MACHINE LEARNING METHODS FOR MODEL PREDICTIVE CONTROL

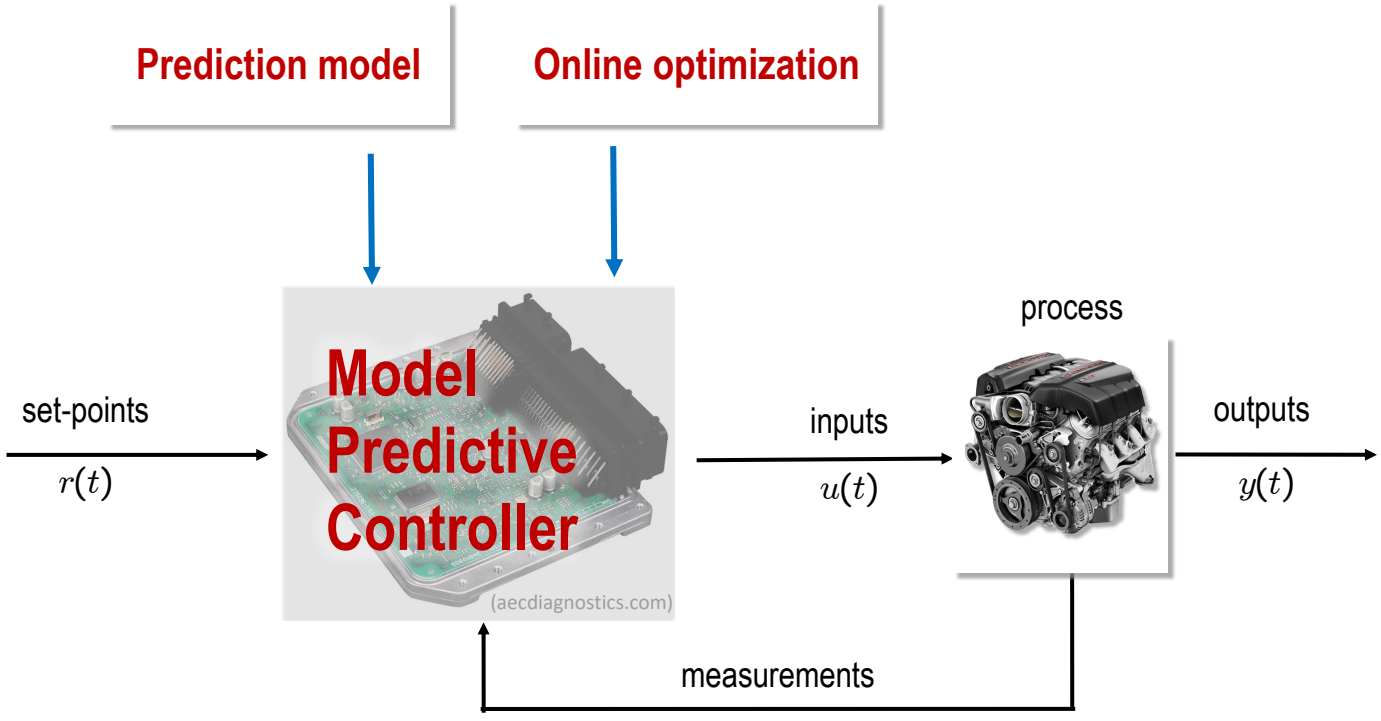
Alberto Bemporad

imt.lu/ab

European Control Conference 2021 - July 2, 2021

IMT SCHOOL FOR ADVANCED STUDIES LUCCA

MODEL PREDICTIVE CONTROL



- Use a dynamical (M)**odel** of the process to (P)**redict** its future evolution and choose the "best" (C)**ontrol** action

MODEL PREDICTIVE CONTROL

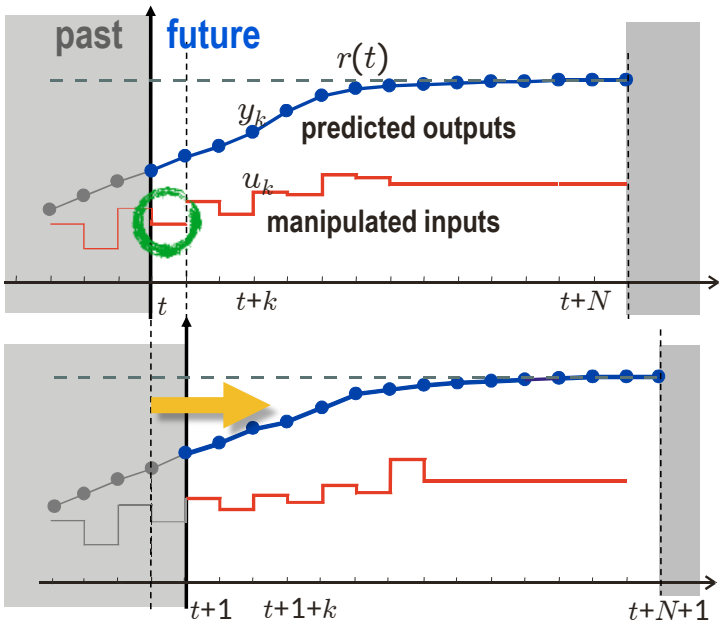
- **MPC problem:** find the best control sequence over a future horizon of N steps

$$\min \sum_{k=0}^{N-1} \|W^y(y_k - r(t))\|_2^2 + \|W^u(u_k - u_r(t))\|_2^2$$

$$\text{s.t.} \quad x_{k+1} = f(x_k, u_k) \qquad \text{prediction model}$$
$$y_k = g(x_k)$$

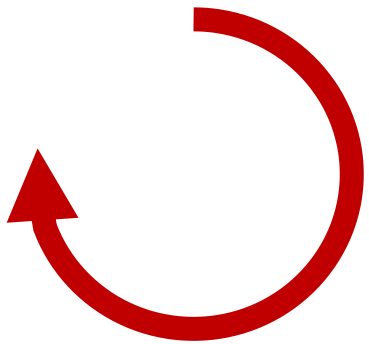
$$u_{\min} \le u_k \le u_{\max} \qquad \text{constraints}$$
$$y_{\min} \le y_k \le y_{\max}$$

$$x_0 = x(t) \qquad \text{state feedback}$$



➡ **numerical optimization problem**

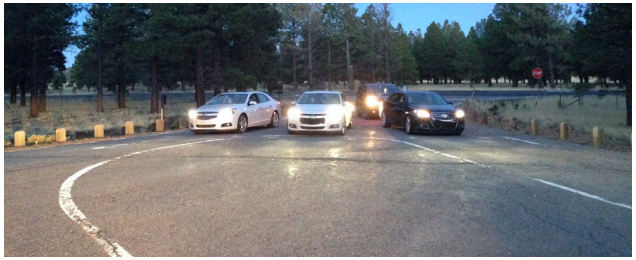
1. **estimate** current state $x(t)$
2. **optimize** wrt $\{u_0, \ldots, u_{N-1}\}$
3. only **apply** optimal u_0 as input $u(t)$



MODEL PREDICTIVE CONTROL

- Conceived in the 60's (Rafal, Stevens, 1968) (Propoi, 1963)

- Used in the **process industries** since the 80's (Qin, Badgewell, 2003)
- Nowadays spreading to the automotive industry and other sectors
- An MPC for engine control developed by **General Motors** and **ODYS** is in high-volume production since 2018 (Bemporad, Bernardini, Long, Verdejo, 2018)

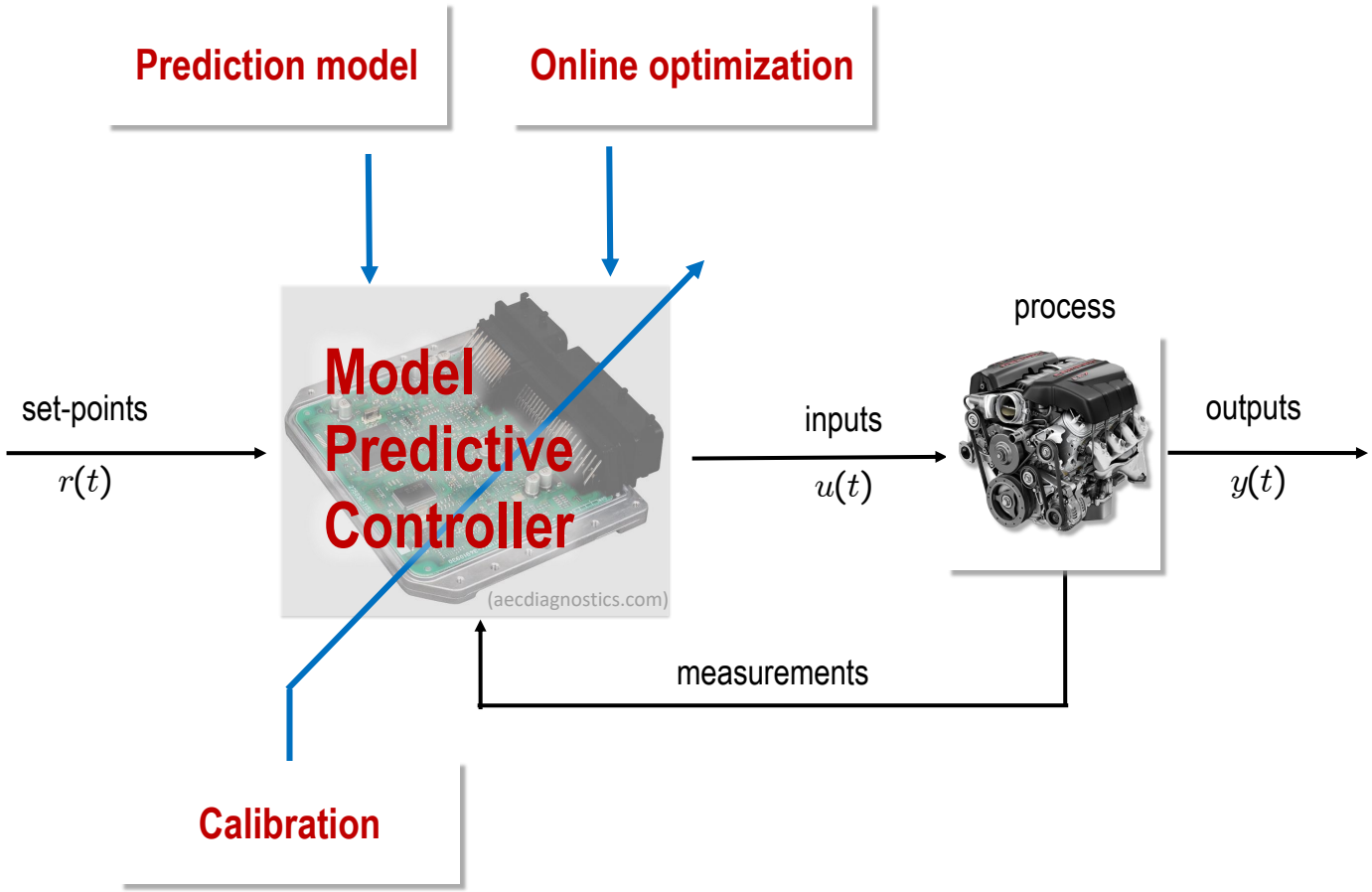


First known mass production of MPC in the automotive industry

ODYS
Advanced Controls & Optimization

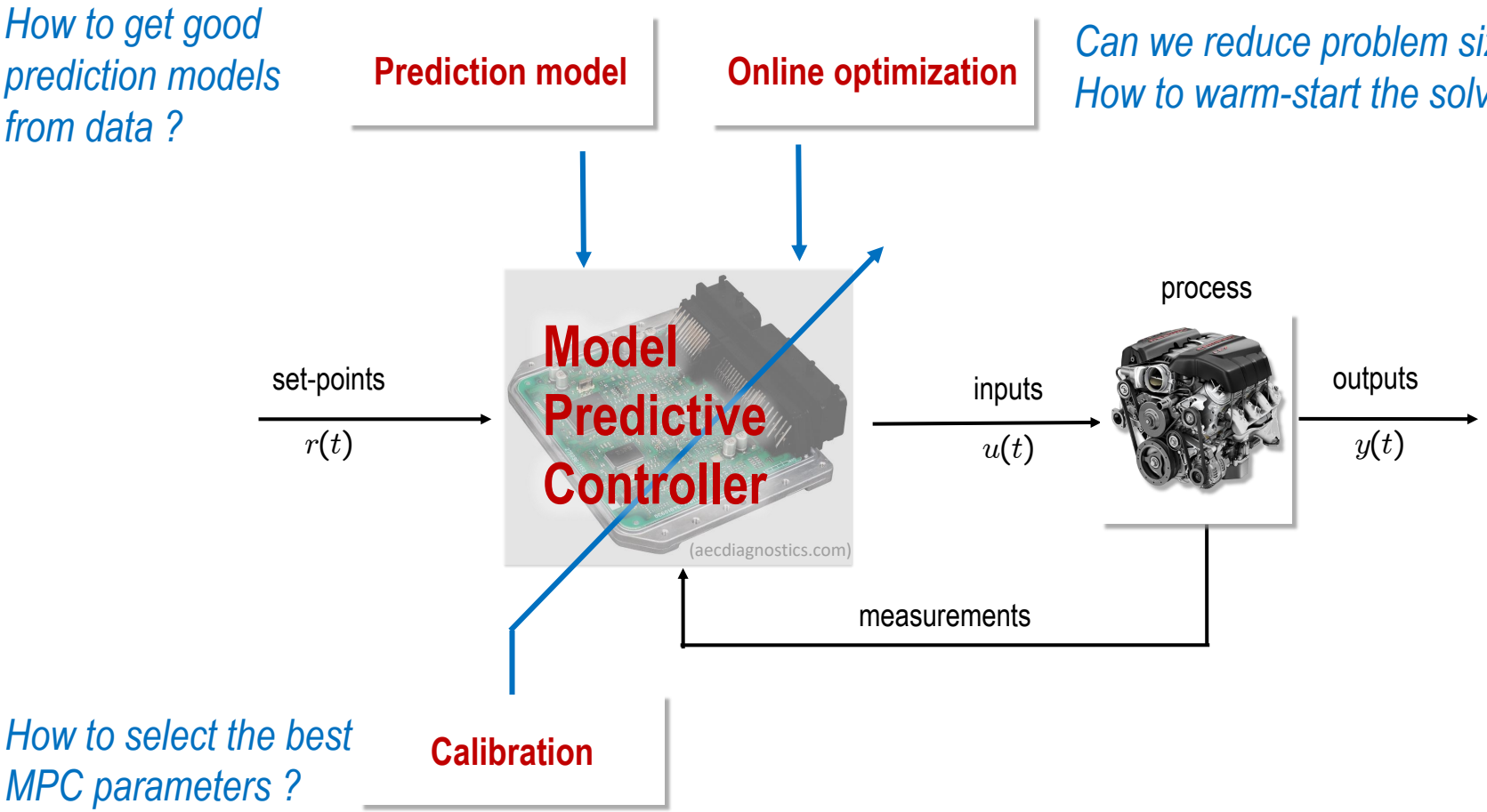
www.odys.it

MODEL PREDICTIVE CONTROL



- MPC requires a **prediction model**, a way to solve **on-line optimization** efficiently, and to **calibrate** various parameters (weights, horizon, etc.)

MODEL PREDICTIVE CONTROL

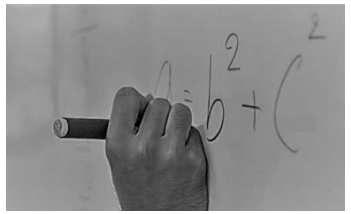


- **Machine Learning (ML)** can help addressing the above questions

OUTLINE

- **Prediction models**
  - **Autoencoders** to learn **nonlinear models** from data
  - **Softmax+ridge regression** to learn **hybrid models** from data

- **Online optimization**
  - **Binary classification** to learn the **initial guess** for hybrid MPC
  - **Unsupervised learning** + **SVD** to reduce the number of optimization variables in linear MPC



- **(Semi)automatic calibration** using **radial basis functions**
  - Learning the best MPC parameters from closed-loop experiments
  - Active preference-based learning from human assessments

LEARNING PREDICTION MODELS

MACHINE LEARNING (ML)

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

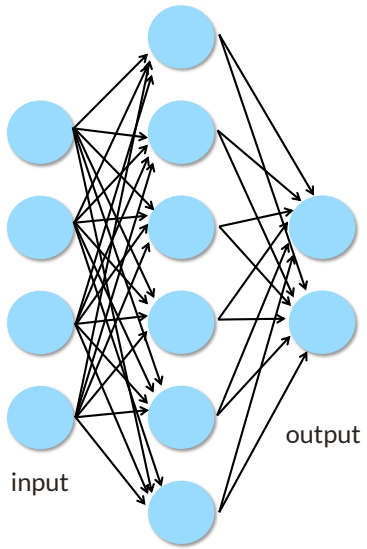
- Good **mathematical foundations** from artificial intelligence, statistics, optimization

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

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

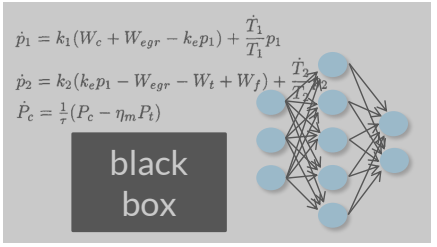
- Availability of excellent open-source **software tools** also explains success

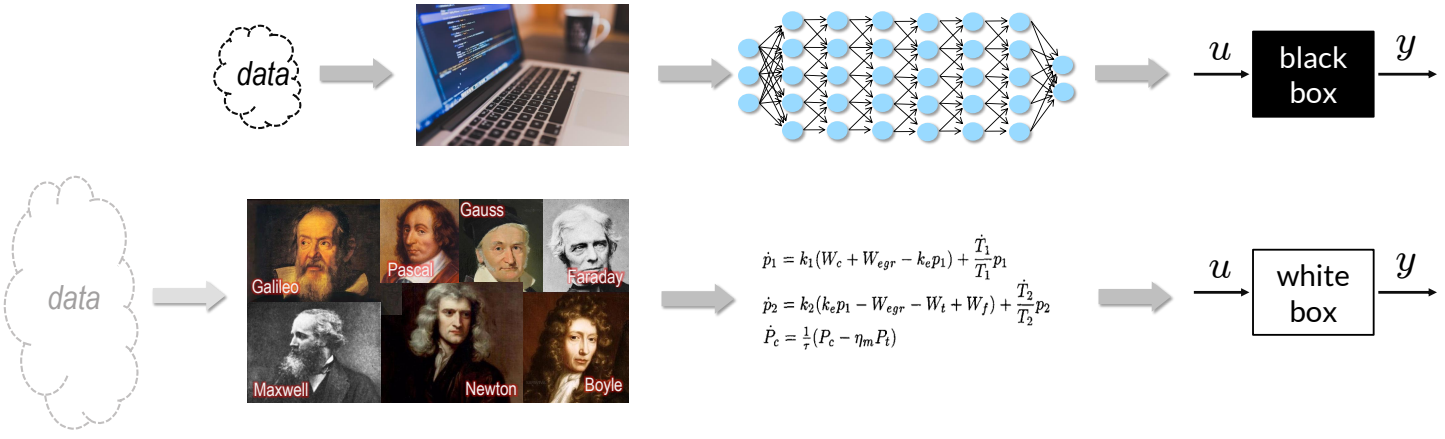
  `scikit-learn`, `TensorFlow/Keras`, `PyTorch`, ... (python)



NONLINEAR PREDICTION MODELS

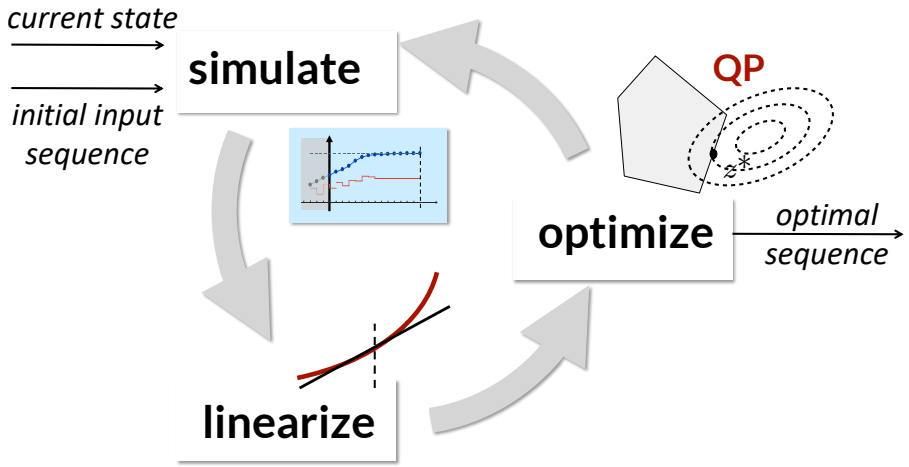
- **Physics-based** nonlinear models (=white-box models)
- **Black-box** nonlinear models (**NL SYS-ID/machine learning**)
- A mix of the above (**gray-box** models) is often the best
- **Online computation complexity** depends on chosen model
- **Jacobians** of prediction models are required





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 (Gros, Zanon, Quirynen, Bemporad, Diehl, 2020)
- Special case of **single QP** = LTV-MPC (a.k.a. **Real-Time Iteration**)

  (Diehl, Bock, Schloder, Findeisen, Nagy, Allgower, 2002)
- Same model can be used for state estimation (e.g., **extended Kalman filtering**)

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:

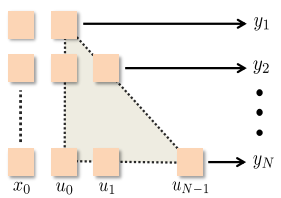
  - **w/ state data**: fit a neural network model $x_{t+1} \approx \mathcal{N}_x(x_t, u_t), \quad y_t \approx \mathcal{N}_y(x_t)$

  - **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)

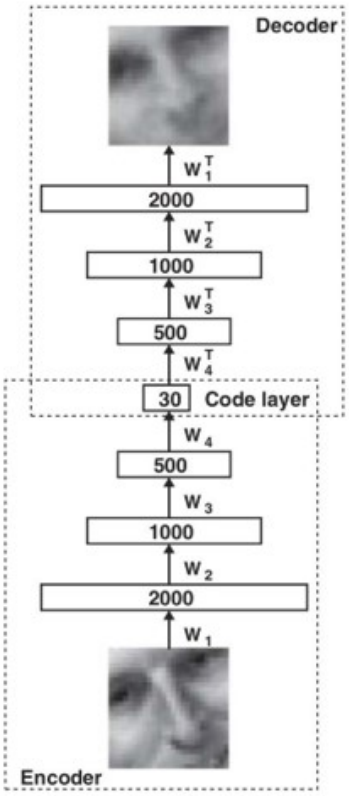
$$y_{t+k} = h_k(x_t, u_t, \ldots, u_{t+k-1}),\ k = 1, \ldots, N$$



LEARNING NONLINEAR STATE-SPACE MODELS FOR MPC

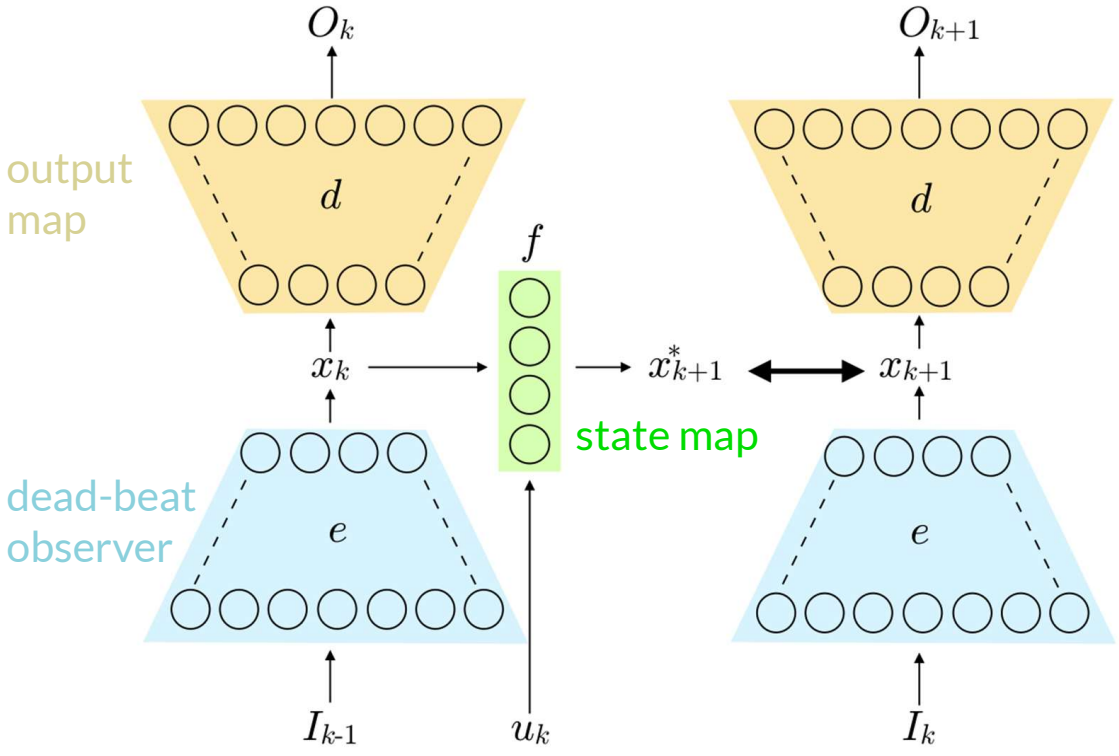
(Masti, Bemporad, 2021)

- **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)



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

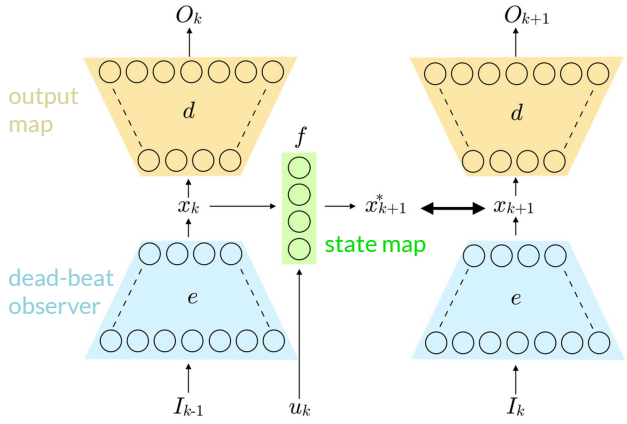
$$I_k = [y'_k \ \dots \ y'_{k-n_a+1} \ u'_k \ \dots \ u'_{k-n_b+1}]'$$

LEARNING NONLINEAR STATE-SPACE MODELS FOR MPC

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

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

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



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

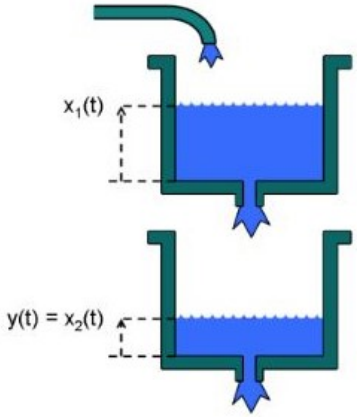
- **Quasi-LPV** structure for MPC: set $(A_{ij}, B_{ij}, C_{ij}$ = feedforward NNs)

$$f(x_k, u_k) = A(x_k, u_k) \begin{bmatrix} x_k \\ 1 \end{bmatrix} + B(x_k, u_k) u_k$$
$$y_k = C(x_k, u_k) \begin{bmatrix} x_k \\ 1 \end{bmatrix}$$

- 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

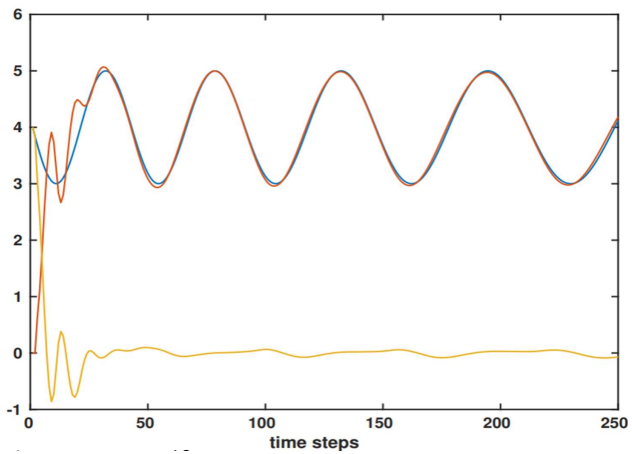


www.mathworks.com

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

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



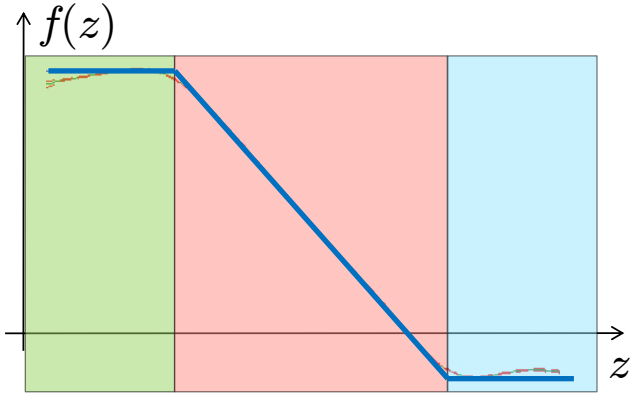
LTV-MPC results

LEARNING HYBRID MODELS

- **Switching dynamics** are better captured by **piecewise affine (PWA)** models and handled by **hybrid MPC** techniques

$$v(k) = \begin{cases} F_1 z(k) + g_1 & \text{if } H_1 z(k) \le K_1 \\ \vdots & \\ F_s z(k) + g_s & \text{if } H_s z(k) \le K_s \end{cases}$$

$$v(k) = \begin{bmatrix} x(k+1) \\ y(k) \end{bmatrix}, \quad z(k) = \begin{bmatrix} x(k) \\ u(k) \end{bmatrix}$$



- **PWA regression**: learn **both** the $\{F_i,\ g_i\}$ **and** the partition $\{H_i,\ K_i\}$

(Ferrari-Trecate, Muselli, Liberati, Morari, 2003) (Roll, Bemporad, Ljung, 2004) (Juloski, Wieland, Heemels, 2004) (Bemporad, Garulli, Paoletti, Vicino, 2005) (Pillonetto, 2016) (Breschi, Piga, Bemporad, 2016) (...)

- Any **ML technique** can be applied that leads to PWA models, such as **ReLU-NNs**, **decision trees**, **softmax regression**, **KNN**, ...

PARC - PIECEWISE AFFINE REGRESSION AND CLASSIFICATION

(Bemporad, 2021, arXiv)

- New **Piecewise Affine Regression and Classification** (**PARC**) algorithm
- Training dataset:
  - **feature vector** $z \in \mathbb{R}^n$ (categorical features **one-hot encoded** in $\{0,1\}$)
  - **target vector** $v_c \in \mathbb{R}^{m_c}$ (numeric), $v_{di} \in \{w_{di}^1, \ldots, w_{di}^{m_i}\}$ (categorical)
- PARC iteratively **clusters** training data in K sets and **fit** linear predictors
  1. fit $v_c = a_j z + b_j$ by **ridge regression** (=ℓ_2-regularized least squares)
  2. fit $v_{di} = w_{di}^{h_*}, h_* = \arg\max\{a_{dih}^h z + b_{di}^h\}$ by **softmax regression**
  3. fit a convex **PWL separation function** by **softmax regression**

$$\Phi(z) = \omega^{j(z)} z + \gamma^{j(z)}, \qquad j(z) = \min\left\{\arg\max_{j=1,\ldots,K}\{\omega^j z + \gamma^j\}\right\}$$

- Data reassigned to clusters based on weighted fit/PWL separation criterion
- PARC is a **block-coordinate descent** algorithm $\Rightarrow$ (local) convergence ensured

PARC - PIECEWISE AFFINE REGRESSION AND CLASSIFICATION

- PARC for learning hybrid models (state-space form):

$$\textbf{feature vector:} \quad z_k = \begin{bmatrix} x_c(k) \\ u_c(k) \\ x_\ell(k) \\ u_\ell(k) \end{bmatrix} \in \mathbb{R}^{n_c+m_c} \times \{0,1\}^{n_\ell+m_\ell}$$

$$\textbf{target vector:} \quad v_k = \begin{bmatrix} x_c(k+1) \\ y_c(k) \\ x_\ell(k+1) \\ y_\ell(k) \end{bmatrix} \in \mathbb{R}^{n_c+p_c} \times \{0,1\}^{n_\ell+p_\ell}$$

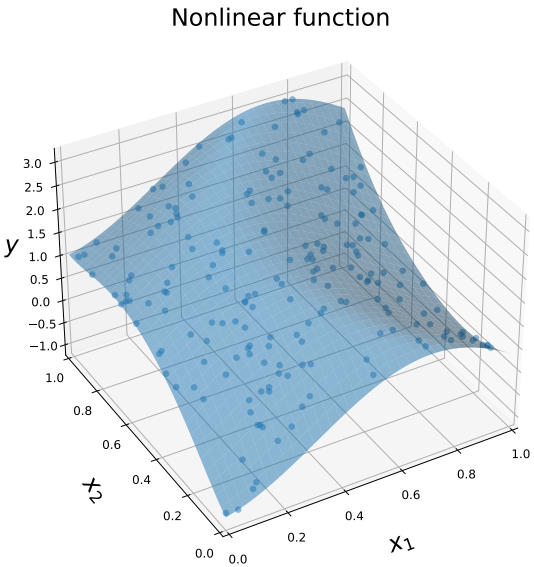
- **Input**: dataset $\{z_k, v_k\}_{k=0}^{N-1}$ and desired number K of partitions.

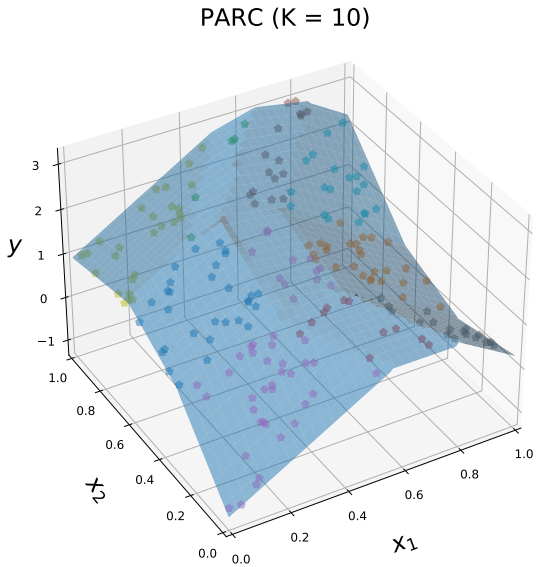
- The **PARC** algorithm iteratively finds:

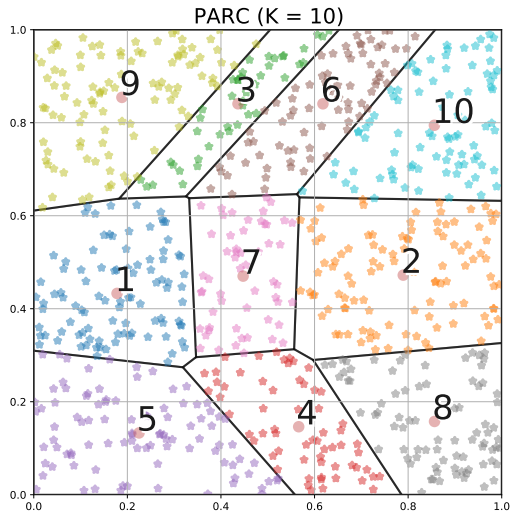
1. $\begin{bmatrix} x_c(k+1) \\ y_c(k) \end{bmatrix} = a_j z_k + b_j$ by **ridge regression** (=ℓ_2-regularized least squares) $\quad j = 1, \dots, K$
2. $\begin{bmatrix} x_\ell(k+1) \\ y_\ell(k) \end{bmatrix}_i = \frac{1}{2}(1 + \text{sign}(c_j^i z_k + d_j^i))$ by **logistic regression** $\quad j = 1, \dots, K$
3. A PWA partition $P_i = \{z : \omega^i z + \gamma^i \geq \omega^j z + \gamma^j, \forall j \neq i\}$ by **softmax regression** (possibly sub-partitioned by categorical target functions)

PARC - PIECEWISE AFFINE REGRESSION AND CLASSIFICATION

- Simple PWA regression example:
  - 1000 samples of $y = \sin(4x_1 - 5(x_2 - 0.5)^2) + 2x_2$ (use 80% for training)
  - Look for PWA approximation over $K = 10$ polyhedral regions

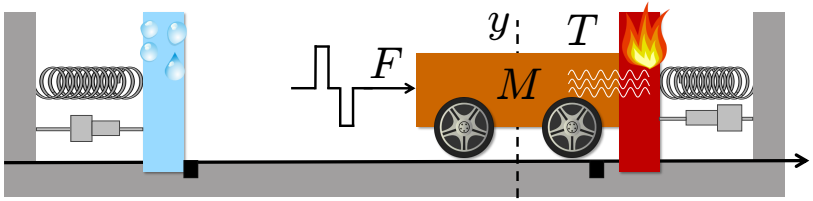




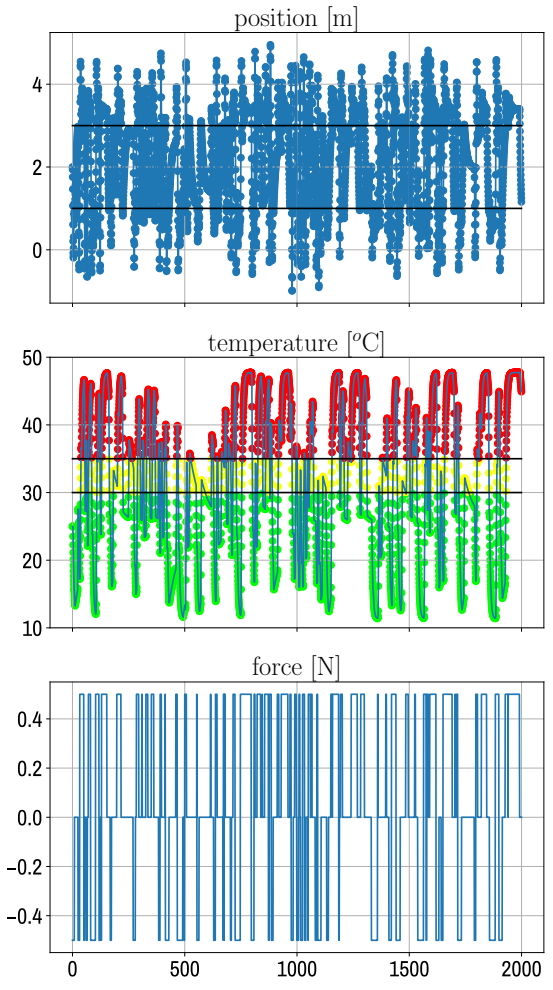


- Code download:  `http://cse.lab.imtlucca.it/~bemporad/parc/`

DATA-DRIVEN HYBRID-MPC EXAMPLE

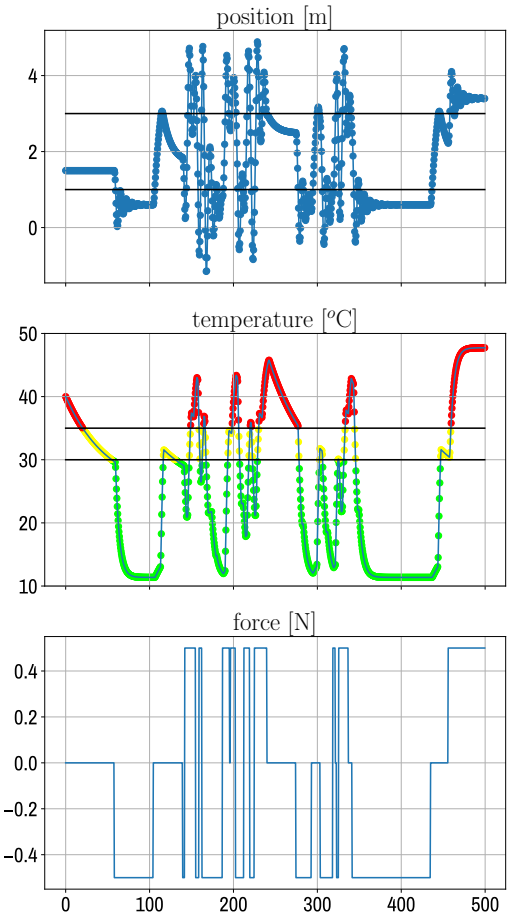


- **Example**: moving cart and bumpers + heat transfer during bumps
- Categorical input $F \in \{-\bar{F}, 0, \bar{F}\}$ and output $c \in \{green, yellow, red\}$
- Continuous-time system simulated for 2,000 s, sample time = 0.5 s (=4000 training samples)
- Feature vector $z_k = [y_k, \dot{y}_k, T_k, F_k]$
- Target vector $v_k = [y_{k+1}, \dot{y}_{k+1}, T_{k+1}, c_k]$
- Hybrid model learned by **PARC** ($K = 6$ regions)

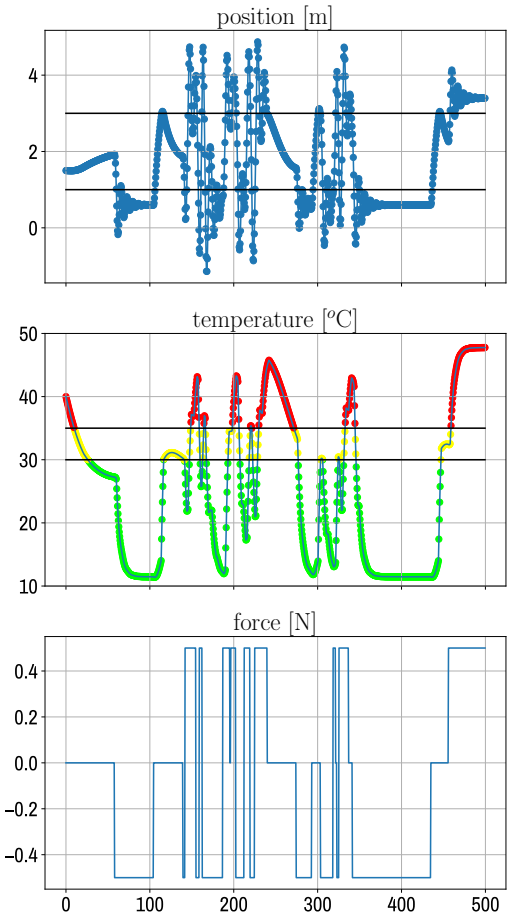


DATA-DRIVEN HYBRID-MPC EXAMPLE

- **Open-loop** simulation on 500 s **test** data:



continuous-time system



discrete-time PWA model

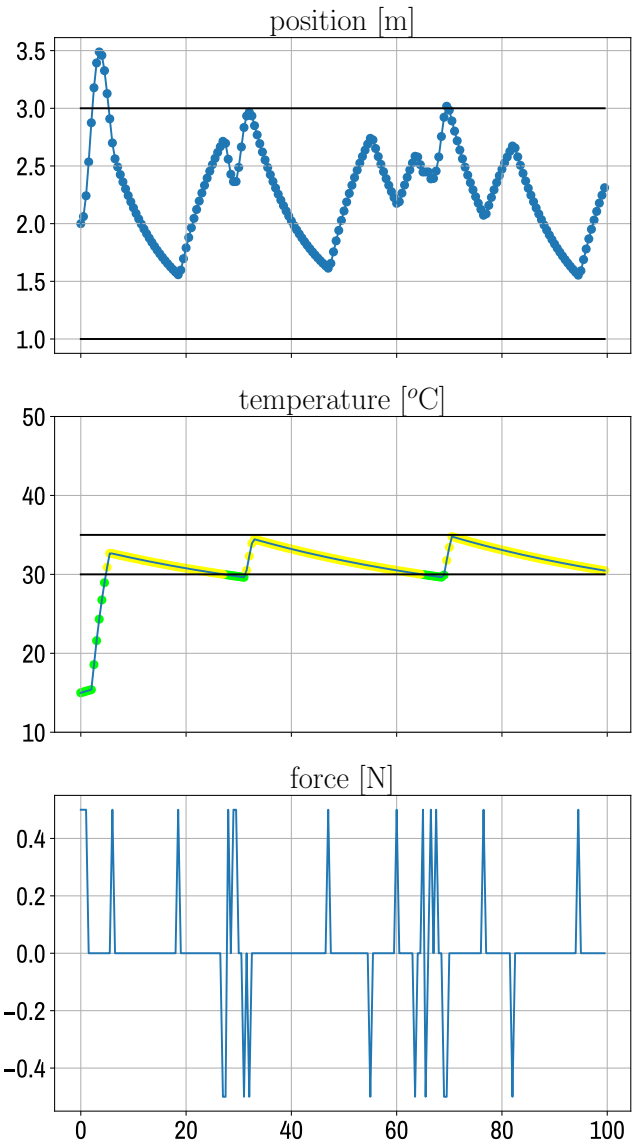
- Model fit is good enough

DATA-DRIVEN HYBRID-MPC EXAMPLE

- MPC problem with prediction horizon $N = 20$:

$$\begin{aligned} \min_{F_0,\ldots,F_{N-1}} \quad & \sum_{k=0}^{N-1} |c_k - 1| + 0.2|F_k| \\ \text{s.t.} \quad & F_k \in \{-\bar{F}, 0, \bar{F}\} \\ & \text{PWA model equations} \end{aligned}$$

- Problem can be cast to MILP
- Continuous-time system in closed-loop with hybrid MPC simulated for 100 s
- **Data-driven hybrid MPC** controller is able to keep mass temperature in yellow zone



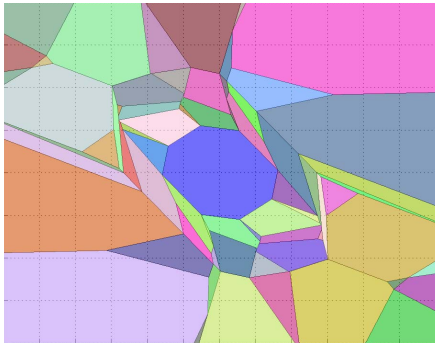
REDUCTION OF ONLINE COMPUTATIONS

(APPROXIMATE) EXPLICIT MPC

- **Explicit MPC** solves (small, LTI) MPC problems offline **exactly**

$$u^*(x) = \begin{cases} F_1 x + g_1 & \text{if} \quad H_1 x \leq K_1 \\ \vdots & \vdots \\ F_M x + g_M & \text{if} \quad H_M x \leq K_M \end{cases}$$

(Bemporad, Morari, Dua, Pistikopoulos, 2002)



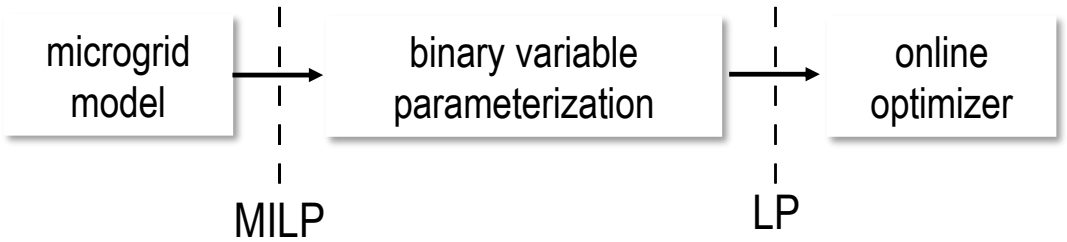
- Limited to small (LTI) problems, online QP is often lighter (Cimini, Bemporad, 2017)
- Any **function regression** technique can be used to **approximate** MPC laws:
  - Collect M samples (x_i, u_i) by solving MPC optimization problem for each x_i
  - Fit approximate mapping $\hat{u}(x)$ on the samples, such as using
    - **Neural networks** (Parisini, Zoppoli, 1995) (Karg, Lucia, 2018)
    - **PWA regression** (...)
    - **Nonlinear system identification** (Canale, Fagiano, Milanese, 2008)
  - Performance / feasibility/ closed-loop stability usually **verified a posteriori**

SEMI-EXPLICIT MPC

- **Semi-explicit MPC**: use **binary classification** to learn the optimal **active set** of a parametric QP for warm start (Klauco, Kalúz, Kvasnica, 2019)

- Learn optimal **binary variables** $\delta^*(x)$ of parametric MIQP/LP, then solve QP/LP online, or warm-start MIP solver (Masti, Bemporad, 2019)

- **Example**: hybrid MPC of for **microgrid optimization**

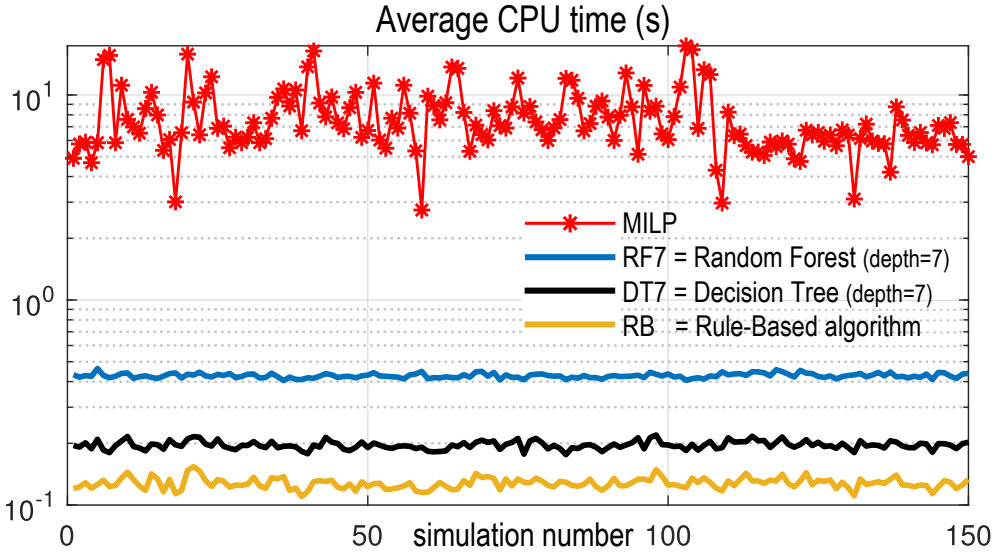


15,725 MILP solutions collected to train $\delta^*(x)$

decision tree and **random forest** classifiers compared to exact MILP solution and **rule-based** controller (Masti, Pippia, Bemporad, De Schutter, 2020)

SEMI-EXPLICIT MPC

- Example: hybrid MPC of a **microgrid** (cont'd)



	OL	CL
MILP	4202.4	4737.0
RB	4341.1 (3.3%)	4779.6 (0.9%)
RF7	4270.5 (1.6%)	4786.6 (1.0%)
DT7	4296.5(2.2%)	4761.9 (0.5%)

	RB	RF7	DT7
% infeasible	0%	1.02%	8.04%

(Pippia, 2020)

- RF7 and DT7 reduce online CPU time by $\approx$ **96÷98 %**
- Optimal costs of exact MILP and RF7 are similar
- RF7 produces almost always feasible solutions
- RB controller is lighter better **but** requires deep domain-specific knowledge !

LOSSLESS REDUCTION OF MPC PROBLEM VARIABLES

(Bemporad, Cimini, 2020, arXiv)

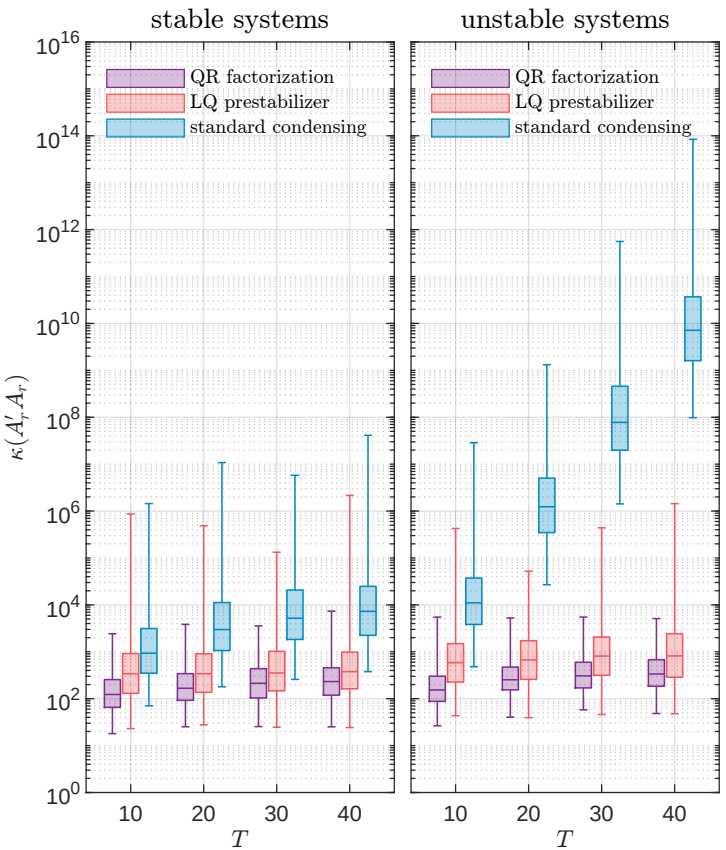
- Linear (parameter-varying) MPC = **constrained least squares problem**

$$\begin{aligned} \min_z \quad & \tfrac{1}{2}\|A(\theta)z - b(\theta)\|_2^2 \\ \text{s.t.} \quad & C(\theta)z = e(\theta) \\ & G(\theta)z \le g(\theta) \end{aligned} \qquad z = \begin{bmatrix} u_0 \\ x_1 \\ \vdots \\ u_{T-1} \\ x_T \end{bmatrix}, \quad \theta = \begin{bmatrix} x(t) \\ r(t) \\ \cdots \end{bmatrix}$$

- **Standard condensing**: eliminate x_k, only keep u_k as optimization variables
- **LQ prestabilizer**: apply $u_k = K_k x_k + v_k$ and condense
- **QR factorization**: $C' = [Q_1\ Q_2] \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$

  $z = Q_2(\theta)s + \bar{z}(\theta) \in \mathbb{R}^{T(n_x+n_u)}$

  $s \in \mathbb{R}^{Tn_u}$ = new variables



numerically very robust

LOSSY REDUCTION OF MPC PROBLEM VARIABLES

(Bemporad, Cimini, 2020, arXiv)

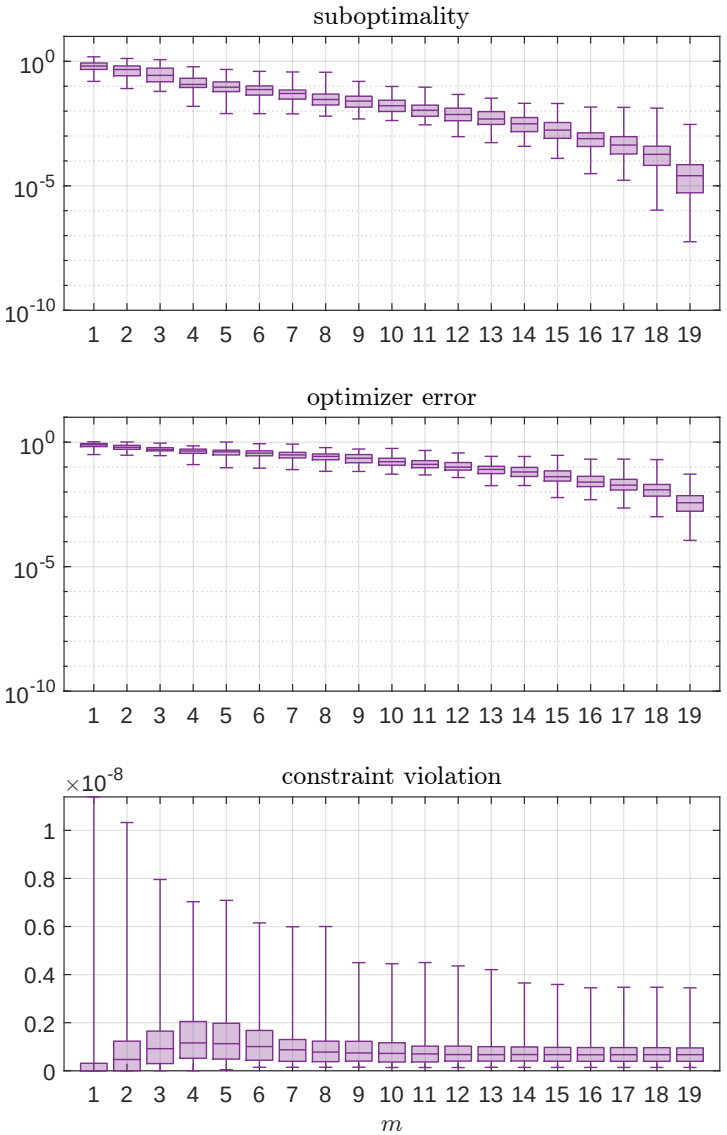
- **Control horizon** often used to reduce # vars
- **Idea**: **linear PCA** to reduce to $m < Tn_u$ vars
  - collect optimal solutions s_k^* for M given values of θ_k, remove mean $\bar{s}$
  - compute **Singular Value Decomposition**

$$S = \begin{bmatrix} s_1^* - \bar{s} & \dots & s_M^* - \bar{s} \end{bmatrix}' = U\Sigma V'$$

  - keep only first m principal directions $\Phi = [V_1 \ \dots \ V_m]$
  - new optimization vector $v \in \mathbb{R}^m$

$$s = \Phi v + \bar{s}$$

- Complexity / solution quality **tradeoff**



LOSSY REDUCTION OF MPC PROBLEM VARIABLES

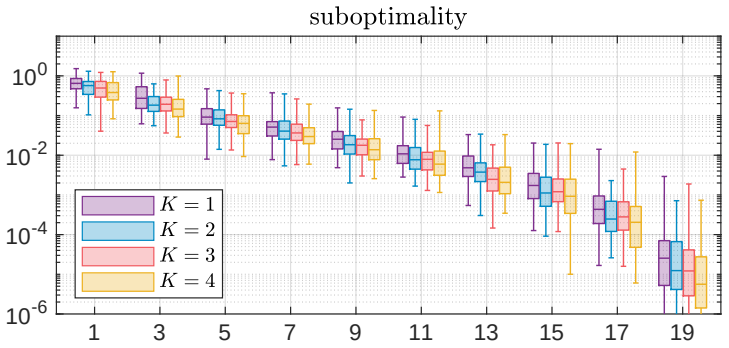
(Bemporad, Cimini, 2020, arXiv)

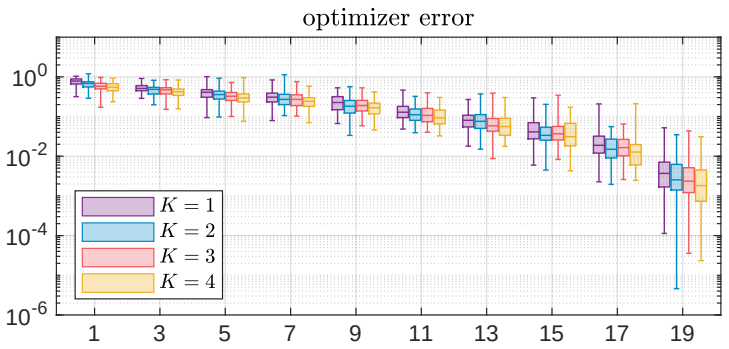
- The optimal basis Φ is an average over all θ_k
- **K-SVD**: a new "K-means"-like algorithm to cluster $\theta_1, \ldots, \theta_M$ in K sets and get corresponding bases Φ_1,...,Φ_K
- K-SVD converges in a finite number of steps to a local minimum of

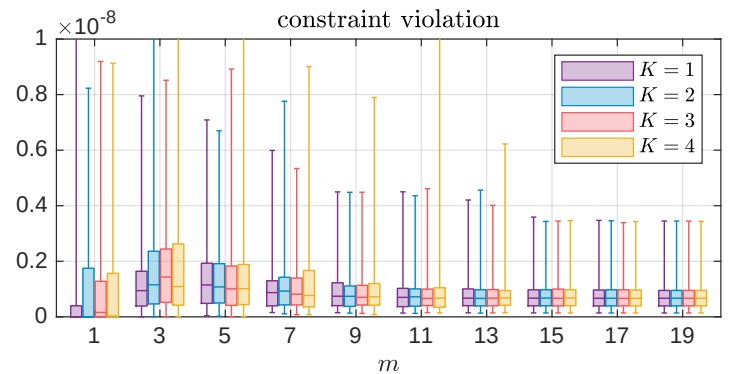
$$\min_{J,\{\Phi_j,\phi_0^j\}_{j=1}^K} \sum_{i=1}^{M} \min_{v} \|s_i^* - \Phi_{J(i)} v - \phi_0^{J(i)}\|_2^2$$

$$\text{s.t.} \quad \phi_0^j = \frac{1}{M_j} \sum_{i \in I_j} s_i^*,\ j = 1, \ldots, K$$

- K **neural one-to-all classifiers** trained to separate the resulting clusters



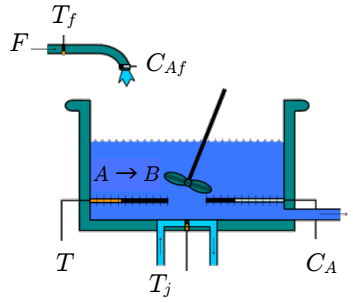


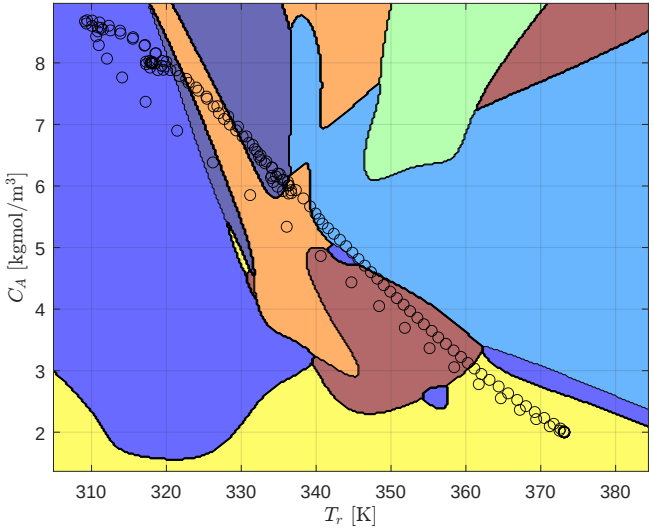


LOSSY REDUCTION OF MPC PROBLEM VARIABLES

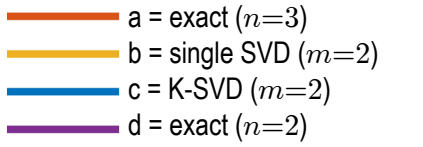
(Bemporad, Cimini, 2020, arXiv)

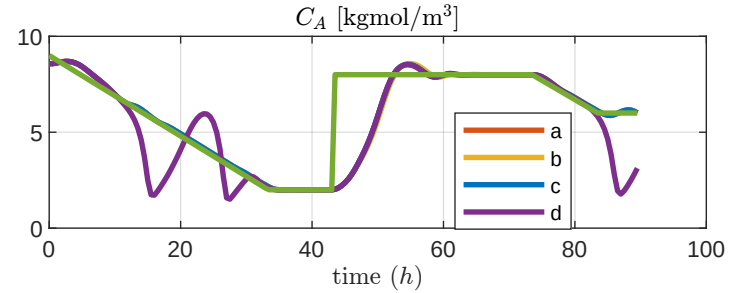
- Example: LPV-MPC control of CSTR
- M=10,000 samples, $K = 10$ clusters

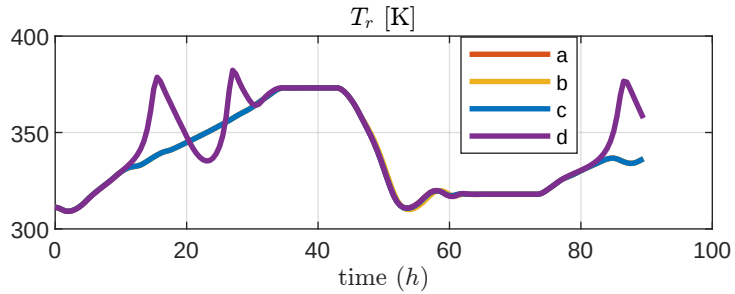


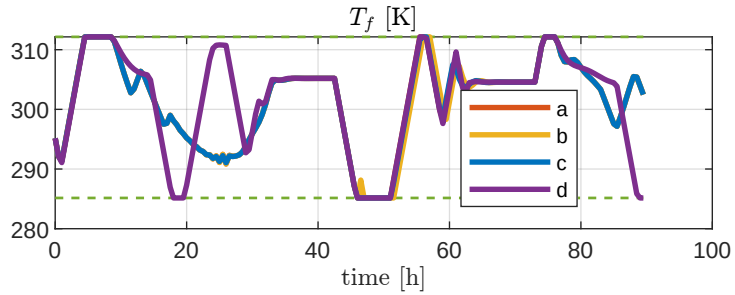


performance	value	MPC setting
$J_{\text{exact},20}$	319.68	$n = 20 = T$, exact
$J_{\text{exact},10}$	319.69	$n = 10$, exact
$J_{\text{exact},4}$	319.93	$n = 4$, exact
$J_{\text{exact},3}$	320.17	$n = 3$, exact
$J_{\text{exact},2}$	567.86	$n = 2$, exact
J_{svd}	328.33	$n = 3$, $m = 2$ (single SVD)
J_{ksvd}	320.17	$n = 3$, $m = 2$ (K-SVD)









LEARNING OPTIMAL MPC CALIBRATION

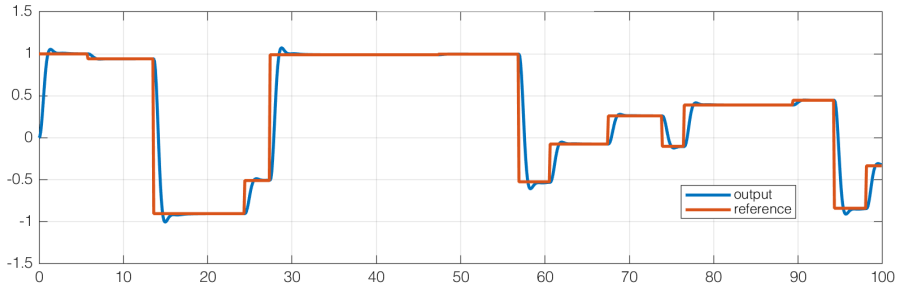
MPC CALIBRATION PROBLEM

- The design depends on a vector x of **MPC parameters**
- MPC parameters are intuitive to set (e.g., weights)
- Still, can we auto-calibrate them ?

- 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**

$$\min_x f(x)$$

AUTO-TUNING - GLOBAL OPTIMIZATION ALGORITHMS

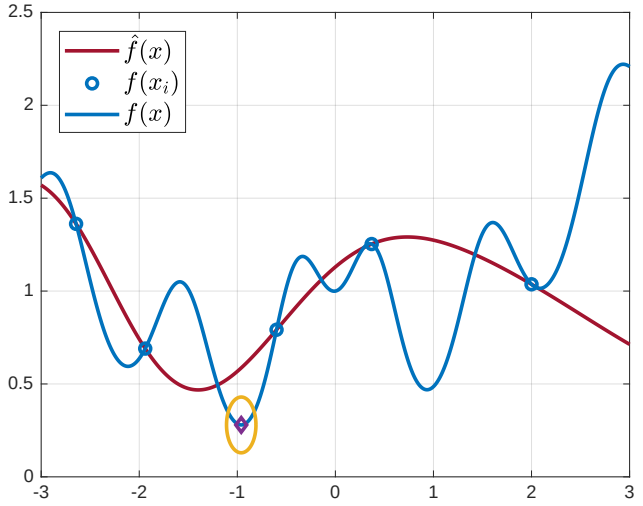
- Several derivative-free global optimization algorithms exist: (Rios, Sahidinis, 2013)
  - Lipschitzian-based partitioning techniques:
    - **DIRECT** (DIvide in RECTangles) (Jones, 2001)
    - Multilevel Coordinate Search (**MCS**) (Huyer, Neumaier, 1999)
  - Response surface methods
    - **Kriging** (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

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & \ell \le x \le u \\ & g(x) \le 0 \end{aligned}$$



- **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

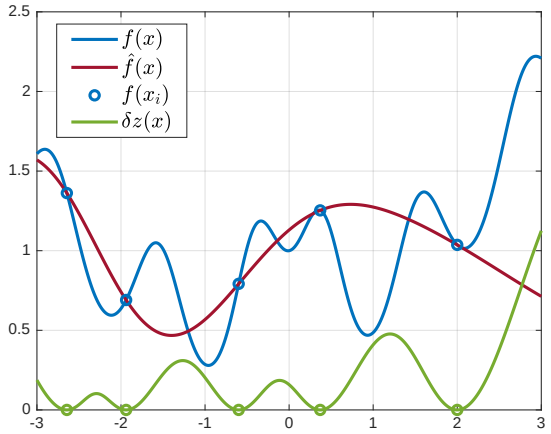
- **Step #2**: construct the **IDW exploration function**

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

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

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

ΔF = observed range of $f(x_i)$



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

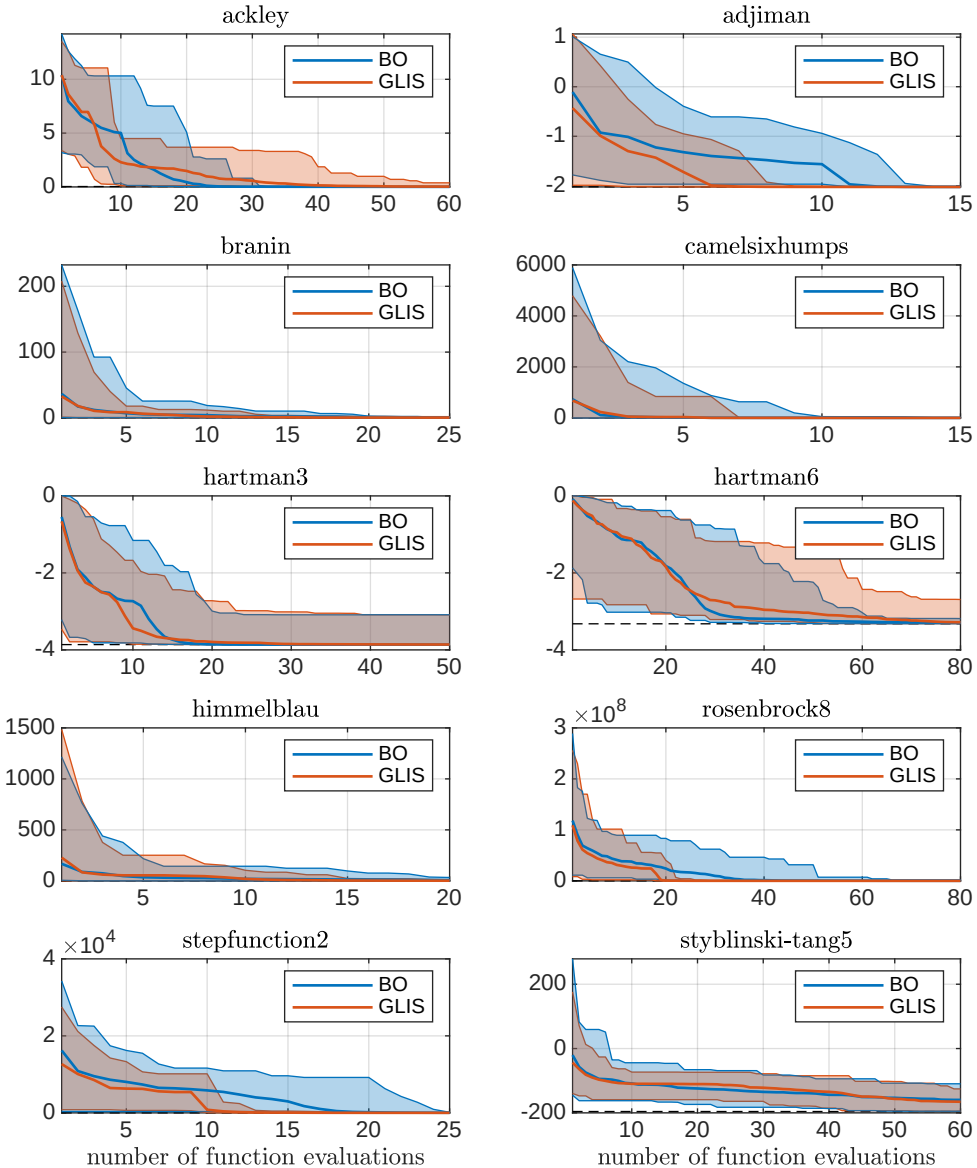
$$x_{N+1} = \arg\min \ \hat{f}(x) - \delta z(x) \quad \text{s.t.} \ \ell \le x \le u,\ g(x) \le 0$$

to get new sample x_{N+1}

δ= *exploitation* vs *exploration* tradeoff parameter

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

GLIS VS BAYESIAN OPTIMIZATION



problem	n	BO [s]	GLIS [s]
ackley	2	29.39	3.13
adjiman	2	3.29	0.68
branin	2	9.66	1.17
camelsixhumps	2	4.82	0.62
hartman3	3	26.27	3.35
hartman6	6	54.37	8.80
himmelblau	2	7.40	0.90
rosenbrock8	8	63.09	13.73
stepfunction2	4	11.72	1.81
styblinski-tang5	5	37.02	6.10

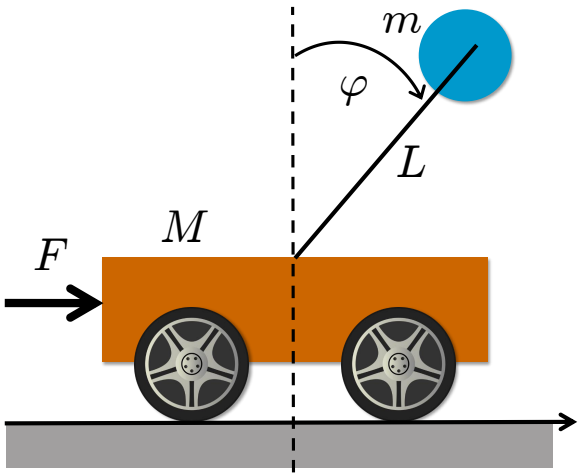
Results computed on 20 runs per test

BO = MATLAB's **bayesopt** fcn

MPC AUTOTUNING EXAMPLE

(Forgione, Piga, Bemporad, 2020)

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



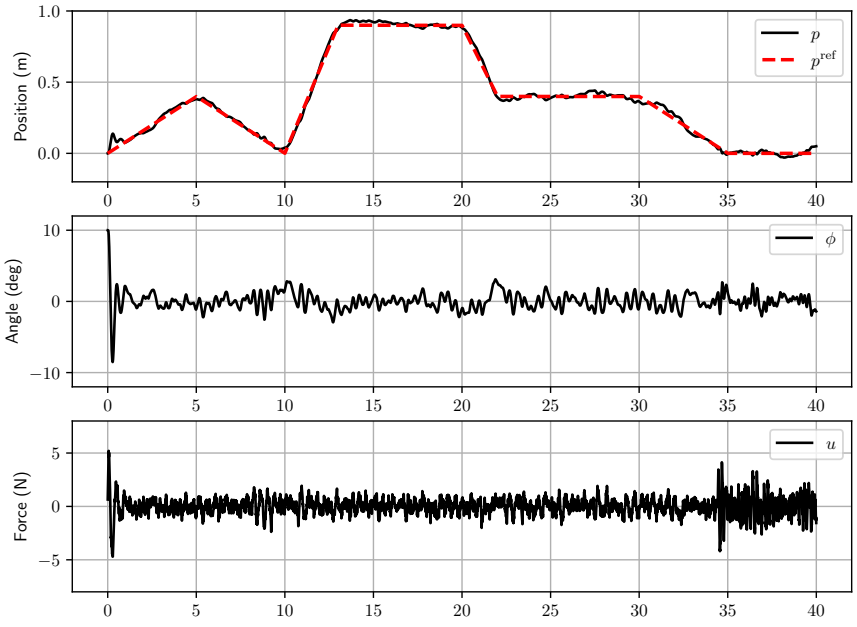
  - **sample time**
  - **weights** on outputs and input increments
  - prediction and control **horizons**
  - **covariance** matrices of Kalman filter
  - absolute and relative **tolerances** of QP solver

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

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

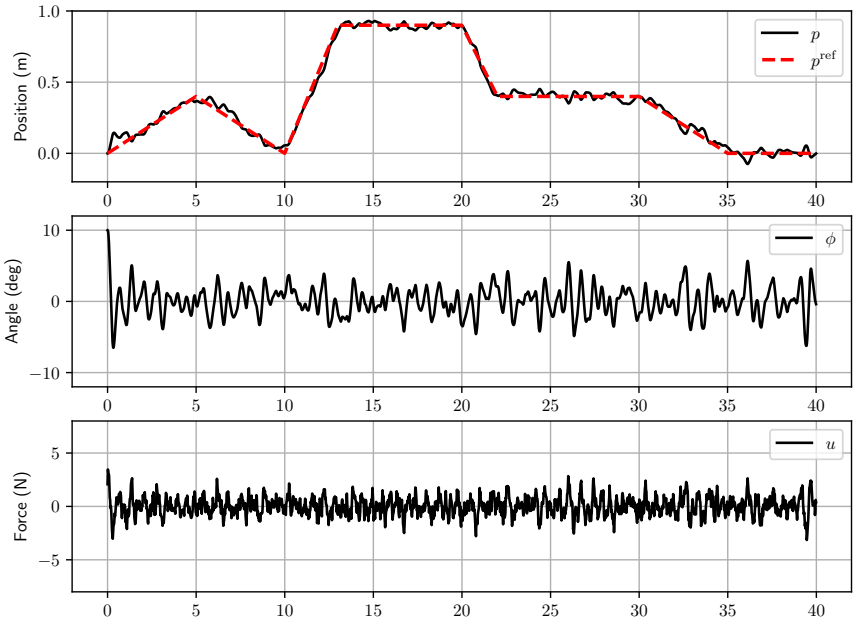
MPC AUTOTUNING EXAMPLE

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
- MPC parameters tuned by **GLIS** global optimizer
- 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:

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

- 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

SEMI-AUTOMATIC TUNING BY PREFERENCE-BASED LEARNING

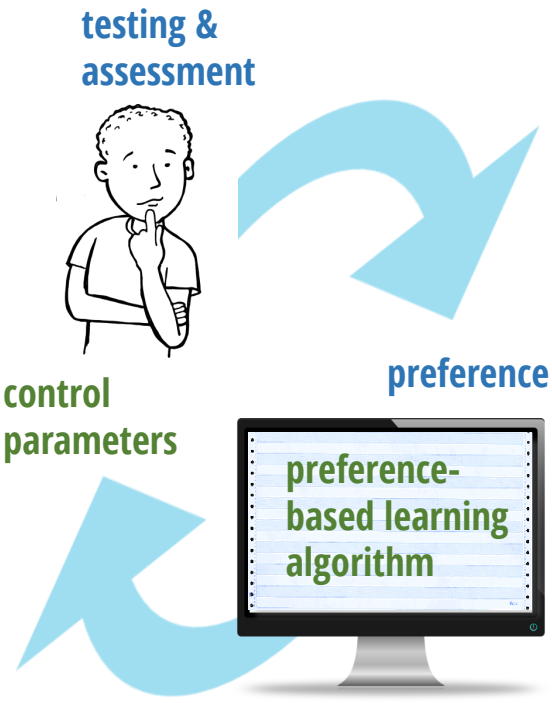
- Use **preference-based optimization** (**GLISp**) algorithm for **semi-automatic tuning** of MPC (Zhu, Bemporad, Piga, 2021) → **WeB06.1**

- 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**

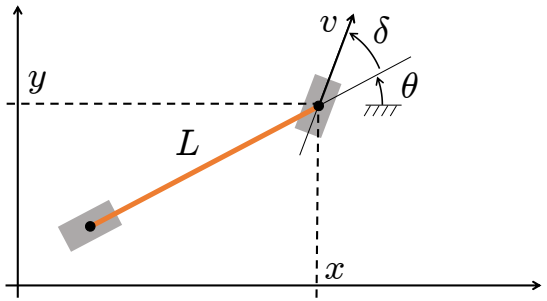


PREFERENCE-BASED TUNING: MPC EXAMPLE

(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:

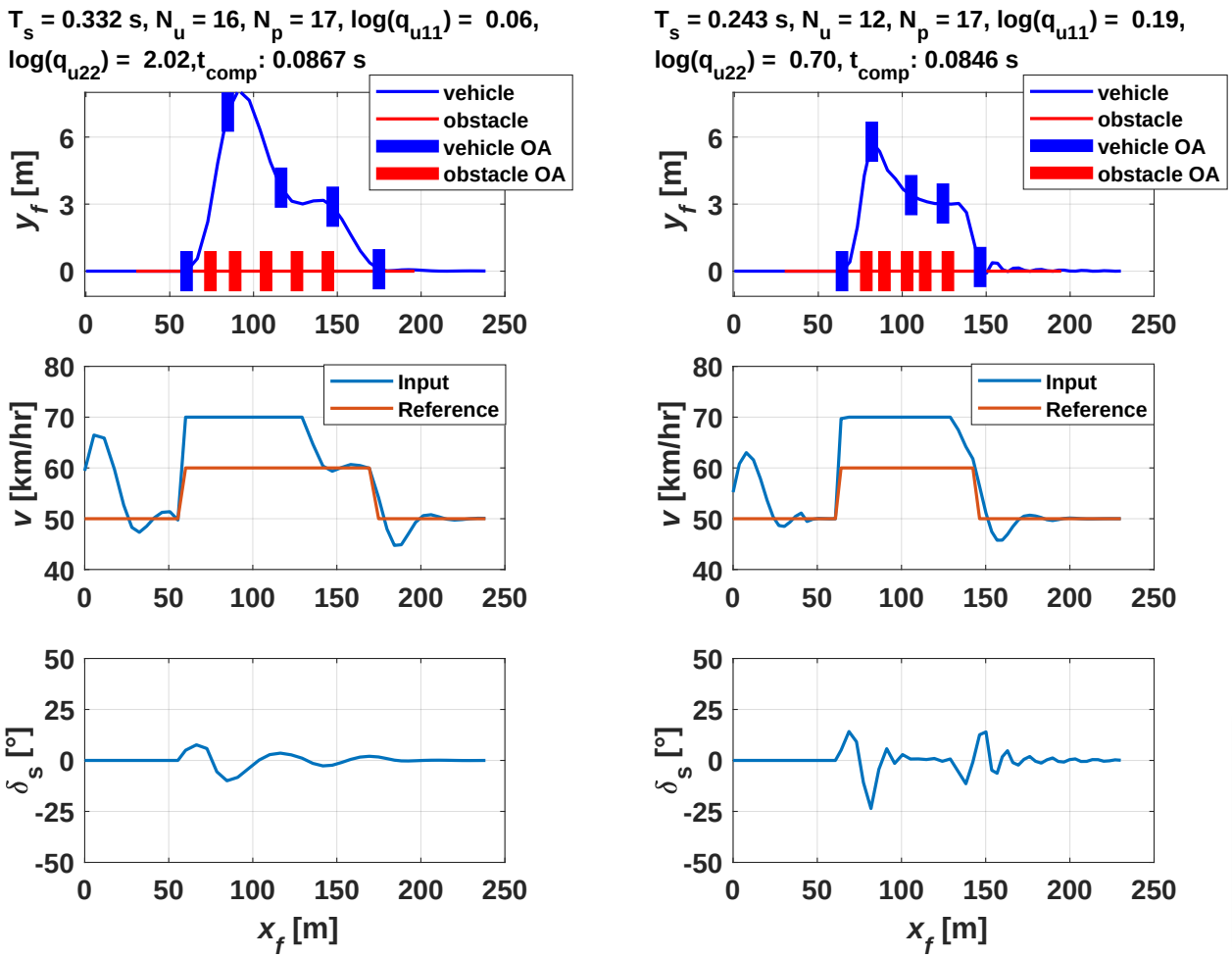
  "*optimal obstacle avoidance*", "*pleasant drive*", "*keep CPU time small*", …

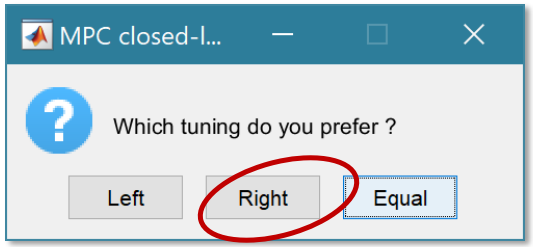
  → **not easy to quantify in a single function**

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

PREFERENCE-BASED TUNING: MPC EXAMPLE

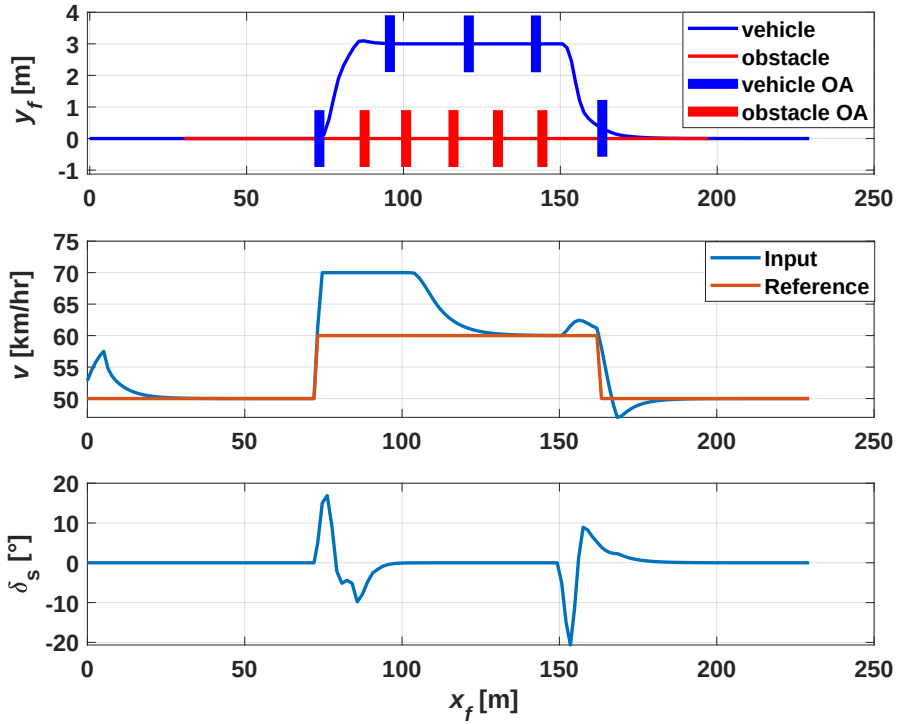
- Preference query window:





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!
- Extended to handle also **unknown constraints** (Zhu, Piga, Bemporad, 2021)

CONCLUSIONS

- Numerical methods have steadily revolutionized control design over the years (linear algebra, LMI's, embedded optimization, ...)
- Now it's machine learning's turn
- **Machine learning** offers several tools for MPC design:
  - to get nonlinear and hybrid **prediction models** from **data**
  - to **reduce on-line computations**
  - to automatically (or semi-automatically) **calibrate** the control law
- A wide spectrum of research opportunities and new practices is open !