MODEL PREDICTIVE CONTROL

DATA-DRIVEN MPC

Alberto Bemporad

imt.lu/ab

COURSE STRUCTURE

- ✔ **Basic concepts** of model predictive control (MPC) and **linear MPC**
- ✔ **Linear time-varying** and **nonlinear** MPC
- ✔ MPC computations: **quadratic programming** (QP), **explicit** MPC
- ✔ Hybrid MPC
- ✔ Stochastic MPC
- Data-driven MPC

Course page:

`http://cse.lab.imtlucca.it/~bemporad/mpc_course.html`

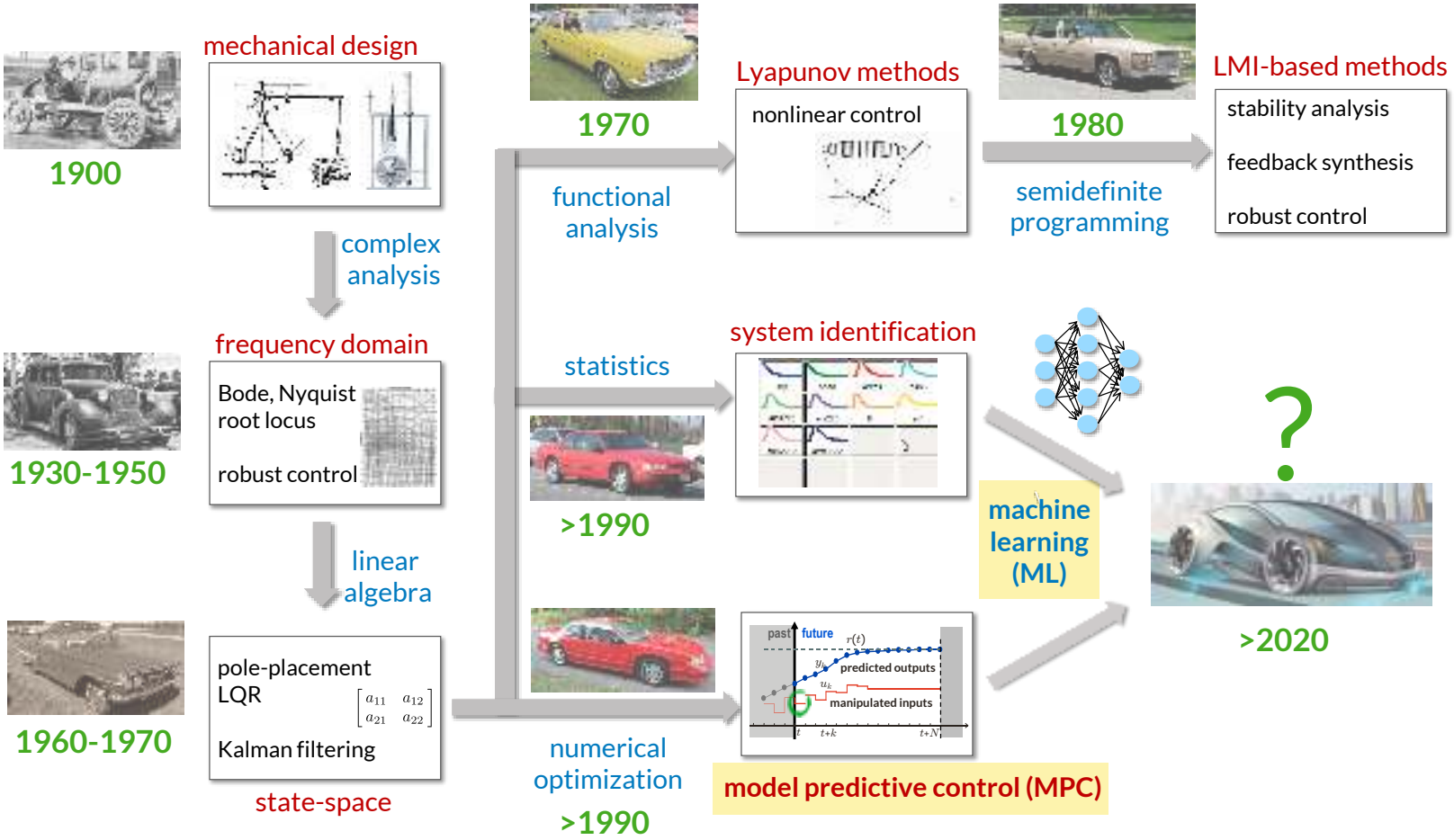
MPC DESIGN FROM DATA

1. Use **system identification**/**machine learning** to learn a **prediction model** from data

2. Use **reinforcement learning** to learn the **MPC law** from data

   - **Q-learning**: learn Q-function defining the MPC law from data
     (Gros, Zanon, 2019) (Zanon, Gros, Bemporad, 2019)
   - **Policy gradient methods**: learn optimal policy coefficients directly from data using stochastic gradient descent (Ferrarotti, Bemporad, 2019)
   - **Global optimization methods**: learn MPC parameters (weights, models, horizon, solver tolerances, ...) by optimizing observed closed-loop performance
     (Piga, Forgione, Formentin, Bemporad, 2019) (Forgione, Piga, Bemporad, 2020) (Zhu, Bemporad, Piga, 2021)

MACHINE LEARNING AND CONTROL ENGINEERING



- **MPC** and **ML** = main trends in control R&D in industry !

MACHINE LEARNING (ML)

- Massive set of techniques to **extract mathematical models from data** for **regression, classification, decision-making**

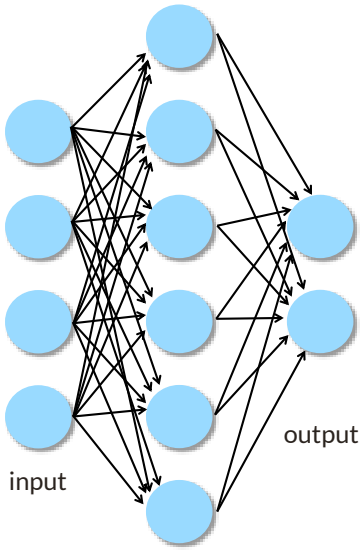
- Good **mathematical foundations** from artificial intelligence, statistics, optimization

- **Works very well** in practice (despite training is most often a nonconvex optimization problem ...)

- Used in myriads of **very diverse application domains**

- Availability of excellent open-source **software tools** also explains success

  `scikit-learn`, `TensorFlow/Keras`, `PyTorch`, ... (python)



LEARNING PREDICTION MODELS FOR MPC

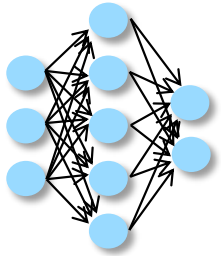
NONLINEAR PREDICTION MODELS

- **Physics-based** nonlinear models (=white-box models)

$$\dot{p}_1 = k_1(W_c + W_{egr} - k_e p_1) + \frac{\hat{T}_1}{T_1} p_1$$
$$\dot{p}_2 = k_2(k_e p_1 - W_{egr} - W_t + W_f) + \frac{\hat{T}_2}{T_2} p_2$$
$$\dot{P}_e = \frac{1}{\tau}(P_c - \eta_m P_t)$$

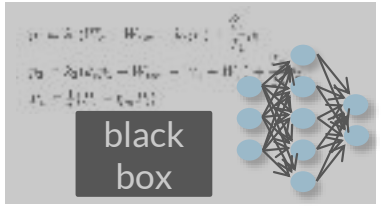
- **Black-box** nonlinear models (**NL SYS-ID/machine learning**)

- A mix of the above (**gray-box** models) is often the best



- **Jacobians** of prediction models are required

- **Computation complexity** depends on chosen model, need to trade off **descriptiveness vs simplicity** of the model



NONLINEAR SYS-ID BASED ON NEURAL NETWORKS

- Neural networks proposed for nonlinear system identification since the '90s

  (Hunt et al., 1992) (Suykens, Vandewalle, De Moor, 1996)

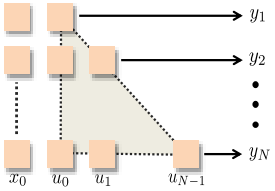
- **NNARX** models: use a **feedforward neural network** to approximate the nonlinear difference equation $y_t \approx \mathcal{N}(y_{t-1}, \ldots, y_{t-n_a}, u_{t-1}, \ldots, u_{t-n_b})$

- **Neural state-space** models:
  - **w/ state data**: fit a neural network model $x_{t+1} \approx \mathcal{N}_x(x_t, u_t), \quad y_t = \mathcal{N}_y(x_t)$
  - **I/O data only**: set x_t = value of an inner layer of the network (Prasad, Bequette, 2003)

- **Recurrent neural networks** are more appropriate for accurate open-loop predictions, but more difficult to train (although not impossible ...)

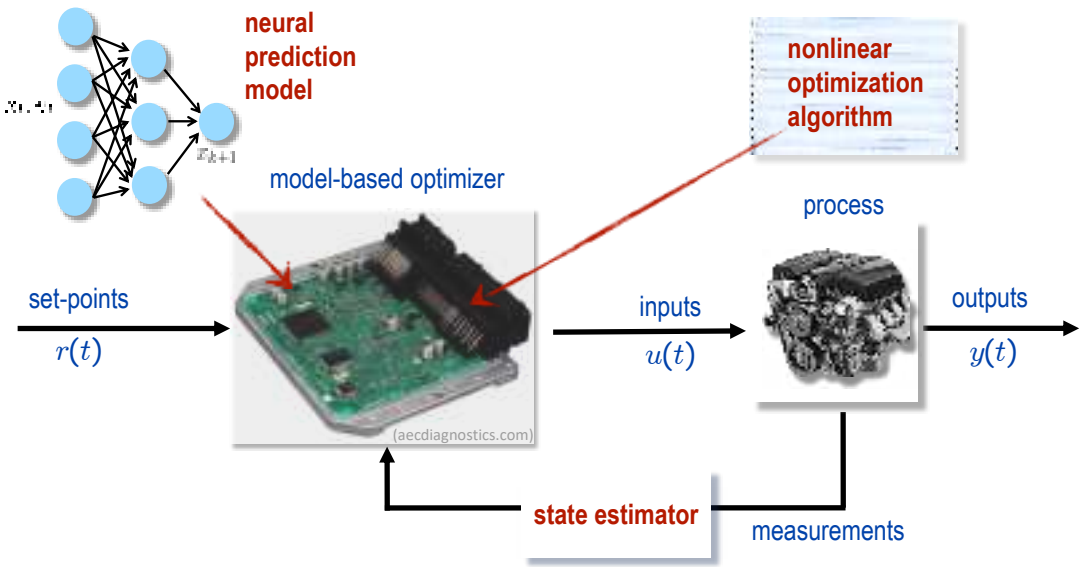
- Alternative for MPC: learn entire prediction (Masti, Smarra, D'Innocenzo, Bemporad, 2020)

$$y_{t+k} = h_k(x_t, u_t, \ldots, u_{t+k-1}), \; k = 1, \ldots, N$$

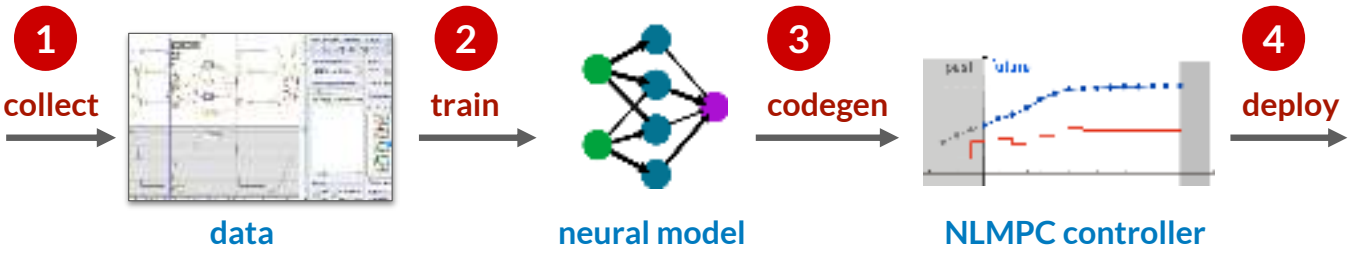


NLMPC BASED ON NEURAL NETWORKS

- **Approach**: use a (feedforward deep) neural network model for prediction

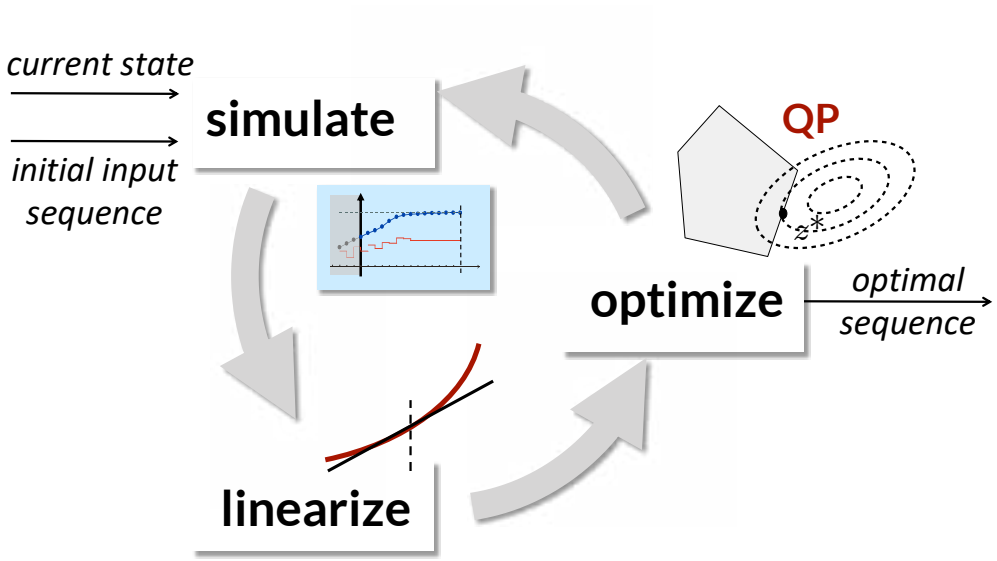


- MPC design workflow:



NONLINEAR MPC

- **Nonlinear MPC**: solve a **sequence of LTV-MPC** problems at each sample step



- **Sequential QP** solves the full nonlinear MPC problem, by using well assessed linear MPC/QP technologies

- One QP iteration is often sufficient (= linear time-varying MPC)

- The current state can be estimated, e.g., by **extended Kalman filtering**

MPC OF ETHYLENE OXIDATION PLANT

- **Chemical process** = **oxidation of ethylene to ethylene oxide** in a nonisothermal continuously stirred tank reactor (CSTR)

$$
\begin{aligned}
&C_2H_4 + \tfrac{1}{2}O_2 \rightarrow C_2H_4O \\
&C_2H_4 + 3O_2 \rightarrow 2CO_2 + 2H_2O \\
&C_2H_4O + \tfrac{5}{2}O_2 \rightarrow 2CO_2 + 2H_2O
\end{aligned}
$$

- **Nonlinear model** (dimensionless variables): (Durand, Ellis, Christofides, 2016)

$$
\left\{
\begin{array}{lcl}
\dot{x}_1 & = & u_1(1 - x_1x_4) \\
\dot{x}_2 & = & u_1(u_2 - x_2x_4) - A_1 e^{\frac{\gamma_1}{x_4}} (x_2x_4)^{\frac{1}{2}} - A_2 e^{\frac{\gamma_2}{x_4}} (x_2x_4)^{\frac{1}{4}} \\
\dot{x}_3 & = & -u_1x_3x_4 + A_1 e^{\frac{\gamma_1}{x_4}} (x_2x_4)^{\frac{1}{2}} - A_3 e^{\frac{\gamma_3}{x_4}} (x_3x_4)^{\frac{1}{2}} \\
\dot{x}_4 & = & \frac{u_1(1-x_4) + B_1 e^{\frac{\gamma_1}{x_4}} (x_2x_4)^{\frac{1}{2}} + B_2 e^{\frac{\gamma_2}{x_4}} (x_2x_4)^{\frac{1}{4}}}{x_1} \\
 & & + \frac{B_3 e^{\frac{\gamma_3}{x_4}} (x_3x_4)^{\frac{1}{2}} - B_4(x_4 - T_c)}{x_1} \\
y & = & x_3
\end{array}
\right.
$$

x_1 = gas density
x_2 = ethylene concentration
x_3 = ethylene oxide concentration
x_4 = temperature in reactor

u_1 = feed volumetric flow rate
u_2 = ethylene concentration in feed

- u_1 = manipulated variables, x_3 = controlled output, u_2 = measured disturbance

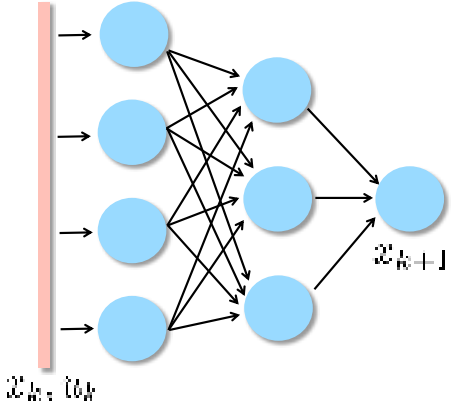
NEURAL NETWORK MODEL OF ETHYLENE OXIDATION PLANT

- Train **state-space neural-network** model

$$x_{k+1} = \mathcal{N}(x_k, u_k)$$

1,000 training samples $\{u_k, x_k\}$
2 layers (6 neurons, 6 neurons)
6 inputs, 4 outputs
sigmoidal activation function

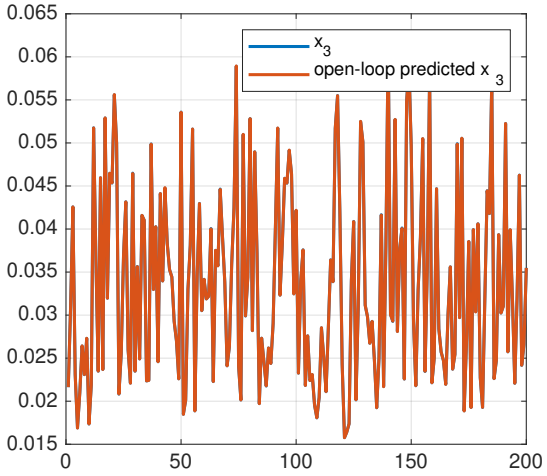
$\rightarrow$ **112 coefficients**

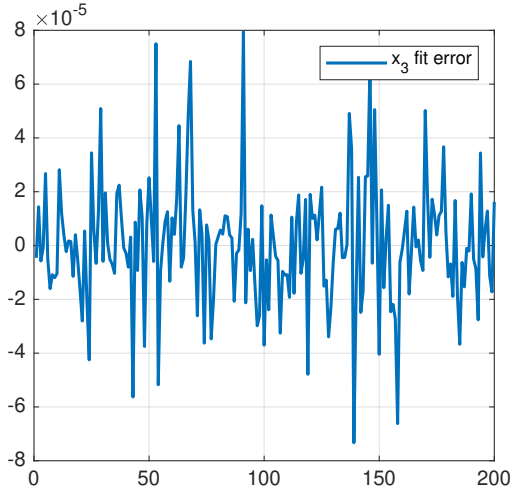


- NN model trained by **ODYS Deep Learning** toolset (model fitting + Jacobians $\rightarrow$ neural model in C)

- Model validated on 200 samples.
  $x_{3,k+1}$ reproduced from x_k, u_k with max 0.4% error

x_3 (validation data)





validation sample

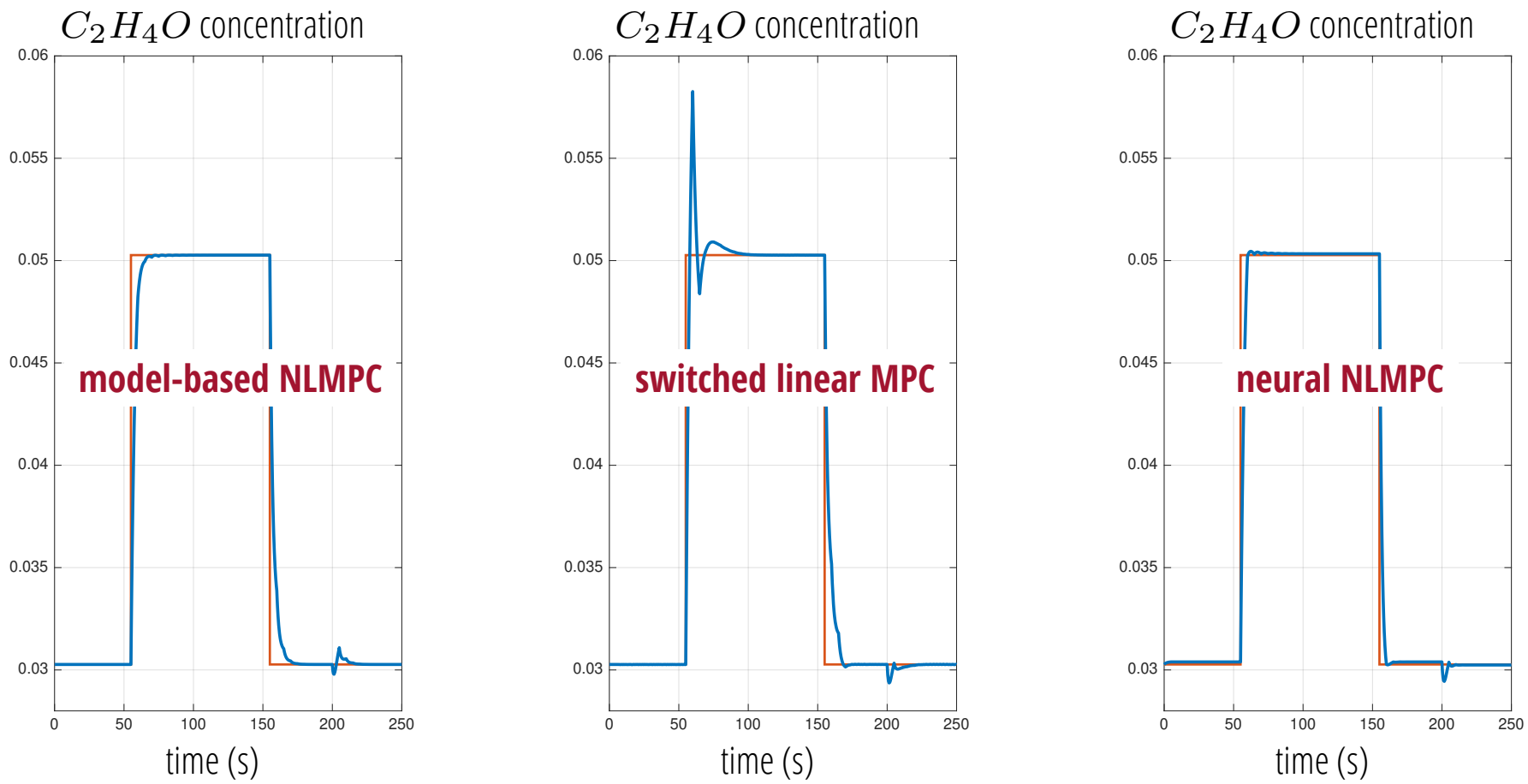
MPC OF ETHYLENE OXIDATION PLANT

- **MPC** settings:

sampling time	$T_s = 5$ s	measured disturbance @t=200
prediction horizon	$N = 10$	
control horizon	$N_u = 3$	
constraints	$0.0704 \leq u_1 \leq 0.7042$	
cost function	$\sum_{k=0}^{N-1}(y_{k+1} - r_{k+1})^2 + \frac{1}{100}(u_{1,k} - u_{1,k-1})^2$	

- We compare 3 different configurations:
  - NLMPC based on **physical model**
  - Switched linear MPC based on **3 linear models** obtained by linearizing the nonlinear model at $C_2H_4O = \{0.03,\ 0.04,\ 0.05\}$
  - NLMPC based on black-box **neural network** model

MPC OF ETHYLENE OXIDATION PLANT - CLOSED-LOOP RESULTS

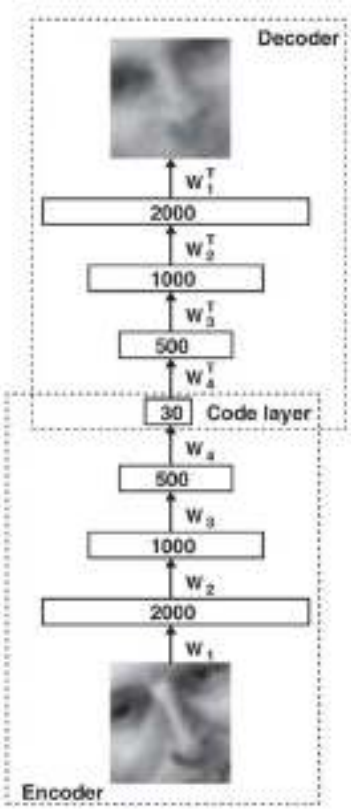


- Neural and model-based NLMPC have **similar** closed-loop performance
- Neural NLMPC requires **no physical model**

LEARNING NONLINEAR STATE-SPACE MODELS FOR MPC

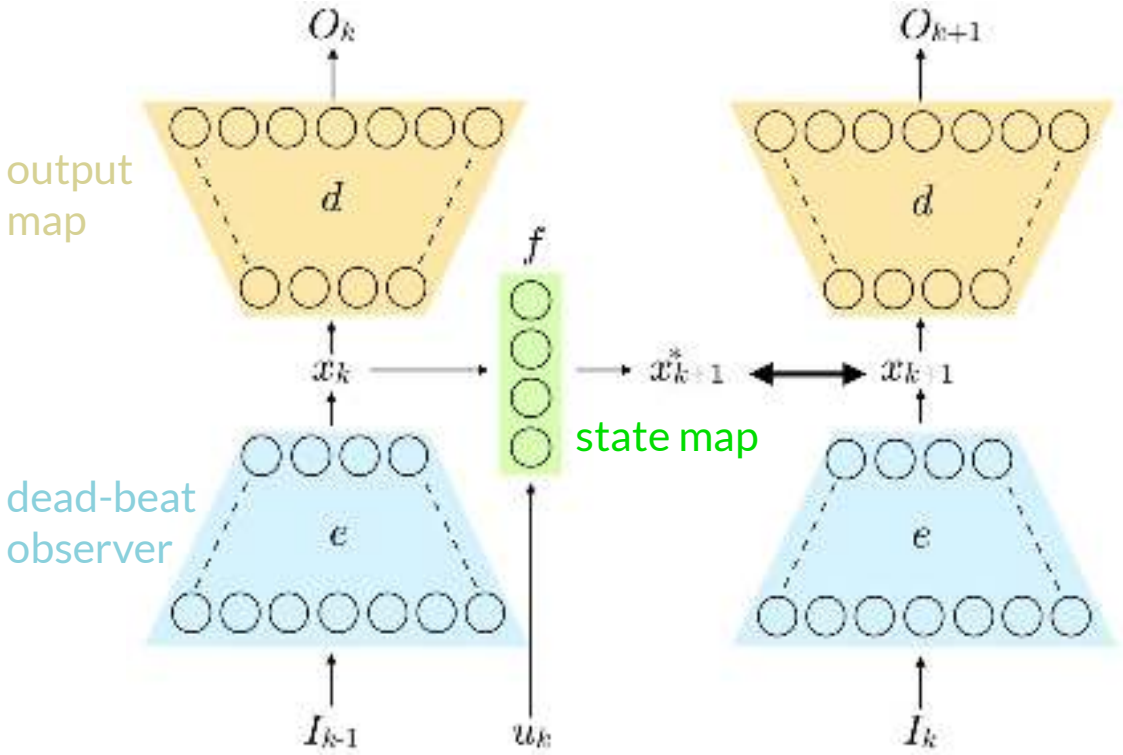
(Masti, Bemporad, 2018)

- **Idea**: use **autoencoders** and artificial neural networks to learn a **nonlinear state-space model** of **desired order** from input/output data



ANN with hourglass structure

(Hinton, Salakhutdinov, 2006)



$$O_k = [y'_k \ \dots \ y'_{k-m}]'$$

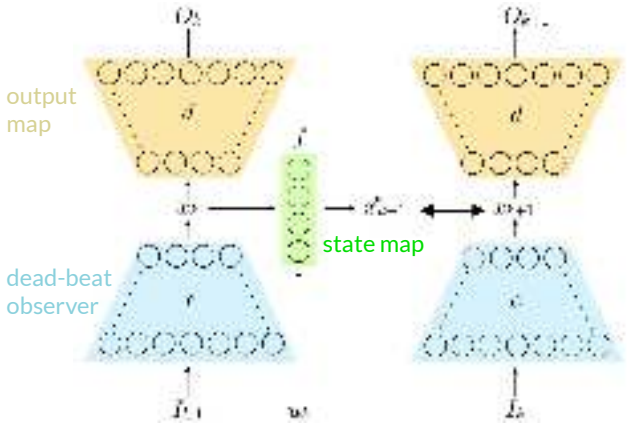
$$I_k = [y'_k \ \dots \ y'_{k-n_a+1} \ u'_k \ \dots \ u'_{k-n_b+1}]'$$

LEARNING NONLINEAR STATE-SPACE MODELS FOR MPC

- **Training problem**: choose n_a, n_b, n_x and solve

$$\min_{f,d,e} \sum_{k=k_0}^{N-1} \alpha \left(\ell_1(\hat{O}_k, O_k) + \ell_1(\hat{O}_{k+1}, O_{k+1})\right) + \beta\ell_2(x^\star_{k+1}, x_{k+1}) + \gamma\ell_3(O_{k+1}, O^\star_{k+1})$$

$$\text{s.t.} \quad \begin{array}{l} x_k = e(I_{k-1}),\ k = k_0, \ldots, N \\ x^\star_{k+1} = f(x_k, u_k),\ k = k_0, \ldots, N-1 \\ \hat{O}_k = d(x_k),\ O^\star_k = d(x^\star_k),\ k = k_0, \ldots, N \end{array}$$



- Model complexity reduction: add **group-LASSO** penalties on subsets of weights

- **Quasi-LPV** structure for MPC: set

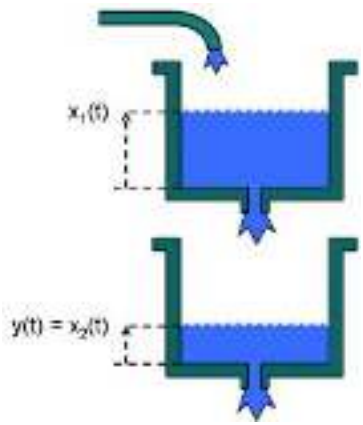
$$\begin{array}{rcl} f(x_k, u_k) & = & A(x_k, u_k)\left[\begin{smallmatrix} x_k \\ 1 \end{smallmatrix}\right] + B(x_k, u_k)u_k \\ y_k & = & C(x_k, u_k)\left[\begin{smallmatrix} x_k \\ 1 \end{smallmatrix}\right] \end{array}$$

(A_{ij}, B_{ij}, C_{ij} = feedforward NNs)

- Different options for the **state-observer**:
  - use encoder e to map past I/O into x_k (deadbeat observer)
  - design extended Kalman filter based on obtained model f, d
  - **simultaneously fit state observer** $\hat{x}_{k+1} = s(x_k, u_k, y_k)$ with loss $\ell_4(\hat{x}_{k+1}, x_{k+1})$

LEARNING NONLINEAR NEURAL STATE-SPACE MODELS FOR MPC

- **Example**: nonlinear two-tank benchmark problem

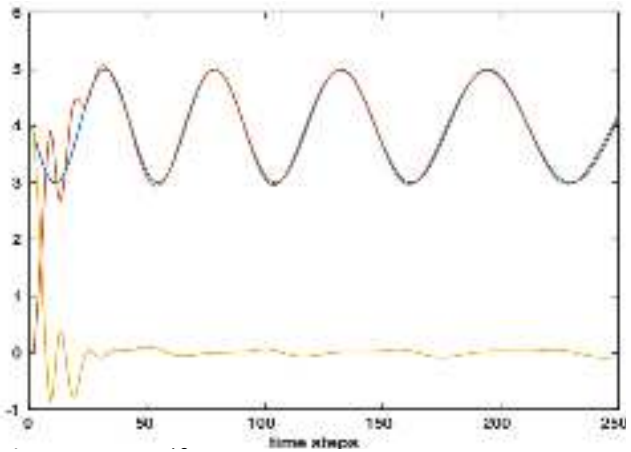


www.mathworks.com

$$\begin{cases} x_1(t+1) = x_1(t) - k_1\sqrt{x_1(t)} + k_2 u(t) \\ x_2(t+1) = x_2(t) + k_3\sqrt{x_1(t)} - k_4\sqrt{x_2(t)} \\ y(t) = x_2(t) + u(t) \end{cases}$$

Model is totally unknown to learning algorithm

- Artificial neural network (ANN): 3 hidden layers 60 exponential linear unit (ELU) neurons
- For given number of model parameters, **autoencoder approach is superior to NNARX**
- **Jacobians** directly obtained from ANN structure for Kalman filtering & MPC problem construction



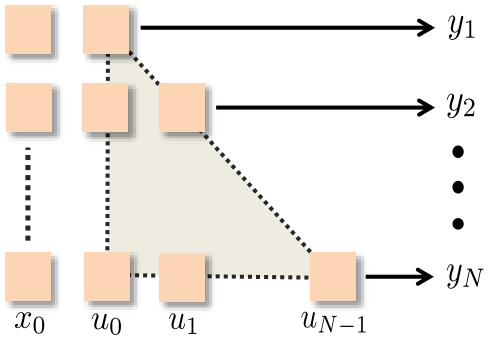
LTV-MPC results

LEARNING AFFINE NEURAL PREDICTORS FOR MPC

(Masti, Smarra, D'Innocenzo, Bemporad, 2020)

- Alternative: **learn the entire prediction**

$$y_k = h_k(x_0, u_0, \ldots, u_{k-1}),\ k = 1, \ldots, N$$



- **LTV-MPC formulation**: linearize h_k around nominal inputs $\bar{u}_j$

$$y_k = h_k(x_0, \bar{u}_0, \ldots, \bar{u}_{k-1}) + \sum_{j=0}^{k-1} \frac{\partial h_k}{\partial u_j}(x_0, \bar{u}_0, \ldots, \bar{u}_{k-1})(u_j - \bar{u}_j)$$

Example: $\bar{u}_k$ = MPC sequence optimized @$k-1$

- Avoid computing Jacobians by fitting h_k in the affine form

$$y_k = f_k(x_0, \bar{u}_0, \ldots, \bar{u}_{k-1}) + g_k(x_0, \bar{u}_0, \ldots, \bar{u}_{k-1}) \begin{bmatrix} u_0 - \bar{u}_0 \\ \vdots \\ u_{k-1} - \bar{u}_{k-1} \end{bmatrix}$$

cf. (Liu, Kadirkamanathan, 1998)

LEARNING AFFINE NEURAL PREDICTORS FOR MPC

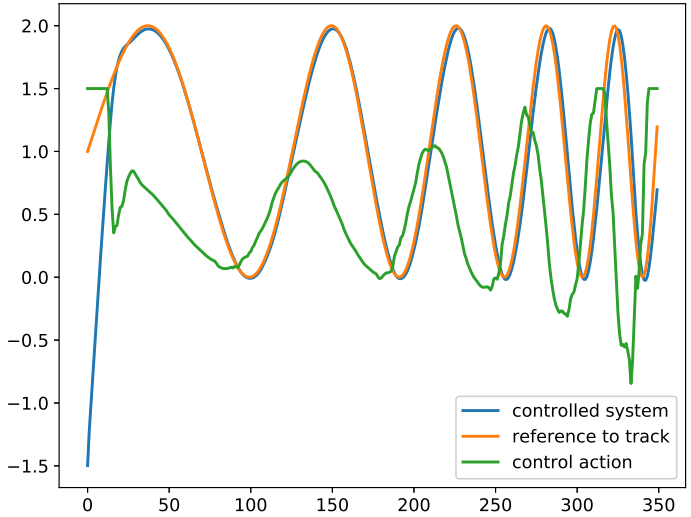
- **Example**: apply **affine neural predictor** to nonlinear two-tank benchmark problem

10000 training samples, ANN with **2** layers of **20 ReLU neurons**

$$e_{\mathrm{FIT}} = \max\left\{0, 1 - \frac{\|\hat{y} - y\|_2}{\|y - \bar{y}\|_2}\right\}$$

Prediction step	e_{FIT}
1	0.959
2	0.958
4	0.948
7	0.915
10	0.858

- Closed-loop LTV-MPC results:



- Model complexity reduction:
  add **group-LASSO** term with penalty λ

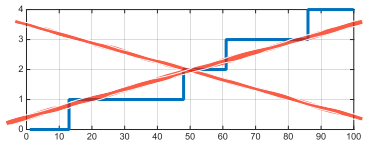
λ	e_{FIT} (average on all prediction steps)	# nonzero weights
.01	0.853	328
0.005	0.868	363
0.001	0.901	556
0.0005	0.911	888
0	0.917	9000

ON THE USE OF NEURAL NETWORKS FOR MPC

- Neural prediction models can **speed up** the MPC design a lot

- Experimental **data** need to well cover the operating range (as in linear system identification)

- No need to define linear operating ranges with NN's, it is a **one-shot model-learning** step



- Physical models may **better predict** unseen situations than black box models

- Physical modeling can help driving the choice of the **nonlinear model structure** to use (**gray-box** models)

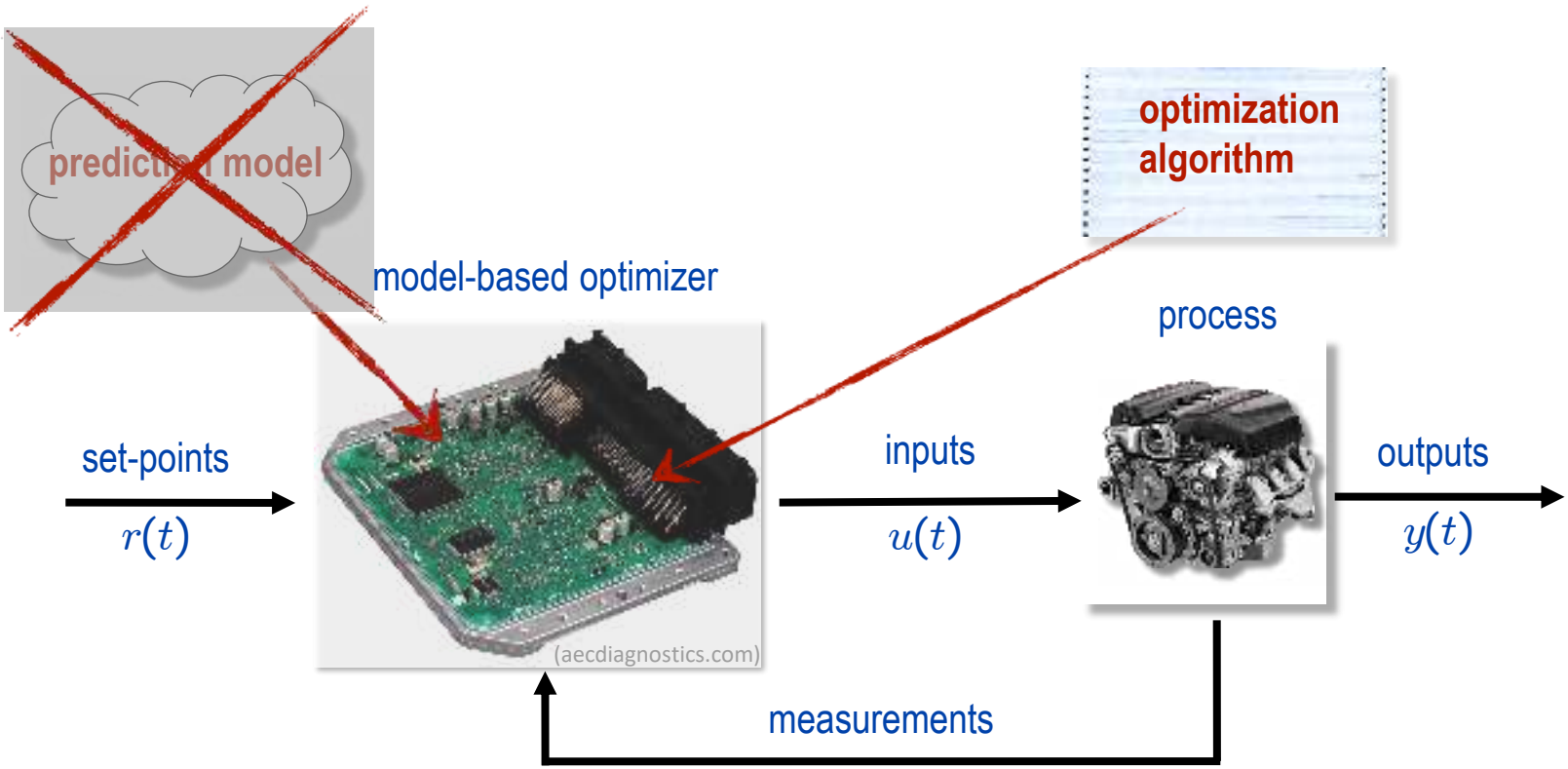
- NN model can be updated on-line for **adaptive nonlinear MPC**

ON THE USE OF NEURAL NETWORKS FOR MPC

- **MPC** + **ML** together can have a tremendous impact in the design and implementation of advanced process control systems:
  - **MPC** and on-line (quadratic or nonlinear) optimization is an extremely powerful advanced process control methodology
  - **ML** extremely useful to get **control-oriented nonlinear models** directly from **data**
- **Neural nonlinear MPC** requires very **advanced technical software** to run efficiently and reliably (**model learning**, **problem construction**, **optimization**)

DIRECT DATA-DRIVEN MPC

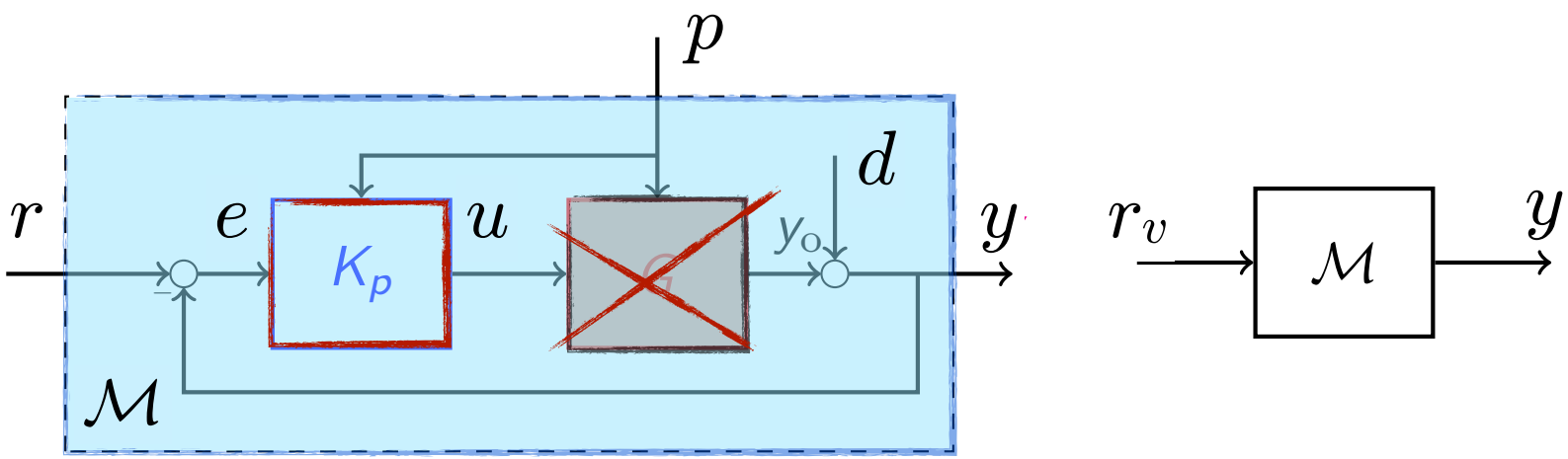
DIRECT DATA-DRIVEN MPC



- Can we design an MPC controller **without** first identifying a model of the **open-loop process** ?

DATA-DRIVEN DIRECT CONTROLLER SYNTHESIS

(Campi, Lecchini, Savaresi, 2002) (Formentin et al., 2015)

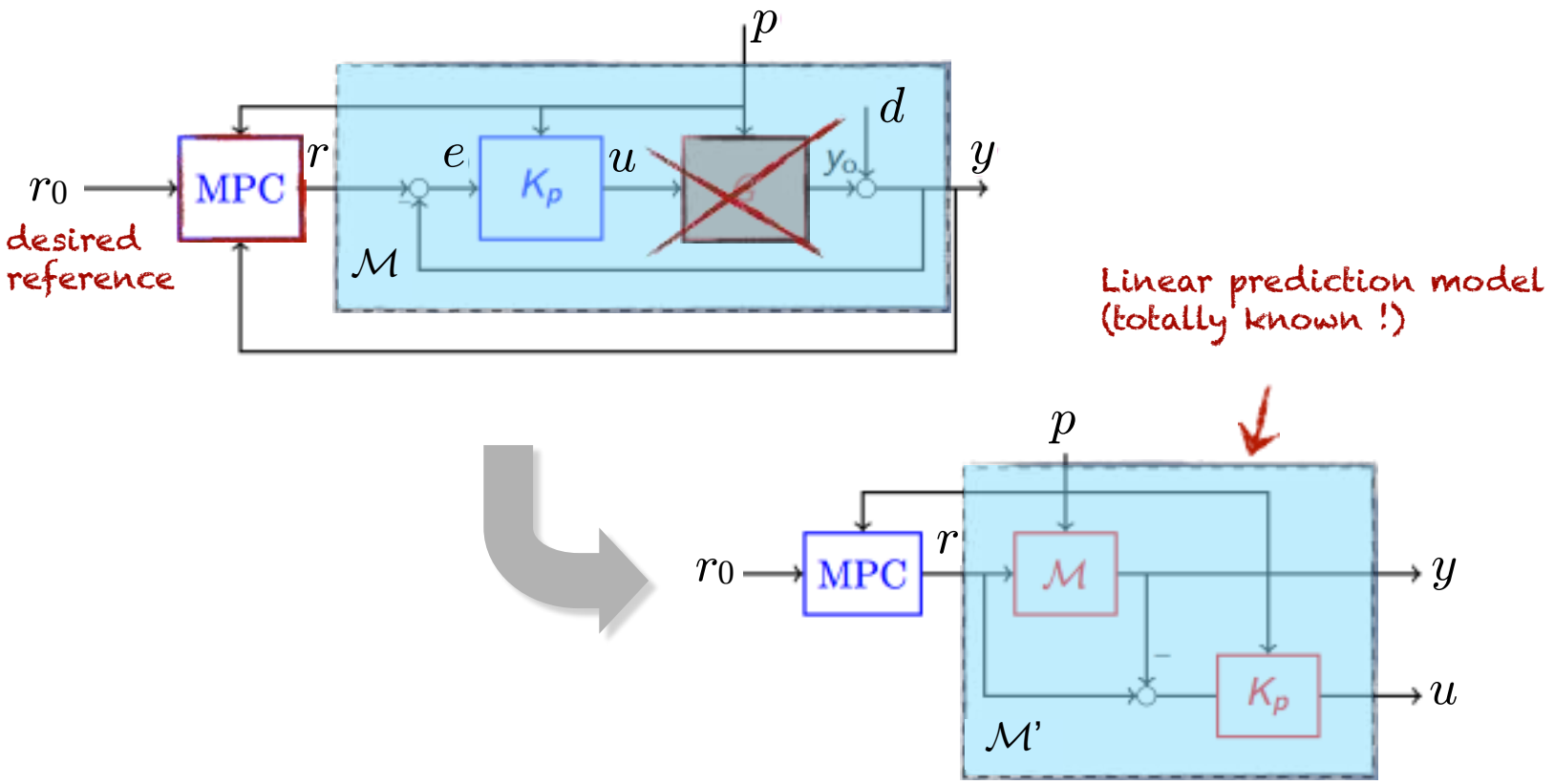


- Collect a set of **data** $\{u(t), y(t), p(t)\}, t = 1, \ldots, N$
- Specify a **desired closed-loop linear model** $\mathcal{M}$ from r to y
- Compute $r_v(t) = \mathcal{M}^{\#} y(t)$ from **pseudo-inverse model** $\mathcal{M}^{\#}$ of $\mathcal{M}$
- **Identify** linear (LPV) model K_p from $e_v = r_v - y$ (virtual tracking error) to u

DIRECT DATA-DRIVEN MPC

- Design a linear MPC (**reference governor**) to generate the reference r

(Bemporad, Mosca, 1994) (Gilbert, Kolmanovsky, Tan, 1994)



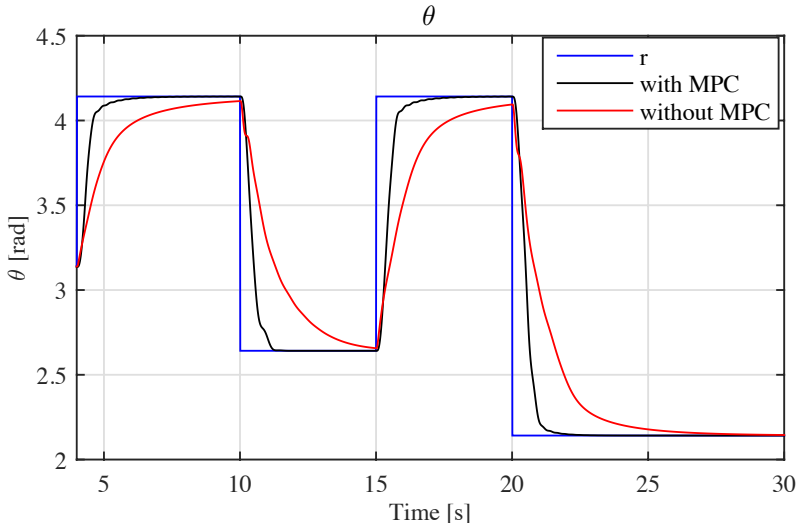
- MPC designed to handle input/output **constraints** and improve **performance**

(Piga, Formentin, Bemporad, 2017)

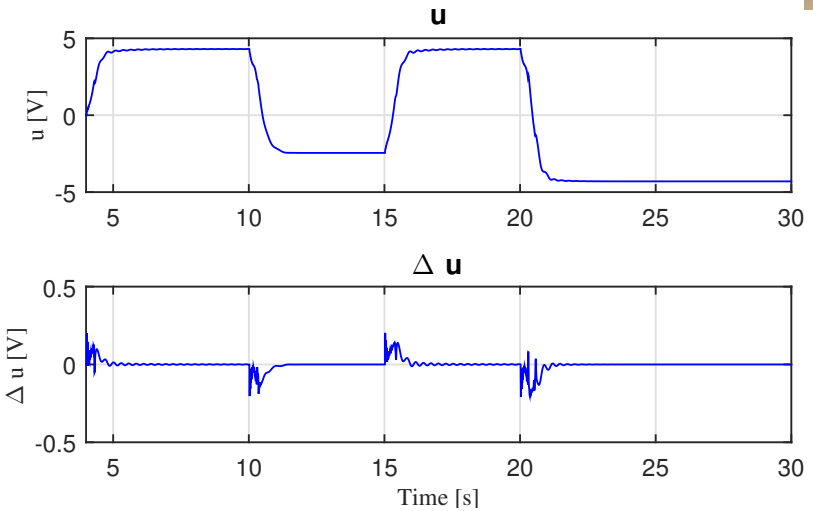
DIRECT DATA-DRIVEN MPC - AN EXAMPLE

- Experimental results: MPC handles soft constraints on u, Δu and y

  (motor equipment by courtesy of TU Delft)



desired tracking performance achieved

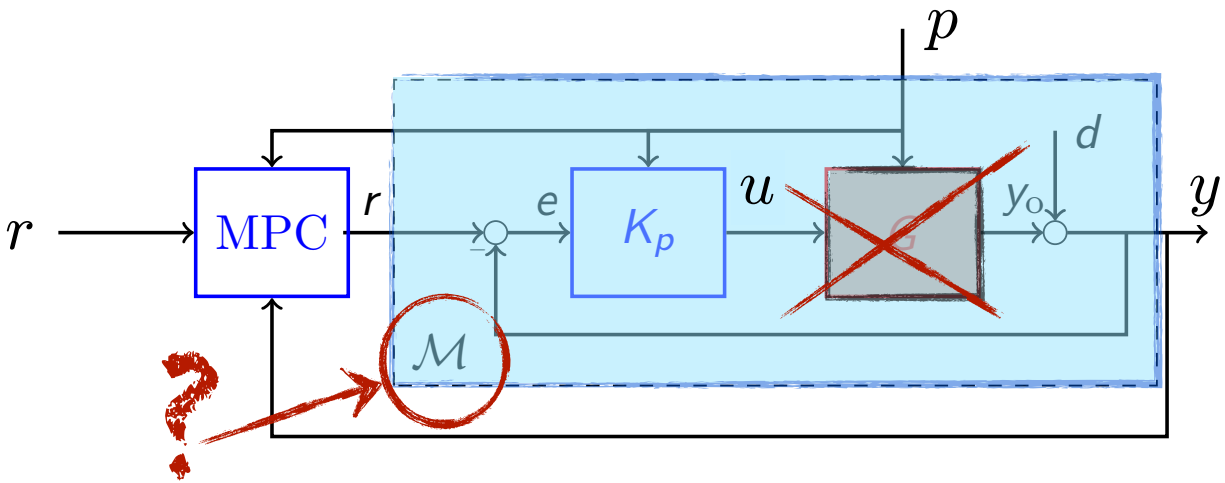


constraints on input increments satisfied

No open-loop process model is identified to design the MPC controller!

OPTIMAL DIRECT DATA-DRIVEN MPC

- **Question**: How to choose the reference model $\mathcal{M}$?



- Can we choose $\mathcal{M}$ from data so that K_p is an **optimal controller** ?

OPTIMAL DIRECT DATA-DRIVEN MPC

(Selvi, Piga, Bemporad, 2018)

- **Idea**: parameterize desired closed-loop model $\mathcal{M}(\theta)$ and optimize

$$\min_{\theta} J(\theta) = \frac{1}{N}\sum_{t=0}^{N-1} \underbrace{W_y(r(t)-y_p(\theta,t))^2 + W_{\Delta u}\Delta u_p^2(\theta,t)}_{\text{performance index}} + \underbrace{W_{\text{fit}}(u(t)-u_v(\theta,t))^2}_{\text{identification error}}$$

- Evaluating $J(\theta)$ requires synthesizing $K_p(\theta)$ from data and simulating the nominal model and control law

$$y_p(\theta,t) = \mathcal{M}(\theta)r(t) \qquad u_p(\theta,t) = K_p(\theta)(r(t)-y_p(\theta,t))$$

$$\Delta u_p(\theta,t) = u_p(\theta,t) - u_p(\theta,t-1)$$

- Optimal θ obtained by solving a **(non-convex) nonlinear programming** problem

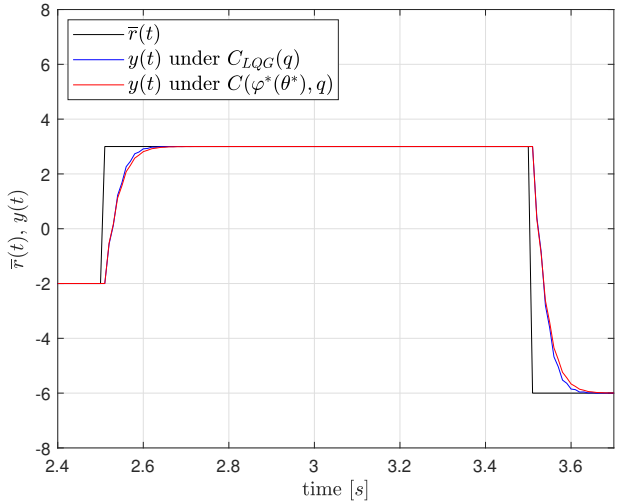
OPTIMAL DIRECT DATA-DRIVEN MPC

(Selvi, Piga, Bemporad, 2018)

- **Results**: **linear** process

$$G(z) = \frac{z - 0.4}{z^2 + 0.15z - 0.325}$$

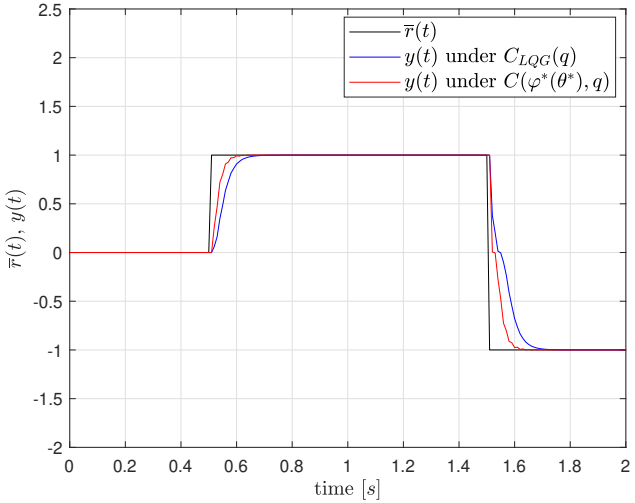
Data-driven controller **only 1.3% worse** than model-based LQR (=SYS-ID on same data + LQR design)



- **Results**: **nonlinear (Wiener)** process

$$\begin{aligned} y_L(t) &= G(z)u(t) \\ y(t) &= |y_L(t)| \arctan(y_L(t)) \end{aligned}$$

The data-driven controller is **24% better** than LQR based on identified open-loop model !



DATA-DRIVEN OPTIMAL POLICY SEARCH

DATA-DRIVEN OPTIMAL POLICY SEARCH

(Ferrarotti, Bemporad, 2019)

- Plant + environment dynamics (**unknown**):

$$s_{t+1} = h(s_t, p_t, u_t, d_t)$$

  - s_t states of plant & environment
  - p_t exogenous signal (e.g., reference)
  - u_t control input
  - d_t unmeasured disturbances

- **Control policy**: $\pi : \mathbb{R}^{n_s+n_p} \longrightarrow \mathbb{R}^{n_u}$ deterministic control policy

$$u_t = \pi(s_t, p_t)$$

- Closed-loop **performance** of an execution is defined as

$$\mathcal{J}_\infty(\pi, s_0, \{p_\ell, d_\ell\}_{\ell=0}^{\infty}) = \sum_{\ell=0}^{\infty} \rho(s_\ell, p_\ell, \pi(s_\ell, p_\ell))$$

$$\rho(s_\ell, p_\ell, \pi(s_\ell, p_\ell)) \;=\; \text{stage cost}$$

OPTIMAL POLICY SEARCH PROBLEM

- **Optimal policy:**

$$\begin{aligned} \pi^* &= \arg\min_\pi \mathcal{J}(\pi) \\ \mathcal{J}(\pi) &= \mathbb{E}_{s_0,\{p_\ell,d_\ell\}}\left[\mathcal{J}_\infty(\pi, s_0, \{p_\ell, d_\ell\})\right] \end{aligned}$$

expected performance

- **Simplifications**:
  - Finite parameterization: $\pi = \pi_K(s_t, p_t)$ with K = parameters to optimize
  - Finite horizon: $\mathcal{J}_L(\pi, s_0, \{p_\ell, d_\ell\}_{\ell=0}^{L-1}) = \sum_{\ell=0}^{L-1} \rho(s_\ell, p_\ell, \pi(s_\ell, p_\ell))$

- Optimal policy search: use **stochastic gradient descent (SGD)**

$$K_t \leftarrow K_{t-1} - \alpha_t \mathcal{D}(K_{t-1})$$

with $\mathcal{D}(K_{t-1})$ = descent direction

DESCENT DIRECTION

- The descent direction $\mathcal{D}(K_{t-1})$ is computed by generating:
  - N_s perturbations $s_0^{(i)}$ around the current state s_t
  - N_r random reference signals $r_\ell^{(j)}$ of length L,
  - N_d random disturbance signals $d_\ell^{(h)}$ of length L,

$$\mathcal{D}(K_{t-1}) = \sum_{i=1}^{N_s}\sum_{j=1}^{N_p}\sum_{k=1}^{N_q} \nabla_K \mathcal{J}_L(\pi_{K_{t-1}}, s_0^{(i)}, \{r_\ell^{(j)}, d_\ell^{(k)}\})$$

SGD step = mini-batch of size $M = N_s \cdot N_r \cdot N_d$

- Computing $\nabla_K \mathcal{J}_L$ requires predicting the effect of π over L future steps

- We use a **local linear model** just for computing $\nabla_K \mathcal{J}_L$, obtained by running **recursive linear system identification**

OPTIMAL POLICY SEARCH ALGORITHM

- At each step t:
  1. Acquire current s_t
  2. Recursively update the local linear model
  3. Estimate the direction of descent $\mathcal{D}(K_{t-1})$
  4. Update policy: $K_t \leftarrow K_{t-1} - \alpha_t \mathcal{D}(K_{t-1})$

- If policy is **learned online** and needs to be applied to the process:
  - Compute the nearest policy $K_t^\star$ to K_t that stabilizes the local model

$$K_t^\star = \begin{array}{l} \text{argmin}_K \|K - K_t^s\|_2^2 \\ \text{s.t. } K \text{ stabilizes local linear model} \end{array}$$

linear matrix inequality

- When policy is learned online, **exploration** is guaranteed by the reference r_t

SPECIAL CASE: OUTPUT TRACKING

- $x_t = [\, y_t,\, y_{t-1},\, \ldots, y_{t-n_o},\, u_{t-1},\, u_{t-2},\, \ldots, u_{t-n_i}\,]$

  $\Delta u_t = u_t - u_{t-1}$ control input increment

- Stage cost: $\| \, y_{t+1} - r_t \, \|^2_{Qy} \; + \; \| \, \Delta u_t \, \|^2_R \; + \; \| \, q_{t+1} \, \|^2_{Q_q}$

- Integral action dynamics $q_{t+1} = q_t + (y_{t+1} - r_t)$

$$\Rightarrow \quad s_t = \begin{bmatrix} x_t \\ q_t \end{bmatrix}, \quad p_t = r_t.$$

- **Linear policy parametrization**:

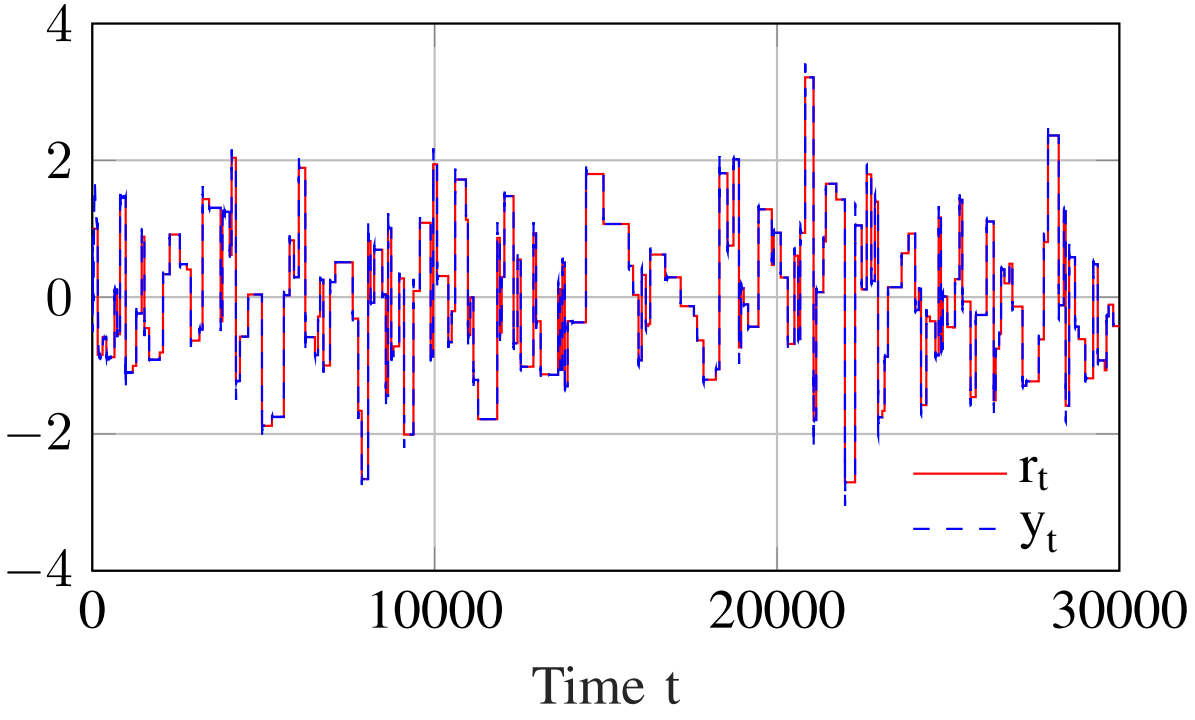
$$\pi_K(s_t,\, r_t) = -K^s \cdot s_t - K^r \cdot r_t, \qquad K = \begin{bmatrix} K^s \\ K^r \end{bmatrix}$$

EXAMPLE: RETRIEVE LQR FROM DATA

$$\begin{cases} x_{t+1} = \begin{bmatrix} -0.669 & 0.378 & 0.233 \\ -0.288 & -0.147 & -0.638 \\ -0.337 & 0.589 & 0.043 \end{bmatrix} x_t + \begin{bmatrix} -0.295 \\ -0.325 \\ -0.258 \end{bmatrix} u_t \\ y_t = \begin{bmatrix} -1.139 & 0.319 & -0.571 \end{bmatrix} x_t \end{cases}$$

model is unknown

Online tracking performance (no disturbance, $d_t = 0$):



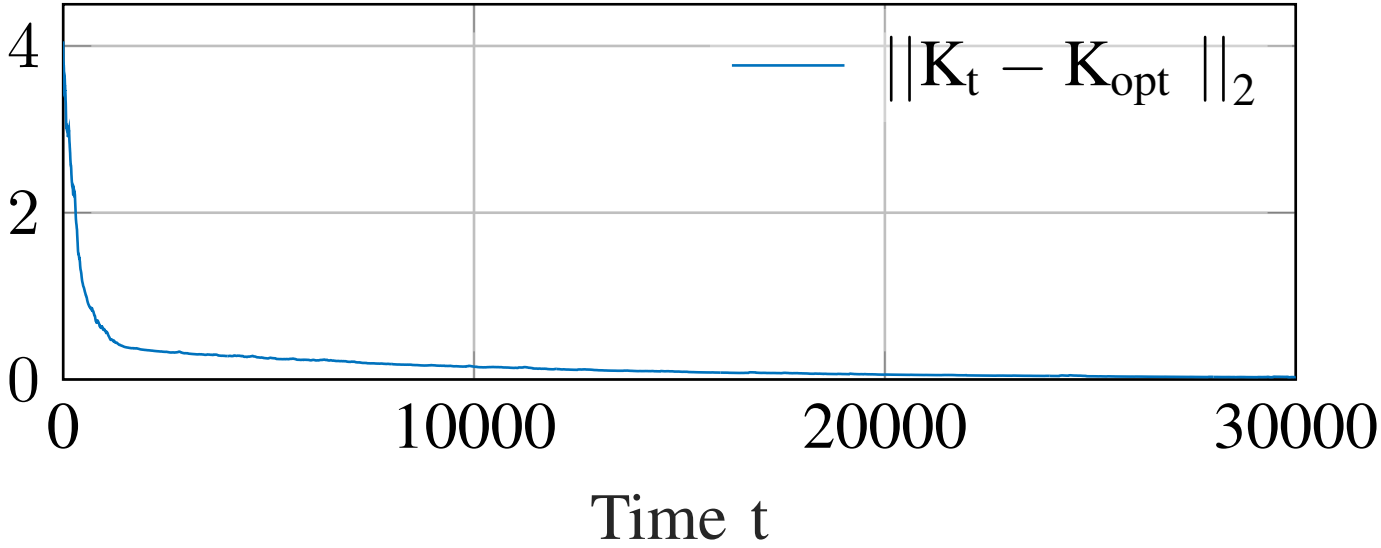
$Q_y = 1$
$R = 0.1$
$Q_q = 1$

n_i	n_o	L
3	3	20

N_0	N_r	N_q
50	1	10

EXAMPLE: RETRIEVE LQR FROM DATA

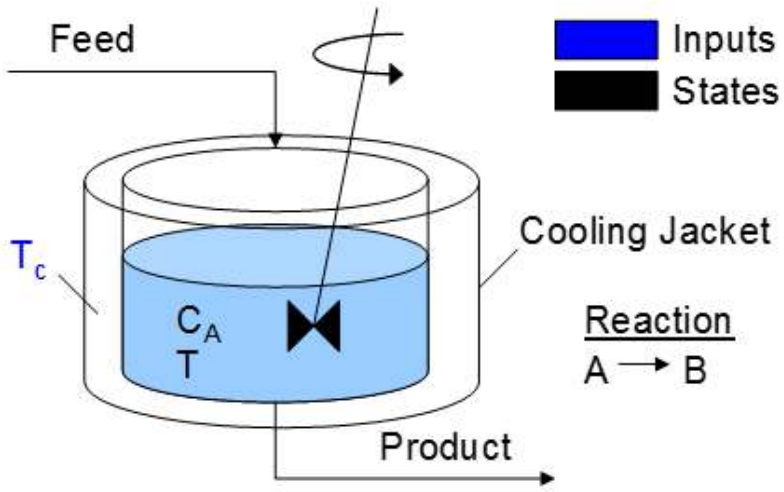
Evolution of the error $\|K_t - K_{opt}\|_2$:



$$K_{\text{SGD}} = [-1.255, 0.218, 0.652, 0.895, 0.050, 1.115, -2.186]$$

$$K_{\text{opt}} = [-1.257, 0.219, 0.653, 0.898, 0.050, 1.141, -2.196]$$

NONLINEAR EXAMPLE



Continuously Stirred Tank Reactor (CSTR)

apmonitor.com

model is unknown

Feed:

- concentration: 10kg mol/m^3
- temperature: 298.15K

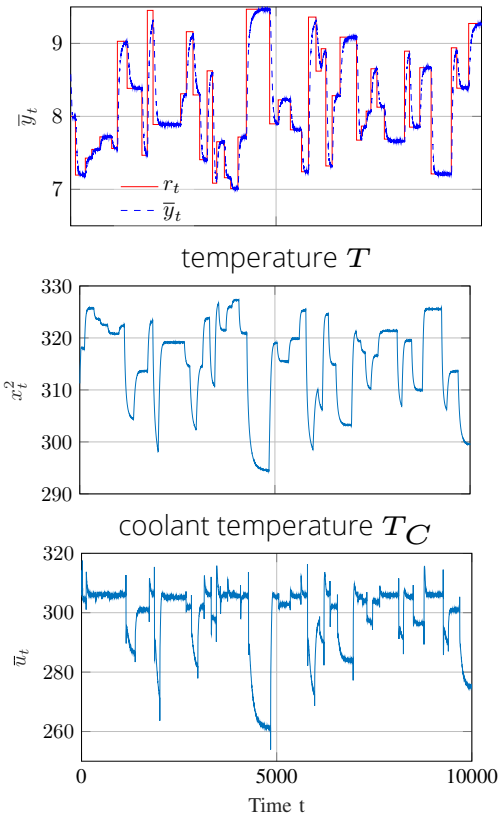
$$T = \hat{T} + \eta_T,\ C_A = \hat{C}_A + \eta_C, \quad \eta_T,\ \eta_C \sim \mathcal{N}(0, \sigma^2), \quad \sigma = 0.01$$

$$\boxed{Q_y = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \qquad R = 0.1 \qquad Q_q = \begin{bmatrix} 0.01 & 0 \\ 0 & 0 \end{bmatrix}}$$

NONLINEAR EXAMPLE

Online learning

concentration C_A and reference r_t

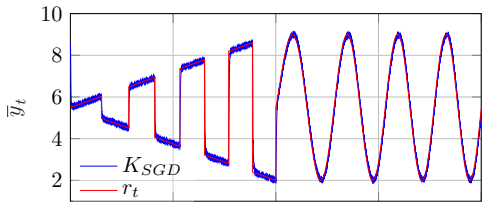


n_i	n_o	L
2	3	10

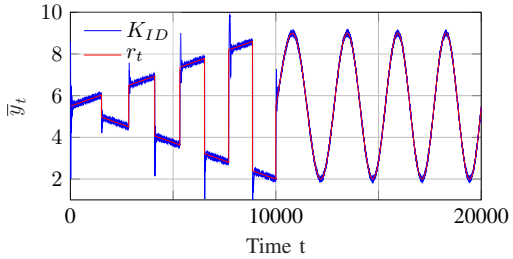
N_0	N_r	N_q
50	20	20

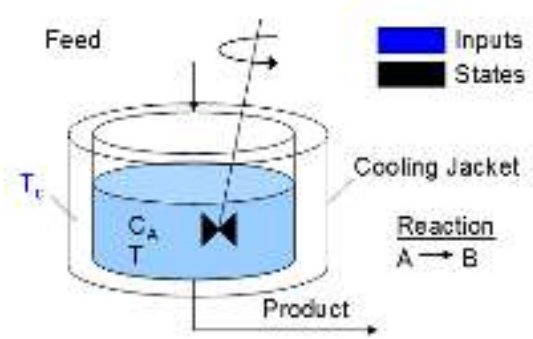
Validation phase

Cost of $\mathbf{K}_{\mathrm{SGD}} = \mathbf{4.3 \cdot 10^3}$



Cost of $\mathbf{K}_{\mathrm{ID}} = \mathbf{2.4 \cdot 10^4}$





Continuously Stirred Tank Reactor (CSTR)

(courtesy: apmonitor.com)

SGD beats SYS-ID + LQR

- Extended to **switching-linear** and **nonlinear** policy, and to **collaborative learning**

(Ferrarotti, Bemporad, 2020a) (Ferrarotti, Bemporad, 2020b) (Ferrarotti, Breschi, Bemporad, 2021)

LEARNING OPTIMAL MPC CALIBRATION

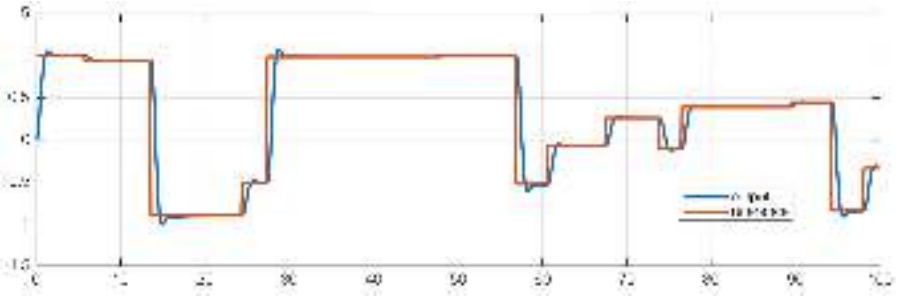
MPC CALIBRATION PROBLEM

- The design depends on a vector x of **MPC parameters**

- Parameters can be many things:
  - MPC weights, prediction model coefficients, horizons
  - Covariance matrices used in Kalman filters
  - Tolerances used in numerical solvers
  - ...
- Define a **performance index** f over a closed-loop simulation or real experiment. For example:

$$f(x) = \sum_{t=0}^{T} \|y(t) - r(t)\|^2$$

(tracking quality)



- **Auto-tuning** = find the best combination of parameters by solving the **global optimization problem**

$$\min_x f(x)$$

GLOBAL OPTIMIZATION ALGORITHMS FOR AUTO-TUNING

What is a good optimization algorithm to solve $\min f(x)$?

- The algorithm should not require the gradient $\nabla f(x)$ of $f(x)$, in particular if experiments are involved (**derivative-free** or **black-box optimization**)

- The algorithm should not get stuck on local minima (**global optimization**)

- The algorithm should make the **fewest evaluations** of the cost function f (which is expensive to evaluate)

AUTO-TUNING - GLOBAL OPTIMIZATION ALGORITHMS

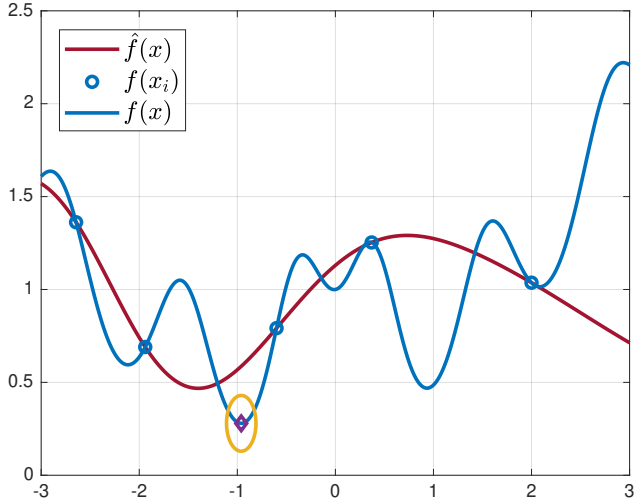
- Several derivative-free global optimization algorithms exist: (Rios, Sahidinis, 2013)
  - Lipschitzian-based partitioning techniques:
    - **DIRECT** (DIvide in RECTangles) (Jones, 2001)
    - Multilevel Coordinate Search (**MCS**) (Huyer, Neumaier, 1999)
  - Response surface methods
    - **Kriging** (Matheron, 1967), **DACE** (Sacks et al., 1989)
    - Efficient global optimization (**EGO**) (Jones, Schonlau, Welch, 1998)
    - **Bayesian optimization** (Brochu, Cora, De Freitas, 2010)
  - Genetic algorithms (**GA**) (Holland, 1975)
  - Particle swarm optimization (**PSO**) (Kennedy, 2010)
  - ...
- **New method**: **radial basis function** surrogates + **inverse distance weighting** (**GLIS**) (Bemporad, 2020)

```
cse.lab.imtlucca.it/~bemporad/glis
```

AUTO-TUNING - GLIS

- **Goal**: solve the **global optimization** problem

$$\begin{array}{rl} \min_x & f(x) \\ \text{s.t.} & \ell \leq x \leq u \\ & g(x) \leq 0 \end{array}$$



- **Step #0**: Get random initial samples $x_1, \ldots, x_{N_{\text{init}}}$ **(Latin Hypercube Sampling)**

- **Step #1**: given N samples of f at $x_1, \ldots, x_N$, build the **surrogate function**

$$\hat{f}(x) = \sum_{i=1}^{N} \beta_i \phi(\epsilon \|x - x_i\|_2)$$

$\phi =$ radial basis function

Example: $\phi(\epsilon d) = \frac{1}{1+(\epsilon d)^2}$ *(inverse quadratic)*

Vector β solves $\hat{f}(x_i) = f(x_i)$ for all $i = 1, \ldots, N$ (=linear system)

- **CAVEAT**: build and minimize $\hat{f}(x_i)$ iteratively may easily miss global optimum!

AUTO-TUNING - GLIS

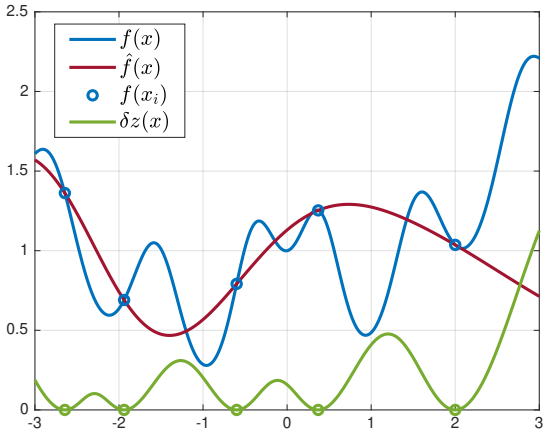
- **Step #2**: construct the **IDW exploration function**

$$z(x) \quad = \quad \frac{2}{\pi}\Delta F \tan^{-1}\left(\frac{1}{\sum_{i=1}^{N} w_i(x)}\right)$$

or 0 if $x \in \{x_1, \ldots, x_N\}$

where $w_i(x) = \dfrac{e^{-\|x-x_i\|^2}}{\|x-x_i\|^2}$

ΔF = observed range of $f(x_i)$



δ= *exploitation* vs *exploration* tradeoff

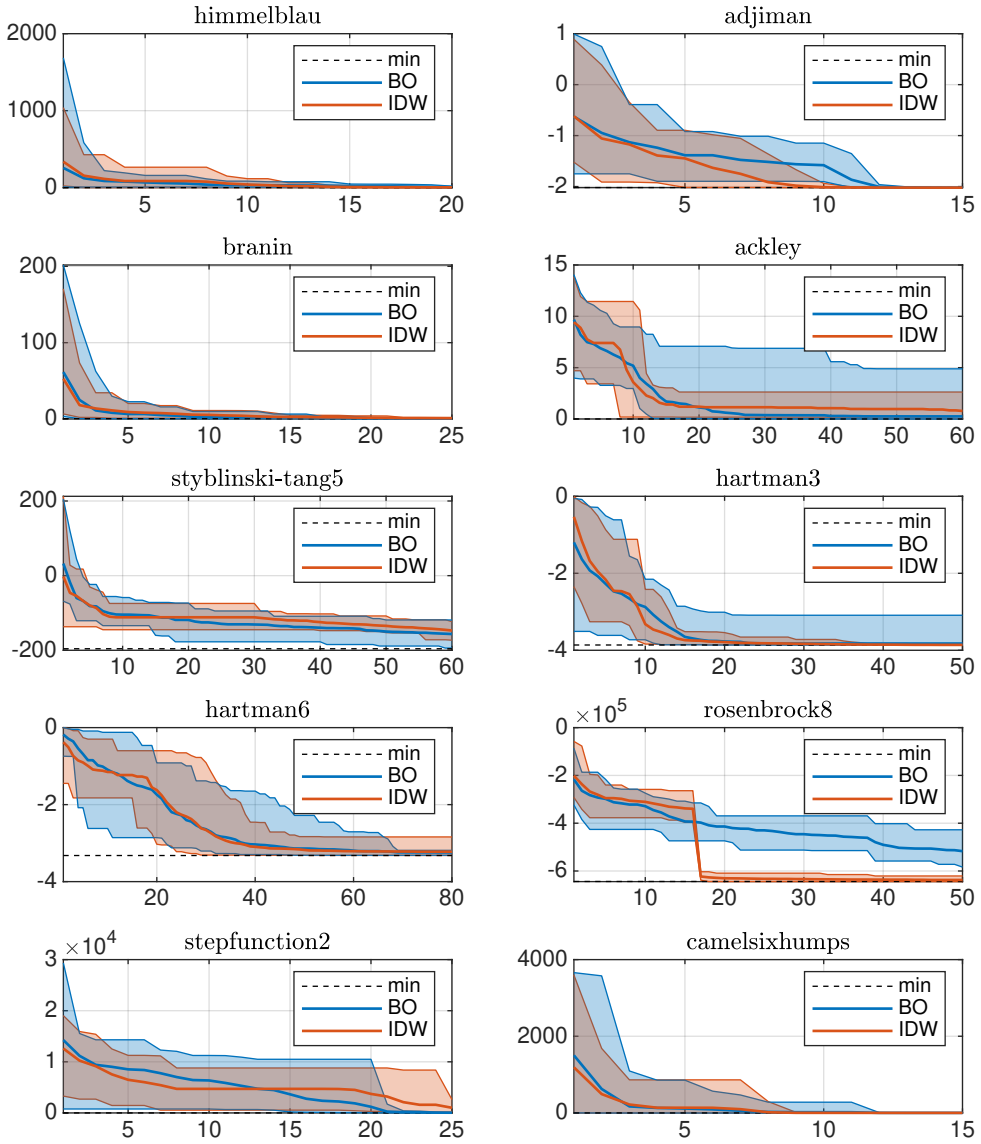
- **Step #3**: optimize the **acquisition function**

$$x_{N+1} = \quad \arg\min \quad \hat{f}(x) - \delta z(x)$$
$$\text{s.t.} \quad \ell \leq x \leq u,\ g(x) \leq 0$$

to get new sample x_{N+1}

- Iterate the procedure to get new samples $x_{N+2}, \ldots, x_{N_{\max}}$

GLIS VS BAYESIAN OPTIMIZATION



problem	n	BO[s]	IDW [s]
ackley	2	26.42	3.24
adjiman	2	3.39	0.66
branin	2	9.58	1.27
camelsixhumps	2	4.49	0.62
hartman3	3	23.19	3.58
hartman6	6	52.73	10.08
himmelblau	2	7.15	0.92
rosenbrock8	8	58.31	11.45
stepfunction2	4	10.52	1.72
styblinski-tang5	5	33.30	5.80

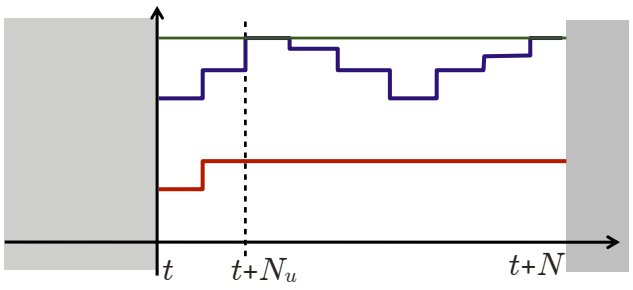
Results computed on 20 runs per test

BO = MATLAB's **bayesopt** fcn

AUTO-TUNING: MPC EXAMPLE

- We want to auto-tune the linear MPC controller

$$\begin{aligned} \min \quad & \sum_{k=0}^{50-1} (y_{k+1} - r(t))^2 + (W^{\Delta u}(u_k - u_{k-1}))^2 \\ \text{s.t.} \quad & x_{k+1} = Ax_k + Bu_k \\ & y_c = Cx_k \\ & -1.5 \leq u_k \leq 1.5 \\ & u_k \equiv u_{N_u}, \forall k = N_u, \ldots, N-1 \end{aligned}$$



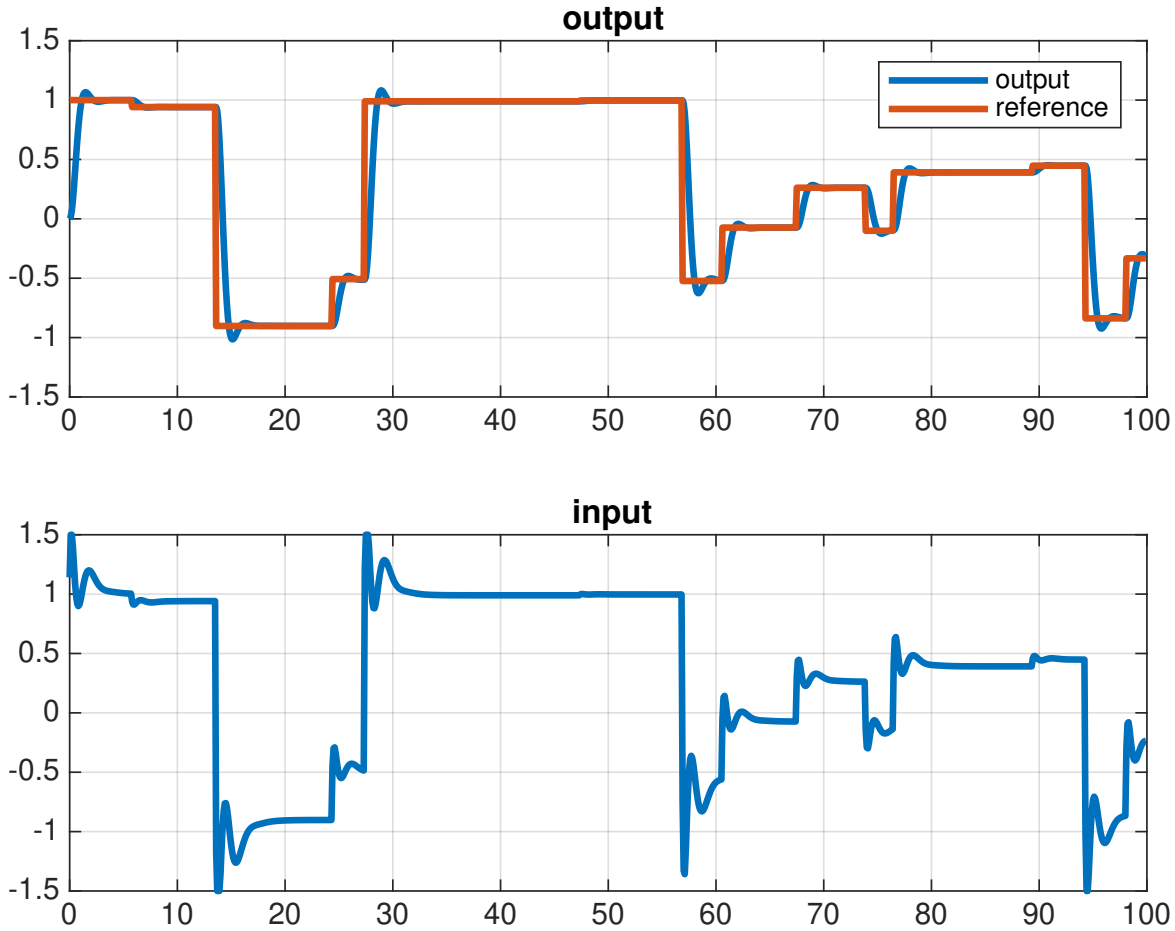
- Calibration parameters: $x = [\log_{10} W^{\Delta u}, N_u]$

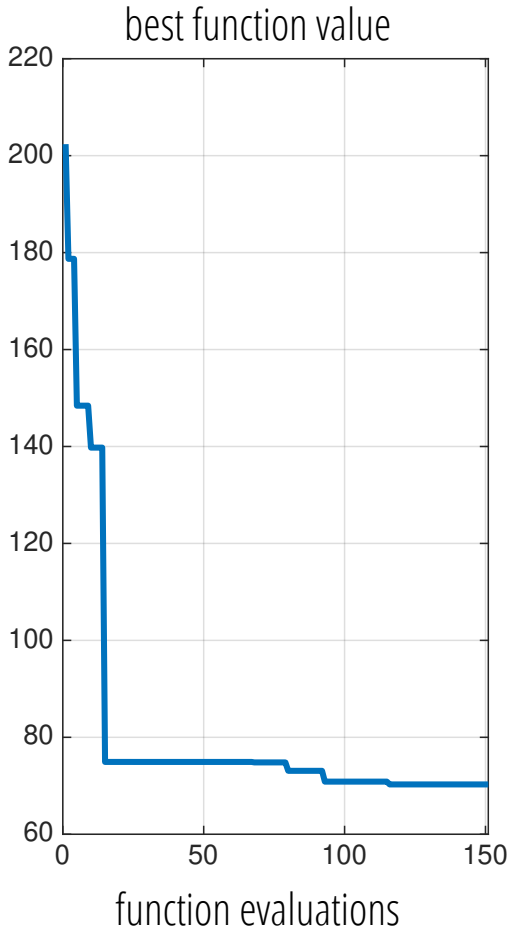
- Range: $-5 \leq x_1 \leq 3$ and $1 \leq x_2 \leq 50$

- Closed-loop performance objective:

$$f(x) = \sum_{t=0}^{T} \underbrace{(y(t) - r(t))^2}_{\text{track well}} + \underbrace{\frac{1}{2}(u(t) - u(t-1))^2}_{\text{smooth control action}} + \underbrace{2N_u}_{\text{small QP}}$$

AUTO-TUNING: EXAMPLE



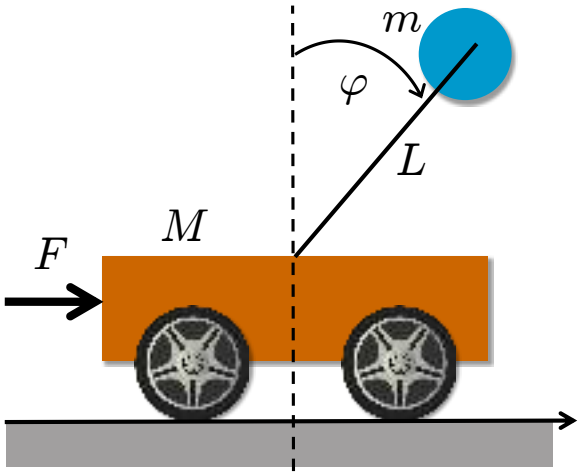


- Result: $x^\star = [-0.2341, 2.3007]$ $\Rightarrow$ $W^{\Delta u} = 0.5833, N_u = 2$

MPC AUTOTUNING EXAMPLE

(Forgione, Piga, Bemporad, 2020)

- Linear MPC applied to cart-pole system: **14 parameters** to tune



  - **sample time**
  - **weights** on outputs and input increments
  - prediction and control **horizons**
  - **covariance** matrices of Kalman filter
  - absolute and relative **tolerances** of QP solver

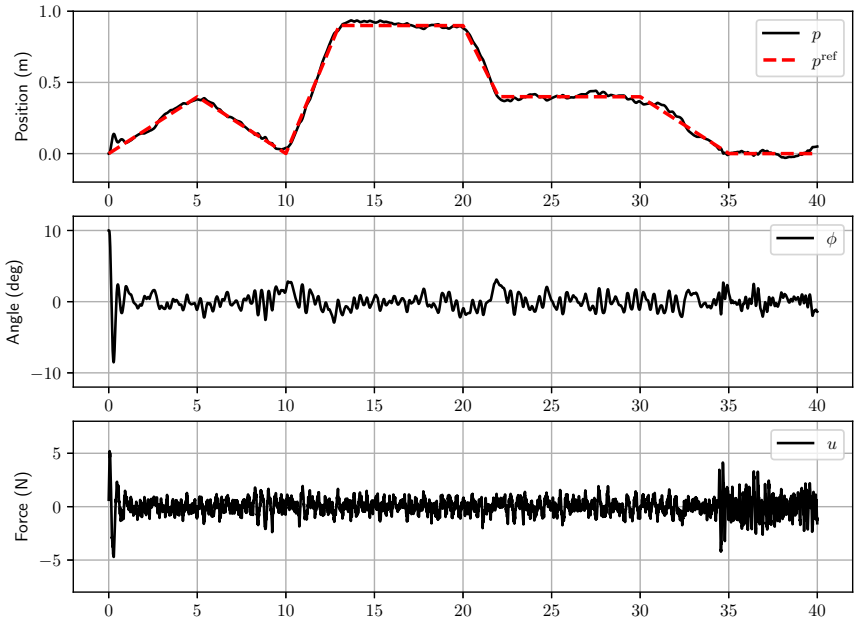
- Closed-loop performance score: $J = \int_0^T |p(t) - p_{\text{ref}}(t)| + 30|\phi(t)|dt$

- MPC parameters tuned using 500 iterations of GLIS

- Performance tested with simulated cart on two hardware platforms (PC, Raspberry PI)

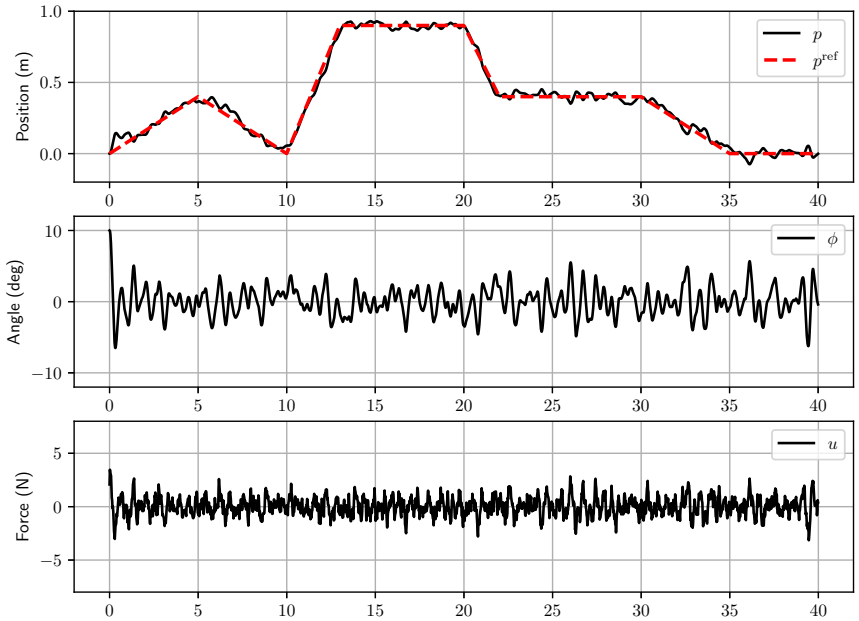
MPC AUTOTUNING EXAMPLE

MPC optimized for **desktop PC**



optimal sample time = **6 ms**

MPC optimized for **Raspberry PI**



optimal sample time = **22 ms**

- Auto-calibration can squeeze max performance out of the available hardware
- Bayesian Optimization gives similar results, but with larger computation effort

AUTO-TUNING: PROS AND CONS

- Pros:
  - Selection of calibration parameters x to test is fully automatic
  - Applicable to any calibration parameter (weights, horizons, solver tolerances, ...)
  - Rather arbitrary performance index $f(x)$ (tracking performance, response time, worst-case number of flops, ...)
- Cons:
  - Need to **quantify** an objective function $f(x)$
  - No room for **qualitative** assessments of closed-loop performance
  - Often have **multiple objectives**, not clear how to blend them in a single one

ACTIVE PREFERENCE LEARNING

(Bemporad, Piga, *Machine Learning*, 2021)

- Objective function $f(x)$ is not available (**latent function**)

- We can only express a **preference** between two choices:

$$\pi(x_1, x_2) = \begin{cases} -1 & \text{if } x_1 \text{ “better” than } x_2 & [f(x_1) < f(x_2)] \\ 0 & \text{if } x_1 \text{ “as good as” } x_2 & [f(x_1) = f(x_2)] \\ 1 & \text{if } x_2 \text{ “better” than } x_1 & [f(x_1) \geq f(x_2)] \end{cases}$$

- We want to find a global optimum $x^\star$ (=“better” than any other x)

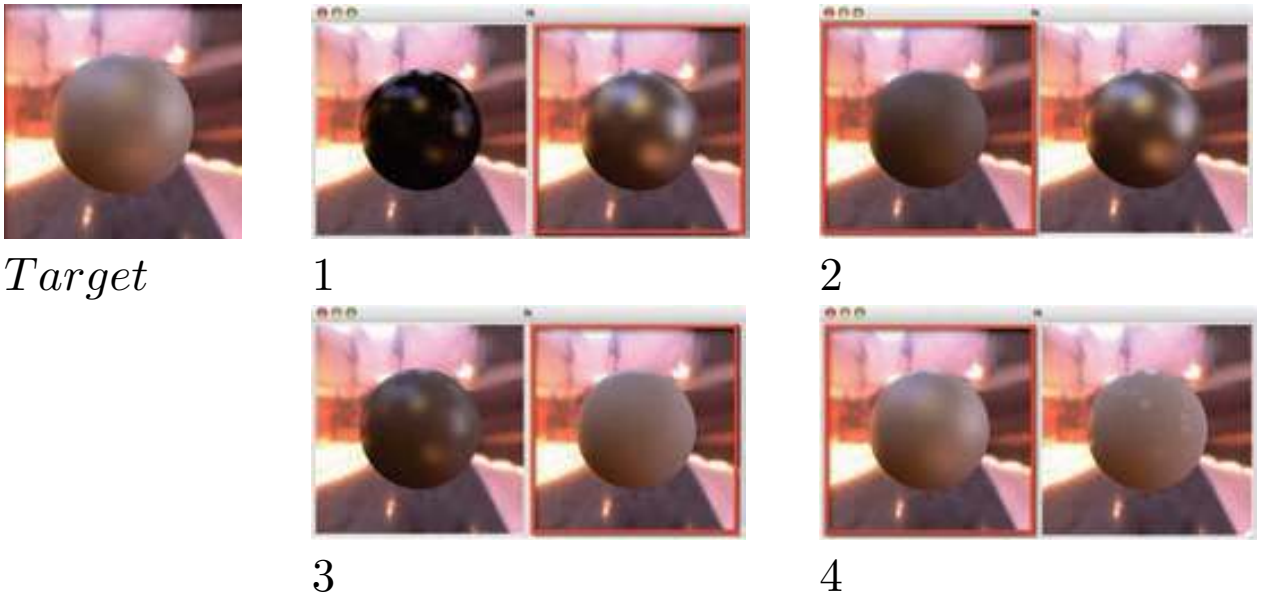
$$\text{find } x^\star \text{ such that } \pi(x^\star, x) \leq 0,\ \forall x \in \mathcal{X},\ \ell \leq x \leq u$$

- **Active preference learning**: iteratively propose a new sample to compare

- **Key idea**: learn a **surrogate** of the (latent) objective function from preferences

PREFERENCE-LEARNING EXAMPLE

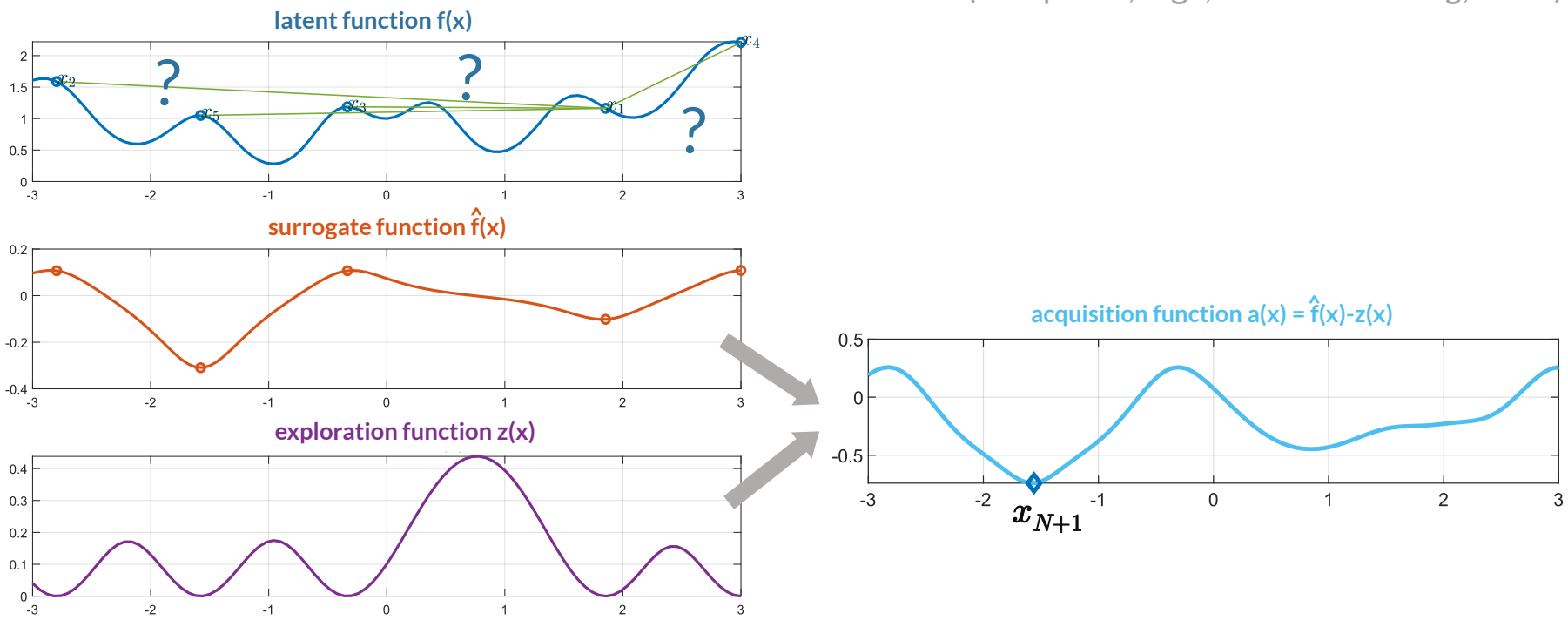
(Brochu, de Freitas, Ghosh, 2007)



- Realistic image synthesis of material appearance are based on models with many parameters $x_1, \ldots, x_n$
- Defining an objective function $f(x)$ is hard, while a human can easily assess whether an image resembles the target one or not
- **Preference gallery** tool: at each iteration, the user compares two images generated with two different parameter instances

ACTIVE PREFERENCE LEARNING ALGORITHM

(Bemporad, Piga, *Machine Learning*, 2021)



- **Fit a surrogate** $\hat{f}(x)$ that respects the **preferences** expressed by the decision maker at sampled points (by solving a QP)
- **Minimize an acquisition function** $\hat{f}(x) - \delta z(x)$ to get a **new sample** x_{N+1}
- **Compare** x_{N+1} to the current "best" point and **iterate**

SEMI-AUTOMATIC CALIBRATION BY PREFERENCE-BASED LEARNING

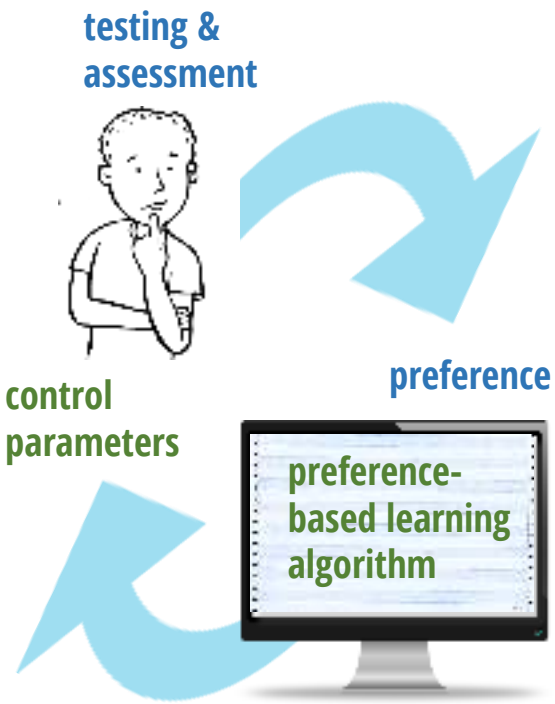
- Use **preference-based optimization** (**GLISp**) algorithm for **semi-automatic tuning** of MPC (Zhu, Bemporad, Piga, 2021)

- Latent function = calibrator's (unconscious) score of closed-loop MPC performance

- GLISp **proposes a new combination** x_{N+1} of MPC parameters to test

- By observing test results, the calibrator expresses a **preference**, telling if x_{N+1} is "**better**", "**similar**", or "**worse**" than current best combination

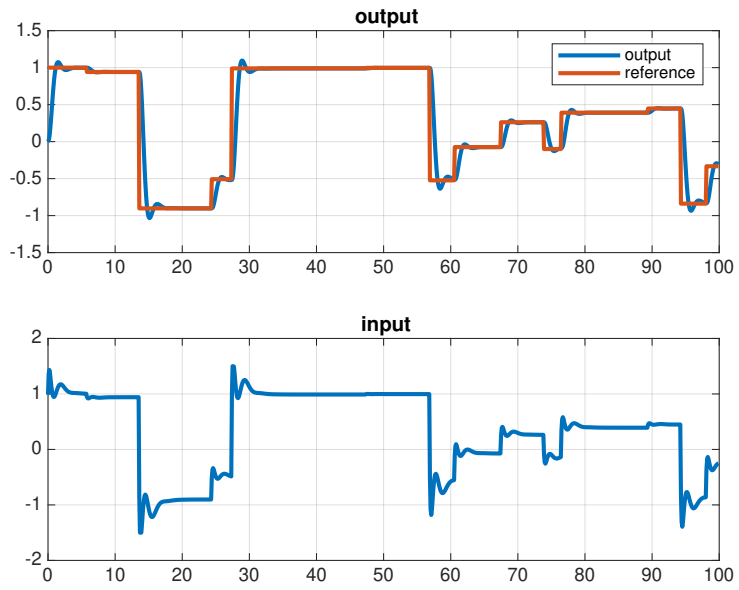
- Preference learning algorithm: **update the surrogate** $\hat{f}(x)$ of the latent function, optimize the acquisition function, **ask preference**, and **iterate**



PREFERENCE-BASED TUNING: MPC EXAMPLE

- Semi-automatic tuning of $x = [\log_{10} W^{\Delta u}, N_u]$ in linear MPC

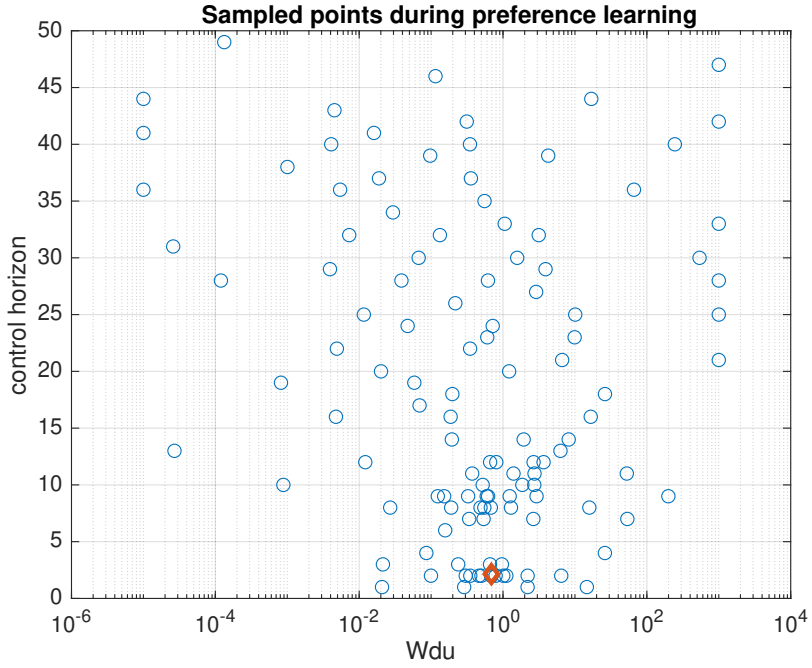
$$
\begin{array}{ll}
\min & \displaystyle\sum_{k=0}^{50-1} (y_{k+1} - r(t))^2 + (W^{\Delta u}(u_k - u_{k-1}))^2 \\
\text{s.t.} & x_{k+1} = Ax_k + Bu_k \\
& y_c = Cx_k \\
& -1.5 \leq u_k \leq 1.5 \\
& u_k \equiv u_{N_u}, \forall k = N_u, \ldots, N-1
\end{array}
$$



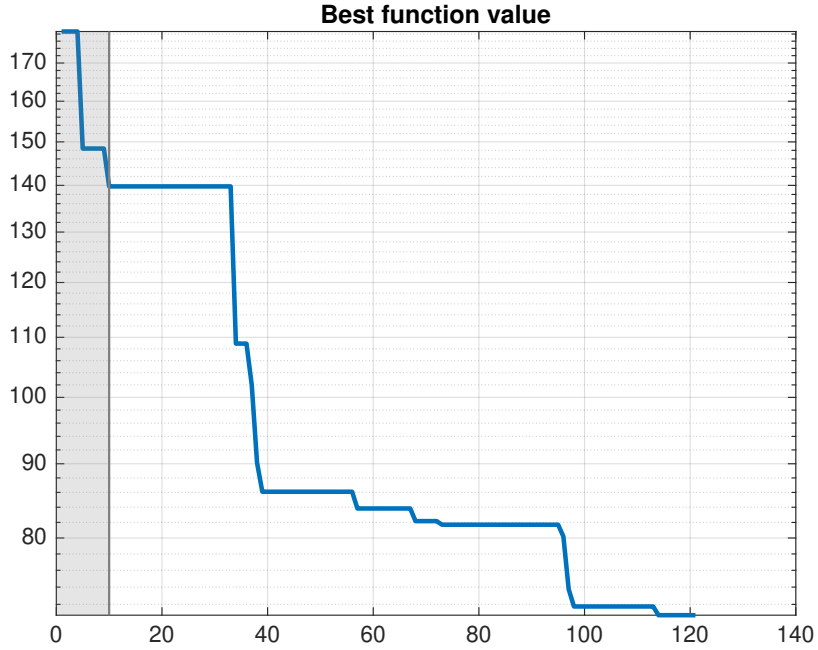
- Same performance index to assess closed-loop quality, but unknown: **only preferences** are available

- Result: $W^{\Delta u} = 0.6888, N_u = 2$

PREFERENCE-BASED TUNING: MPC EXAMPLE



tested combinations of MPC params



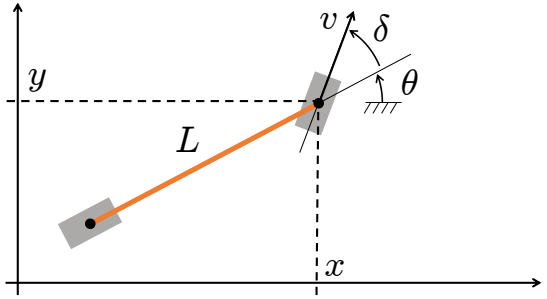
(latent) performance index

PREFERENCE-BASED TUNING: MPC EXAMPLE

(Zhu, Bemporad, Piga, 2021)

- Example: calibration of a simple MPC for lane-keeping (2 inputs, 3 outputs)

$$\begin{cases} \dot{x} &= v\cos(\theta+\delta) \\ \dot{y} &= v\sin(\theta+\delta) \\ \dot{\theta} &= \frac{1}{L}v\sin(\delta) \end{cases}$$



- Multiple control objectives:

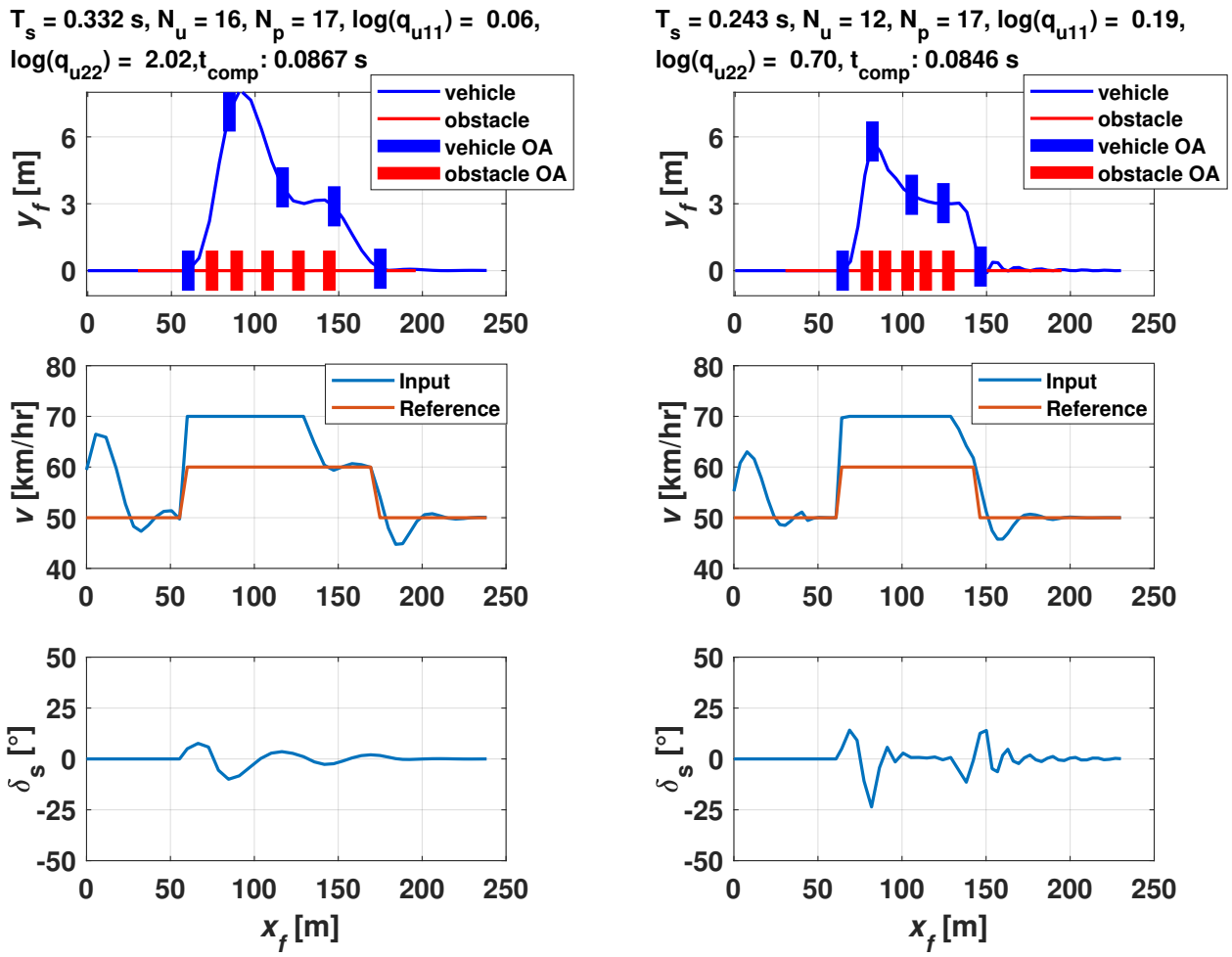
  "optimal obstacle avoidance", "pleasant drive", "CPU time small enough", ...

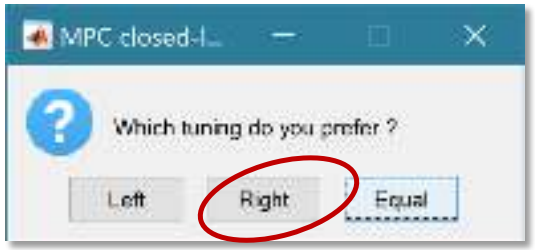
  ⇒ **not easy to quantify in a single function**

- 5 MPC parameters to tune:
  - **sampling time**
  - prediction and control **horizons**
  - **weights** on input increments Δv, $\Delta\delta$

PREFERENCE-BASED TUNING: MPC EXAMPLE

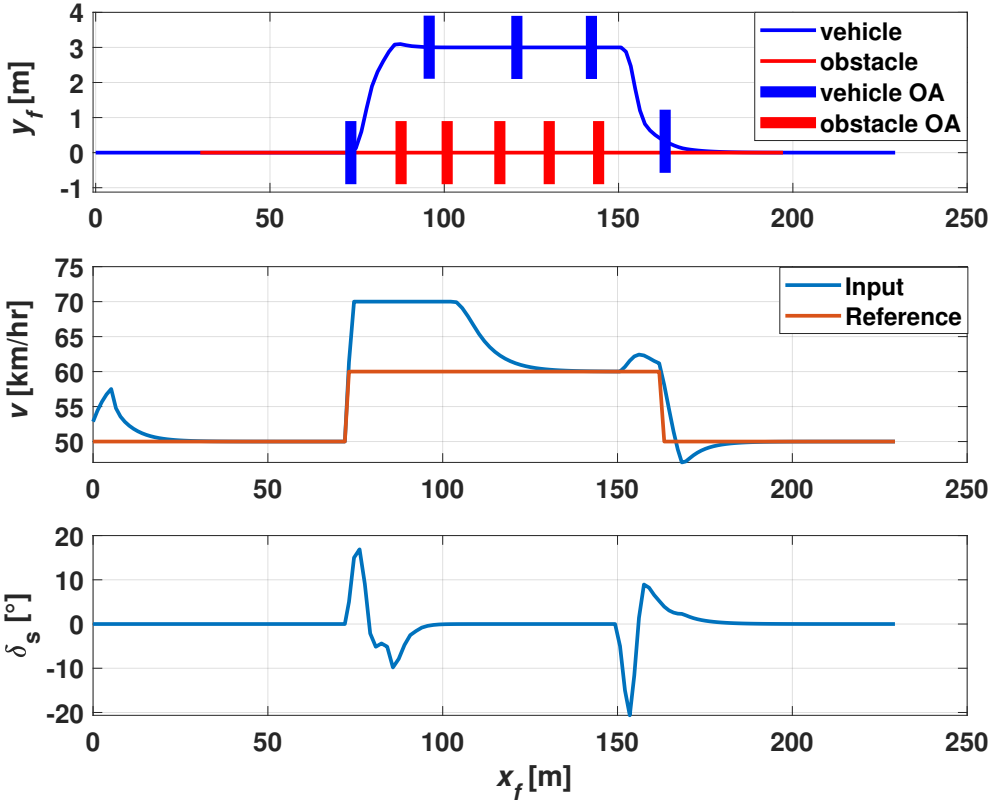
- Preference query window:





PREFERENCE-BASED TUNING: MPC EXAMPLE

- Convergence after 50 GLISp iterations (=49 queries):



Optimal MPC parameters:

- sample time = 85 ms (CPU time = 80.8 ms)
- prediction horizon = 16
- control horizon = 5
- weight on Δv = 1.82
- weight on $\Delta\delta$ = 8.28

- **Note**: no need to define a closed-loop performance index explicitly!

LEARNING MPC FROM DATA - LESSON LEARNED SO FAR

- Model/policy structure **includes** real plant/optimal policy:
  - **Sys-id + model-based** synthesis = model-free **reinforcement learning**
  - Reinforcement learning **may** require more data
    (model-based can instead "extrapolate" optimal actions)

- Model/policy structure **does not include** real plant/optimal policy:
  - optimal policy **learned from data** **may** be better than **model-based** optimal policy
  - when open-loop model is used as a tuning parameter, **learned model** can be quite different from best **open-loop model** that can be identified from the same data