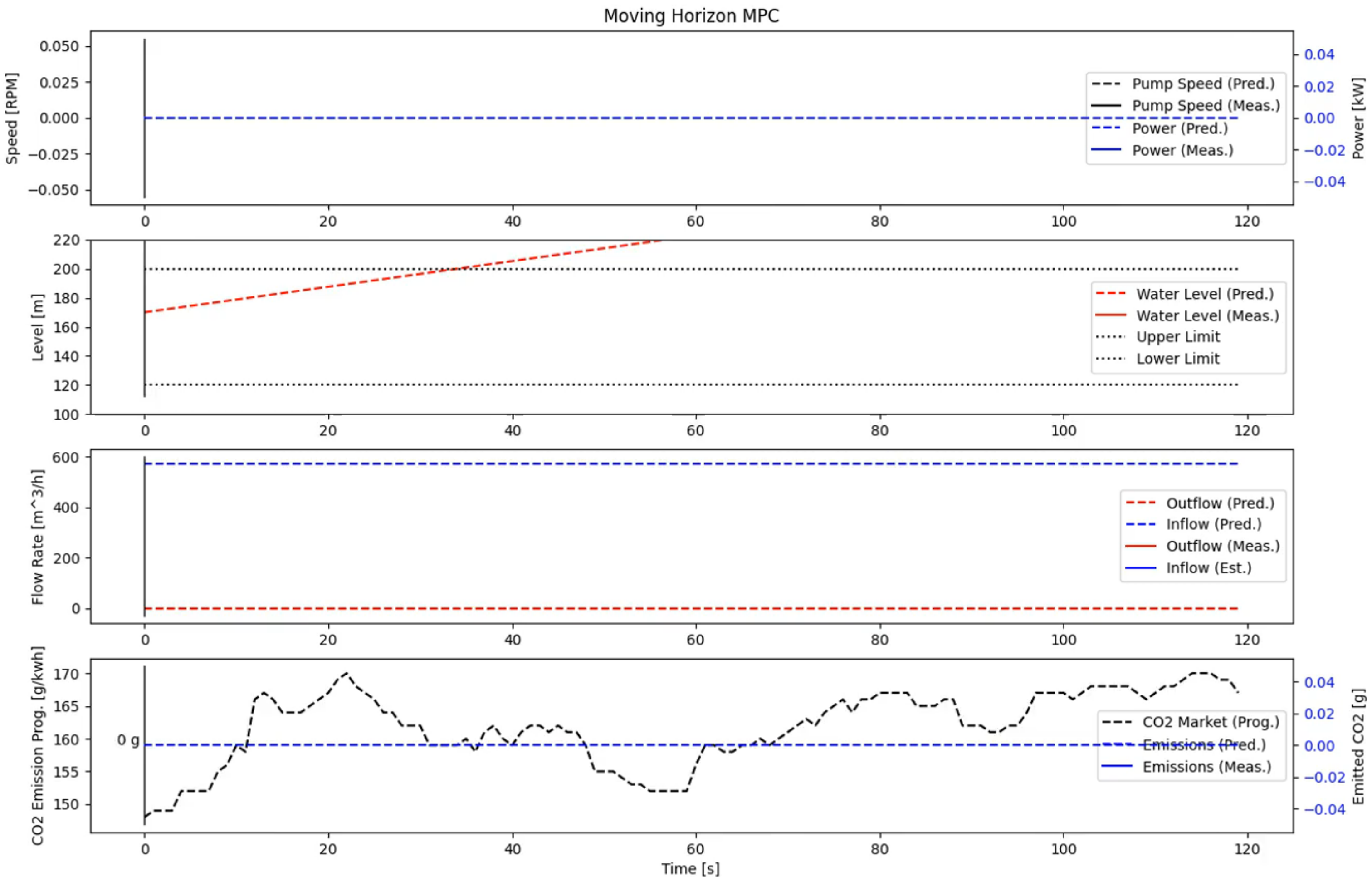
5. Iterate

```python
simtime = 250
for i in range(0,simtime):
    # Set CO2 Prognosis
    c.value = df_co2["CO2Emission"][i:i+n]

    # Get Qin estimate from Kalman Filter
    Qin.VALUE = Qin_v[i]

    m.solve(disp=False)

    # Set RPM to previous value
    w.MEAS = w.NEWVAL
```

Results


Moving Horizon MPC
Speed [RPM]
Power [kW]
Pump Speed (Pred.)
Pump Speed (Meas.)
Power (Pred.)
Power (Meas.)
Level [m]
Water Level (Pred.)
Water Level (Meas.)
Upper Limit
Lower Limit
Flow Rate [m^3/h]
Outflow (Pred.)
Inflow (Pred.)
Outflow (Meas.)
Inflow (Est.)
CO2 Emission Prog. [g/kwh]
Emitted CO2 [g]
0 g
CO2 Market (Prog.)
Emissions (Pred.)
Emissions (Meas.)
Time [s]

Comparison to Conventional Controller

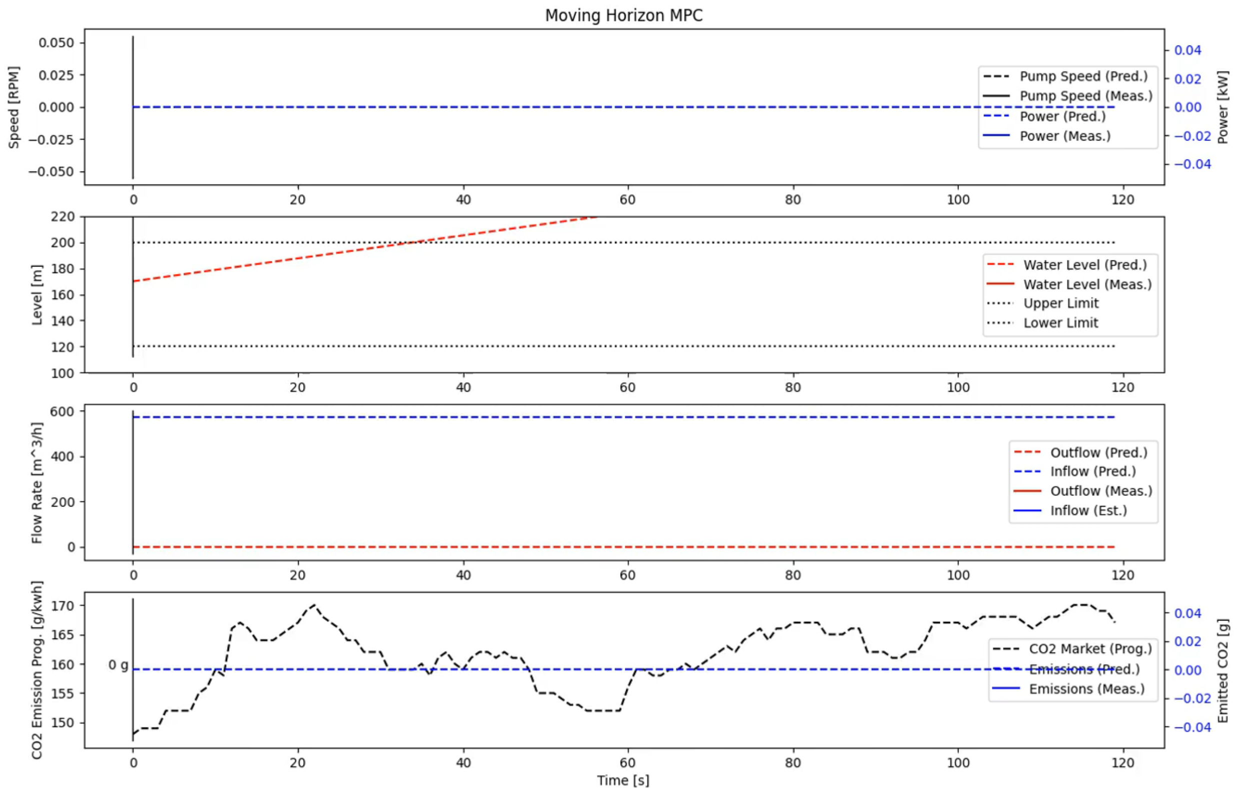


Comparison to Conventional Controller

- Over 250 iterations (s):
  - Bang-Bang (Hysteresis) Controller:
    - 371 g CO_2
  - MPC Controller:
    - 333 g CO_2 (͡~ ͜ʖ ͡°)

  ➔**10 %** decrease in CO_2 emission

Leaves room for improvement



Conclusions

- The MPC control strategy succesfully optimizes pump operation and the comparision to a traditional controller on reduction of CO_2 emissions is measured
- … while rejecting external disturbances

- Further improvements:
  - Tuning of weights/penalties assigned to the parameter limits
  - Implement a more refined model taking into account all dynamics of the system and improved data filtration
  - Set a mesaured input – speed as the w.MEAS in the iterator to take into account the actual variation of the input speed by the controller compared to the predicted speed.

- To make feasible realtime controller:
  - Reduce solve time to < 1s
  - Increase optimization horizon to > 6h
  - Compute reference height over entire horizon based on CO_2 prog. and inflow est.
  - MPC only follows precomputed reference

- Realtime implementation through Imitation Learning