MACHINE LEARNING: A NEW ICE AGE ?

(IDENTIFICATION, CONTROL, ESTIMATION)

Alberto Bemporad

imt.lu/ab

IMT SCHOOL FOR ADVANCED STUDIES LUCCA

IFAC 2020 Workshop “Machine Learning meets Model-based Control”

A TIMELINE OF CONTROL ENGINEERING

1900

mechanical design

complex analysis

1930-1950

frequency domain

Bode, Nyquist
root locus

robust control

linear algebra

1960-1970

pole-placement
LQR

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

Kalman filtering

state-space

1970

functional analysis

Lyapunov methods

nonlinear control

1980

semidefinite programming

LMI-based methods

stability analysis

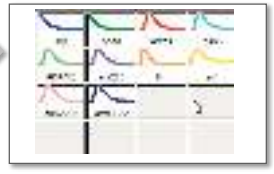
feedback synthesis

robust control

statistics

>1990

system identification



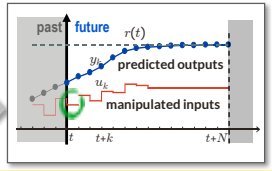
machine learning (ML)

?

>2020

numerical optimization

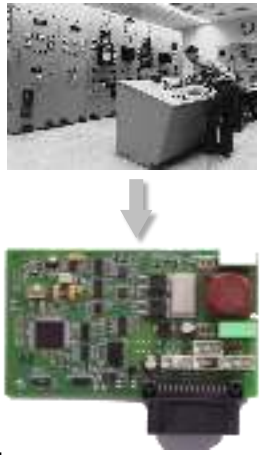
>1990



model predictive control (MPC)

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

MODEL PREDICTIVE CONTROL (MPC)



- Long history of success of MPC in the **process industries**, now spreading in the **automotive industry** (and many others):
  - multivariable, linear/nonlinear/stochastic systems w/ **constraints**
  - intuitive to **design** and **calibrate**, easy to **reconfigure**
  - great **tools for MPC design** ( ) and **deployment** (ODYS Advanced Controls & Optimization) exist
    (Bemporad, Ricker, Morari, 1998-today) (ODYS Srl, 2013-present)

- 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

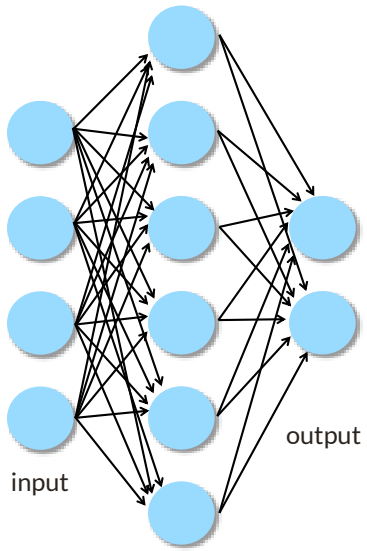
`https://www.odys.it/odys-and-gm-bring-online-mpc-to-production`

MACHINE LEARNING (ML)

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

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

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

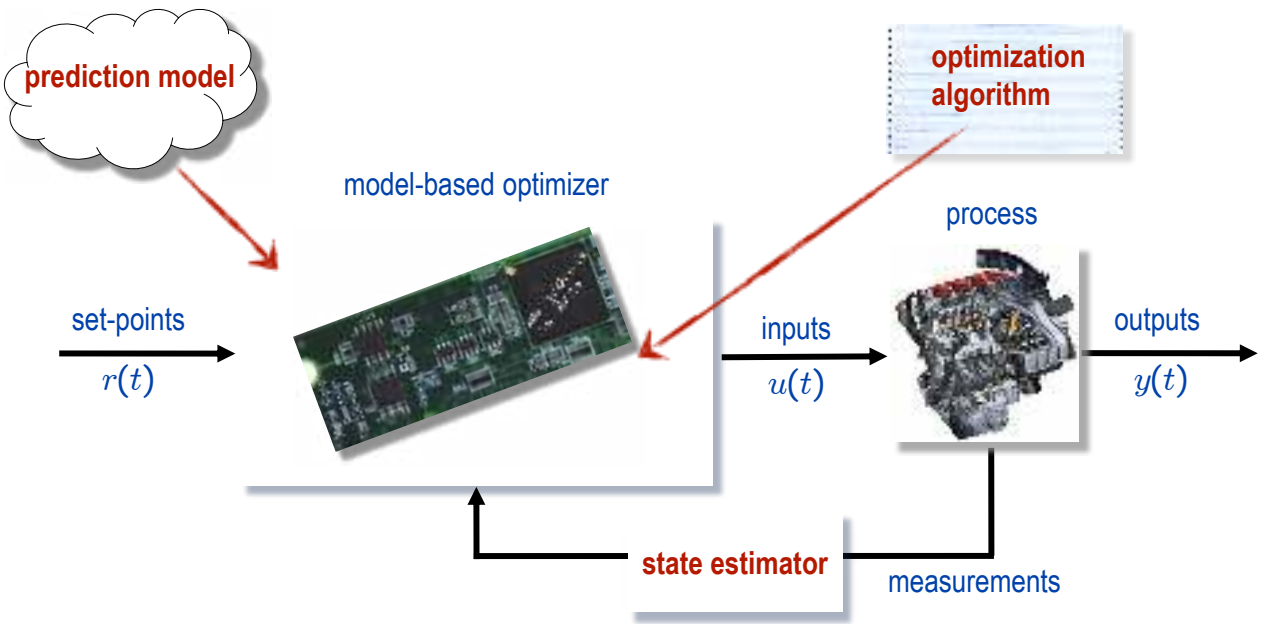


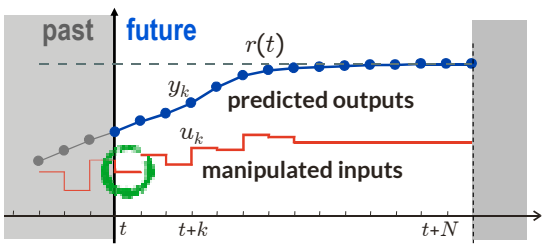
- Used in myriads of most **diverse application domains**

- Availability of excellent open-source **software tools** exist like `scikit-learn`, `Keras/TensorFlow` also explains success

ML FOR MPC

- How can ML be useful in MPC:
  - **Identification** = learn the **prediction model** from data
  - **Control** = learn the MPC **control law** from data
  - **Estimation** = learn how to reconstruct **unmeasured signals** from data





OUTLINE OF MY TALK

- **Identification**
  - Black-box identification of state-space models using **autoencoders**
  - Learning the entire MPC prediction from data
- **Control**
  - Reinforcement learning (direct policy search)
  - Automatic and semi-automatic **calibration** of MPC
- **Estimation**
  - Learning **virtual sensors** / **state observers** from data

LEARNING PREDICTION MODELS FOR MPC

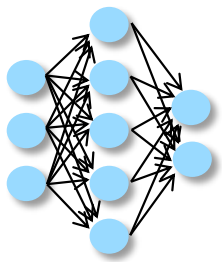
PREDICTION MODELS FOR MPC

- **Physics-based** nonlinear models are often too complex

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

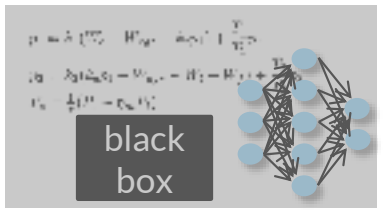
- Use **black-box system identification** algorithms to fit linear or nonlinear models to data

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



- **Jacobians** of prediction models are required

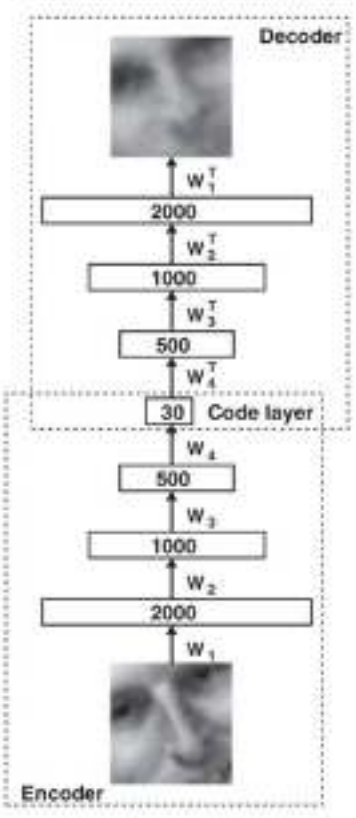
- **Computation complexity** depends on chosen model, need to trade off **descriptiveness vs simplicity** of the model



LEARNING NONLINEAR STATE-SPACE MODELS FOR MPC

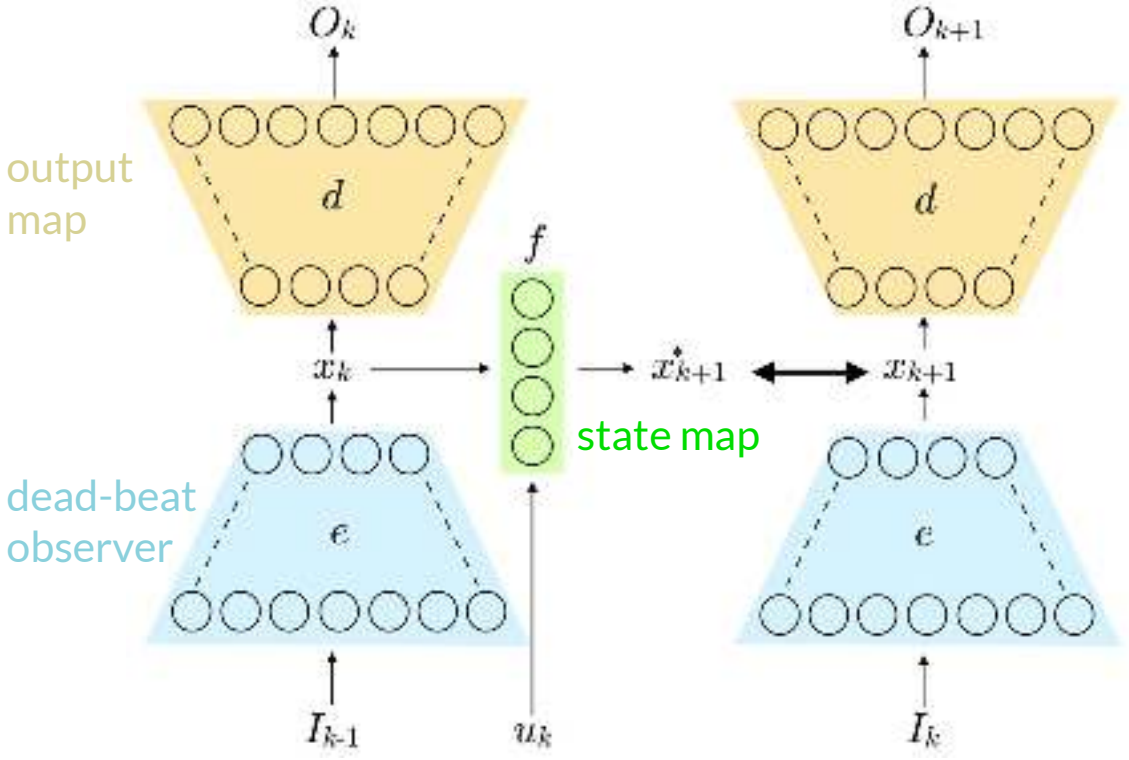
(Masti, Bemporad, 2018)

- **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)



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

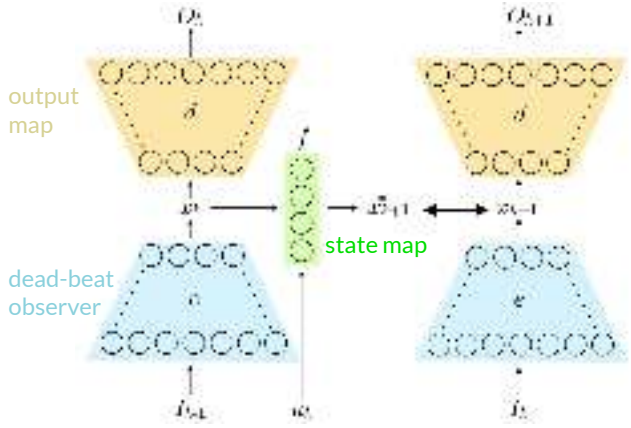
$$I_k = [y'_k \ \dots \ y'_{k-n_a+1} \ u'_k \ \dots \ u'_{k-n_b+1}]'$$

LEARNING NONLINEAR STATE-SPACE MODELS FOR MPC

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

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

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



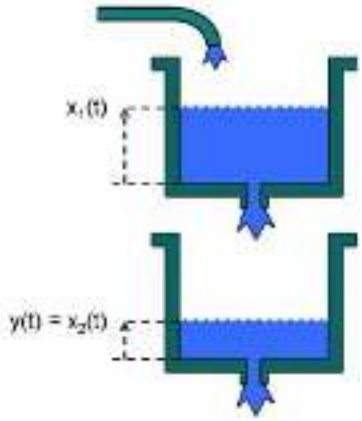
- Model complexity reduction: add **group-LASSO** penalties on subsets of weights

- **Quasi-LPV** structure for MPC: set $f(x_k, u_k) = A(x_k, u_k) \begin{bmatrix} x_k \\ 1 \end{bmatrix} + B(x_k, u_k) u_k$, $y_k = C(x_k, u_k) \begin{bmatrix} x_k \\ 1 \end{bmatrix}$
  (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

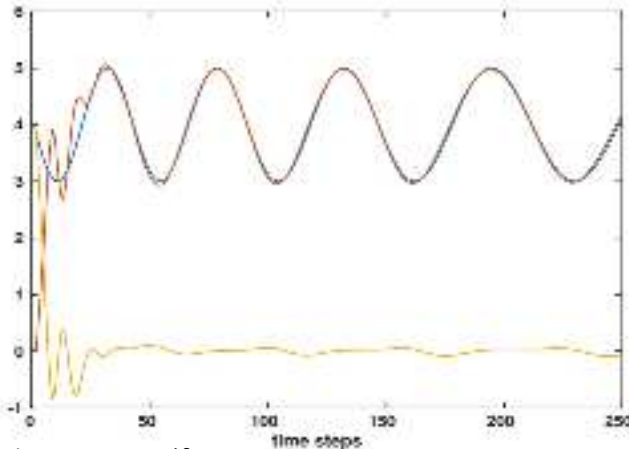


www.mathworks.com

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

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



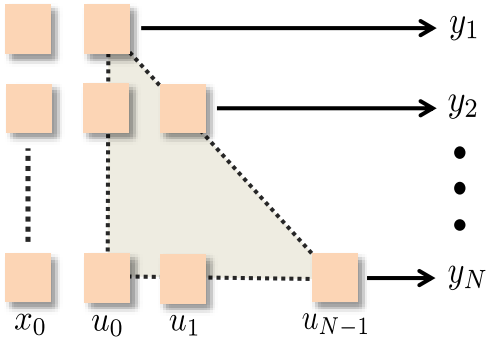
LTV-MPC results

LEARNING AFFINE NEURAL PREDICTORS FOR MPC

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

- Alternative: **learn the entire prediction**

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



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

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

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

- Avoid computing Jacobians by fitting h_k in the affine form

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

cf. (Liu, Kadirkamanathan, 1998)

LEARNING AFFINE NEURAL PREDICTORS FOR MPC

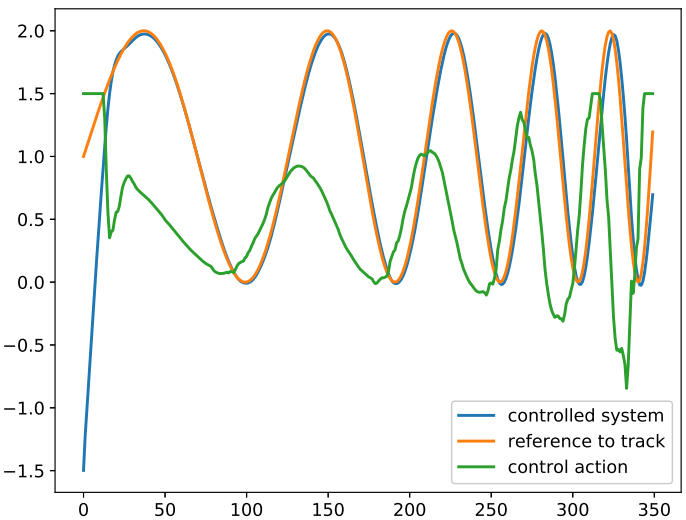
- **Example**: apply **affine neural predictor** to nonlinear two-tank benchmark problem

  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:

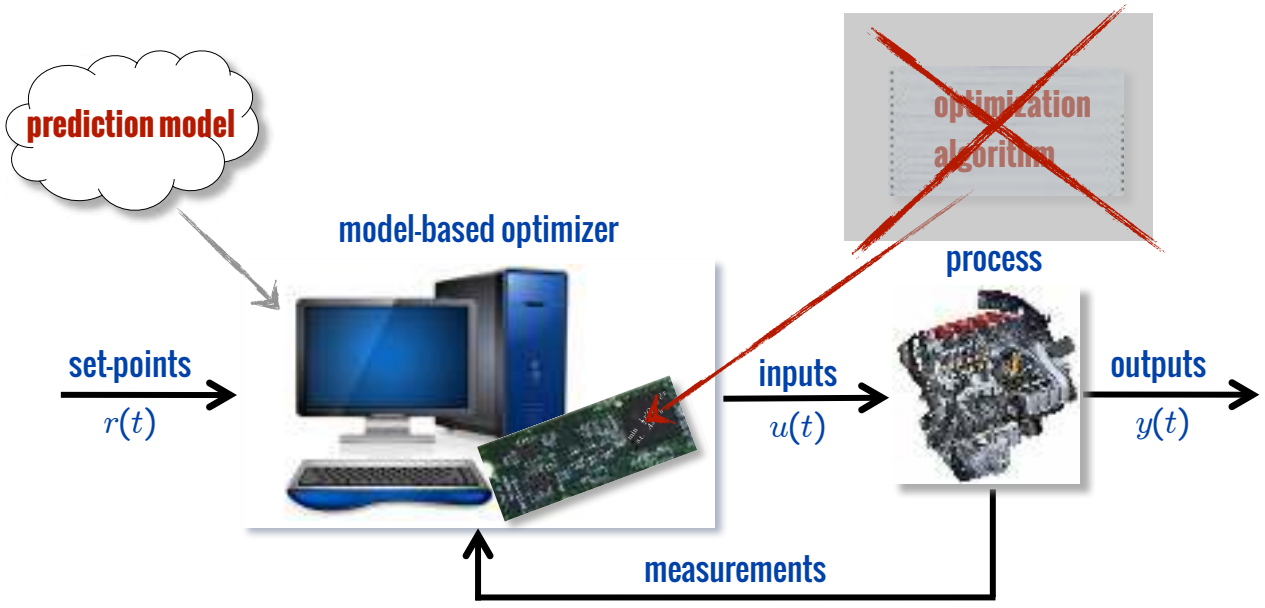


- Model complexity reduction:
  add **group-LASSO** term with penalty λ

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

LEARNING APPROXIMATE MPC LAWS

MPC WITHOUT ON-LINE QP



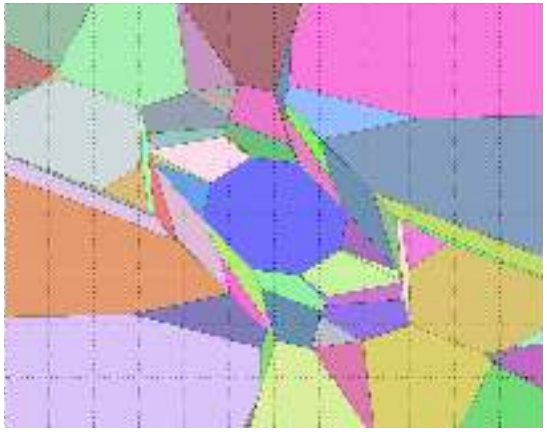
- Can we implement MPC **without on-line optimization** ?
- If model / constraints are linear, and model / constraints / quadratic cost are **time-invariant**:

YES ! (Bemporad, Morari, Dua, Pistikopoulos, 2002)

EXPLICIT MODEL PREDICTIVE CONTROL

- **Explicit MPC**: **continuous** and **piecewise affine** control law

$$u_0^*(x) = \begin{cases} F_1 x + g_1 & \text{if} \quad H_1 x \leq K_1 \\ \vdots & \vdots \\ F_M x + g_M & \text{if} \quad H_M x \leq K_M \end{cases}$$



- Only limited to **small MPC problems** with **time-invariant linear/hybrid** models, linear/quadratic costs, linear constraints

- Approximate explicit MPC solutions are possible to simplify the control law

(Alessio, Bemporad, 2008) (Johansen, Grancharova, 2003) (Christophersen, Zeilinger, Jones, Morari, 2007)

APPROXIMATE MPC LAWS

- Use any **function regression** technique 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
  - Check performance / feasibility/ prove closed-loop stability (if possible)

- Possible function regression approaches:
  - **Lookup tables** (linear interpolation, inverse distance weighting, ...)
  - **Neural networks** (Parisini, Zoppoli, 1995) (Karg, Lucia, 2018)
  - **Hybrid system identification / PWA regression** (Breschi, Piga, Bemporad, 2016)

- **Semi-explicit MPC**: use **binary classification** methods to learn the optimal
  - **binary variables** solving parametric MIQP/LP, $\delta^* = \delta(x)$, then solve QP/LP online
    (Masti, Bemporad, 2019) (Masti, Pippia, Bemporad, De Schutter, IFAC 2020)
  - **active set** of parametric QP for warm start (Klauco, Kalúz, Kvasnica, 2019)

LEARNING THE CONTROL LAW DIRECTLY FROM DATA

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)**

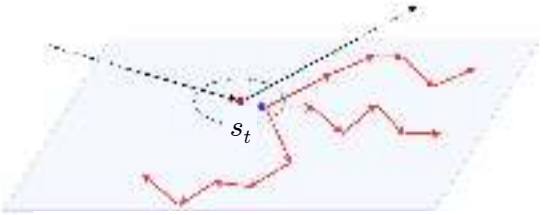
$$K_t \leftarrow K_{t-1} - \alpha_t \mathcal{D}(K_{t-1})$$

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

$$K_t^\star = \underset{K}{\text{argmin}} \|K - K_t^s\|_2^2$$
$$\text{s.t. } K \text{ stabilizes local linear model}$$

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:**

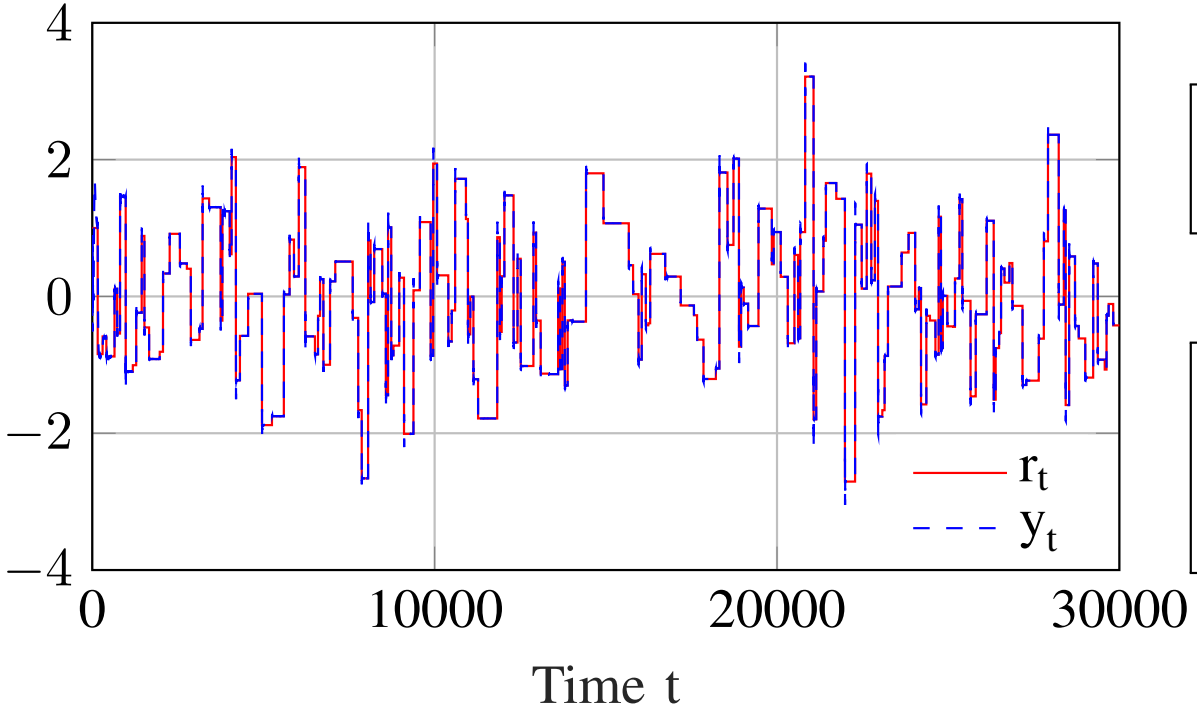
$$\pi_K(s_t,\, r_t) = -K^s \cdot s_t - K^r \cdot r_t, \qquad K = \begin{bmatrix} K^s \\ K^r \end{bmatrix}$$

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, $d_t = 0$):



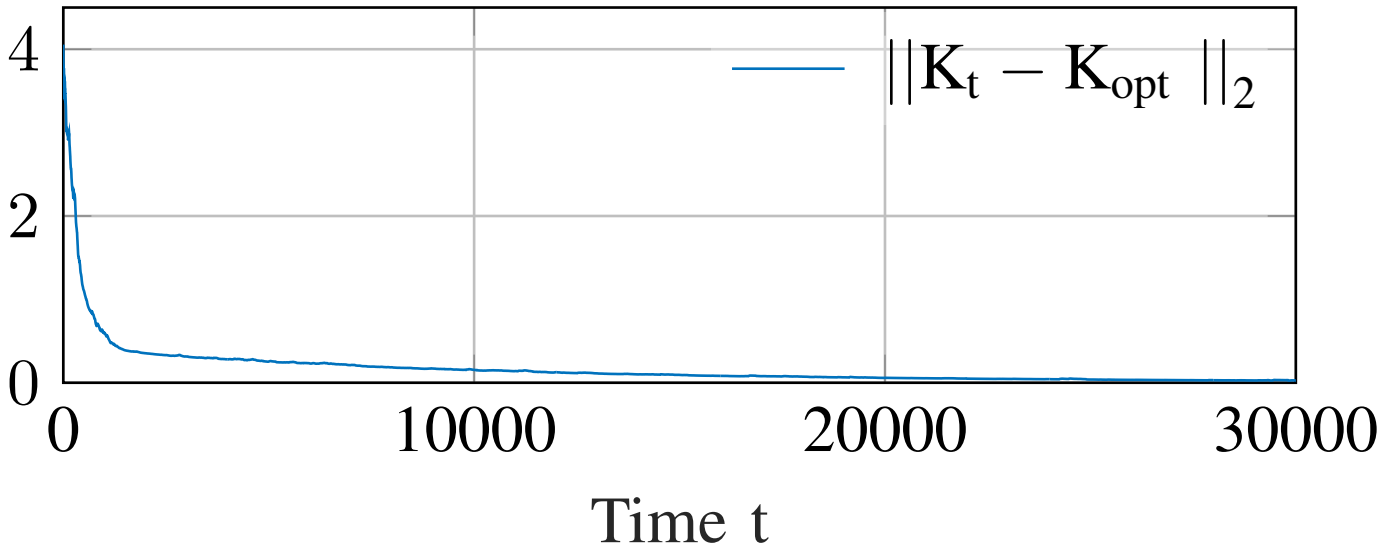
$$\begin{aligned} Q_y &= 1 \\ R &= 0.1 \\ Q_q &= 1 \end{aligned}$$

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 $\|K_t - K_{opt}\|_2$:



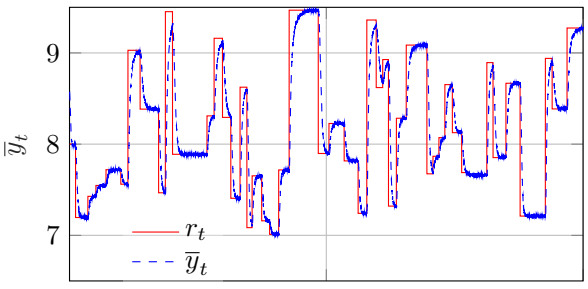
$$K_{\text{SGD}} = [-1.255, 0.218, 0.652, 0.895, 0.050, 1.115, -2.186]$$

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

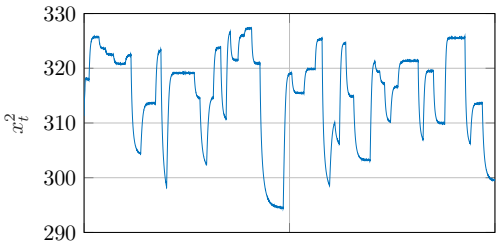
NONLINEAR EXAMPLE

Online learning

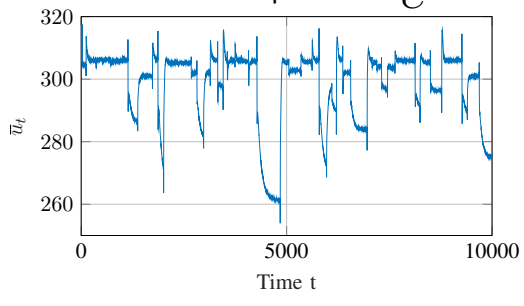
concentration C_A and reference r_t



temperature T



coolant temperature T_C

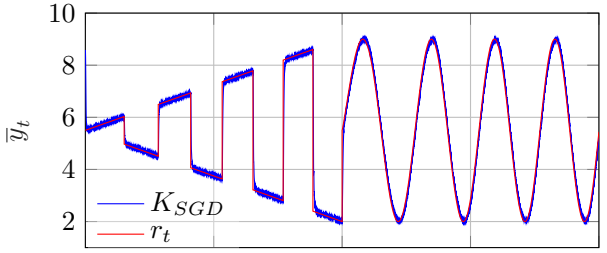


n_i	n_o	L
2	3	10

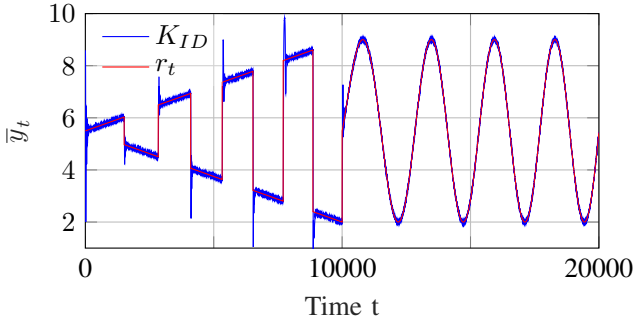
N_0	N_r	N_q
50	20	20

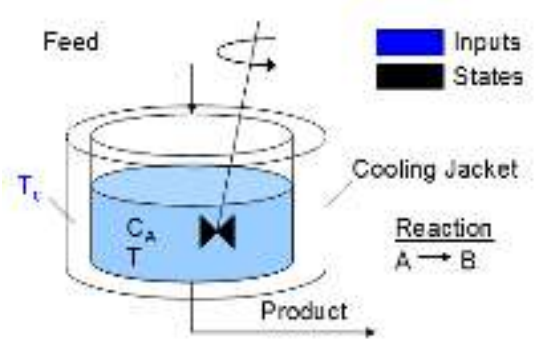
Validation phase

Cost of $\mathbf{K_{SGD} = 4.3 \cdot 10^3}$



Cost of $\mathbf{K_{ID} = 2.4 \cdot 10^4}$





Continuously Stirred Tank Reactor (CSTR)

(courtesy: apmonitor.com)

SGD beats SYS-ID + LQR

- Approach currently extended to **switching-linear** and **nonlinear policies**

(Ferrarotti, Bemporad, IFAC 2020) (Ferrarotti, Bemporad, 2020)

LEARNING OPTIMAL MPC TUNING

MPC CALIBRATION PROBLEM

- Controller depends on a vector x of parameters

- Parameters can be many things:
  - MPC weights, prediction model coefficients, horizons
  - Entries of covariance matrices in Kalman filter
  - 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 that solves 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 ∇f of $f(x)$ (**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)

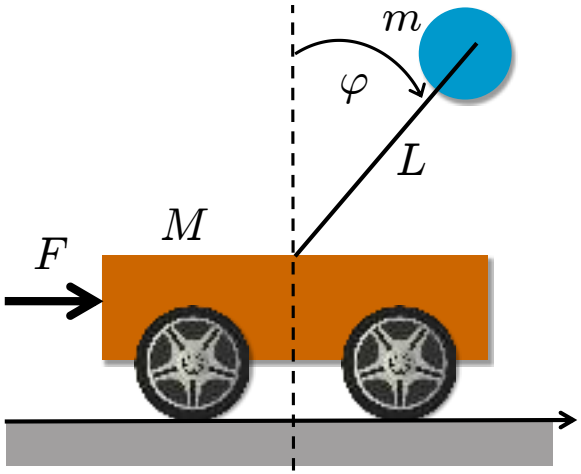
AUTO-TUNING - GLOBAL OPTIMIZATION ALGORITHMS

- 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, 2019) `cse.lab.imtlucca.it/~bemporad/glis`

MPC AUTOTUNING EXAMPLE

(Forgione, Piga, Bemporad, IFAC 2020)

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

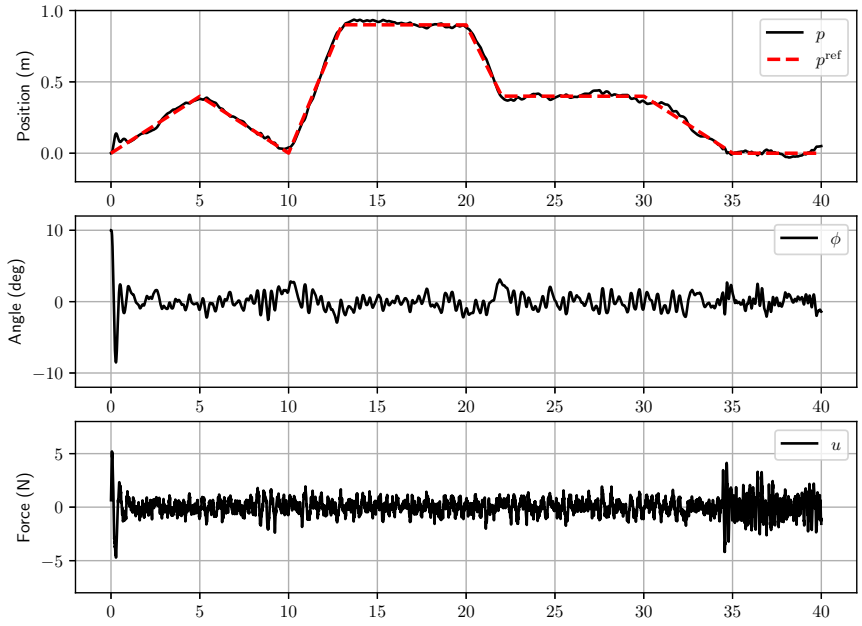


  - sample time
  - weights on outputs and input increments
  - prediction and control horizons
  - covariance matrices of Kalman filter
  - absolute and relative tol of QP solver

- 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)

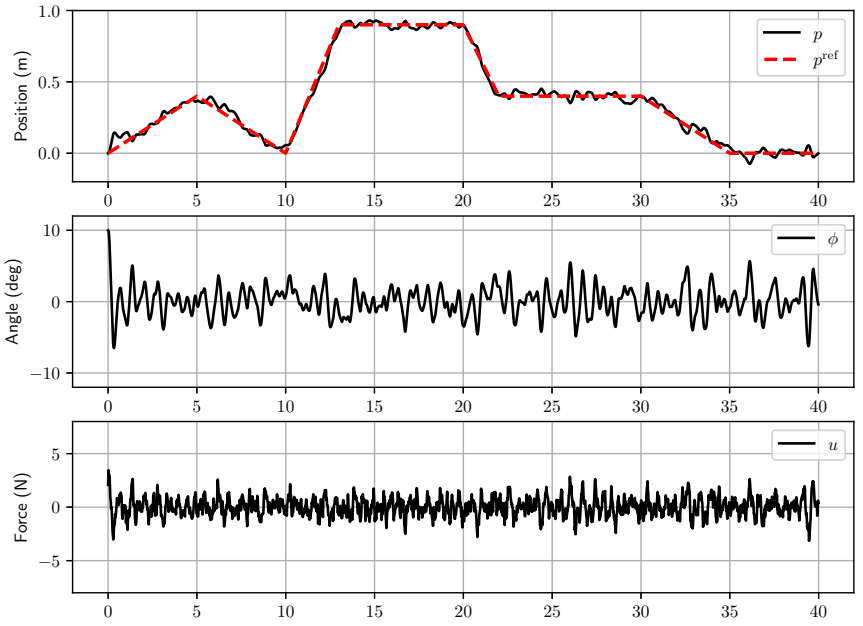
MPC AUTOTUNING EXAMPLE

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, ...)

    (Piga, Forgione, Formentin, Bemporad, 2019) (Forgione, Piga, Bemporad, IFAC 2020)
  - 👍 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 objectives are multiple, not clear how to blend them in a **single** one
- Current research: **preference-based optimization** (**GLISp**), having human assessments in the loop (**semi-automatic tuning**)

  (Bemporad, Piga, 2019)

  (Zhu, Bemporad, Piga, 2020)

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

ESTIMATION: LEARNING VIRTUAL SENSORS

LEARNING VIRTUAL SENSORS

(Masti, Bernardini, Bemporad, 2019)

- **unknown** model of a dynamical system

$$\begin{aligned} x_{k+1} &= f(x_k, u_k, \rho_k) \\ y_k &= g(x_k, \rho_k) \\ \rho_{k+1} &= h(\rho_k, k, u_k) \end{aligned}$$

$x \in \mathbb{R}^{\bar{n}_x}$ **state vector**
$u \in \mathbb{R}^{n_u}$ **command input**
$y \in \mathbb{R}^{n_y}$ **output vector**
$\rho \in \mathbb{R}^{S}$ **signal to estimate**

- ρ_k can model equipment wear/component drift, faults, ... or unmeasured states of the system

- Special cases:
  - **linear parameter-varying (LPV) systems**

$$\begin{aligned} x_{k+1} &= A(\rho_k)x_k + B(\rho_k)u_k \\ y_k &= C(\rho_k)x_k + D(\rho_k)u_k \end{aligned}$$

  - **switched affine systems** $\rho_k \in \{1, \dots, s\}$

(aa1car.com)

LEARNING VIRTUAL SENSORS

- **Assumption #1**: ρ_k cannot be measured at runtime
- **Assumption #2**: ρ_k is available in training data
- **Assumption #3**: we do not know model f, g, h

$$\begin{aligned} x_{k+1} &= f(x_k, u_k, \rho_k) \\ y_k &= g(x_k, \rho_k) \\ \rho_{k+1} &= h(\rho_k, k, u_k) \end{aligned}$$

Goal: estimate ρ_k from input/output data at runtime

- Applications:
  - onboard **diagnosis**, anomaly/**fault detection and isolation**, **predictive maintenance**
  - **gain scheduling control**

- Existing approaches to estimate ρ_k are mostly **model-based**:

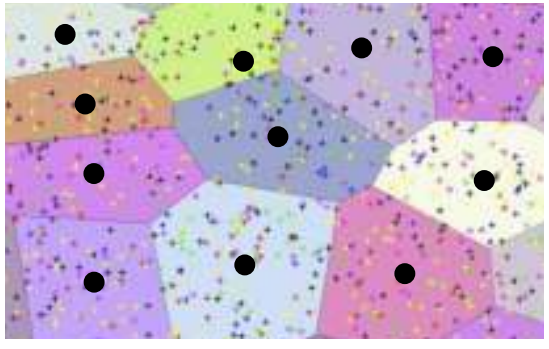
  extended Kalman-filtering, moving horizon estimation, interacting multiple model (IMM) estimator, ...

LEARNING VIRTUAL SENSORS - DESIGN

- $D = \{u_k, y_k, \rho_k\}$, $k = 1, \ldots, K$, = **training dataset** (acquired off line)
- For each k estimate a **local linear model** using recursive SYS-ID, γ_k = vector of model parameters. Example:

$$y_k = -\sum_{i=1}^{n_a} a_i y_{k-i} + \sum_{i=1}^{n_b} b_i u_{k-i}, \qquad \gamma = [a_1 \ \ldots \ a_{n_a} \ b_1 \ \ldots \ b_{n_b}]'$$

- Use **unsupervised learning** to partition the set $\{\gamma_{k_1}, \ldots, \gamma_K\}$ in N **clusters** (more generally: partition the set of pairs (ρ_k, γ_k))



γ-space

- Identify N **linear time-invariant models** Σ_j for each cluster ($j = 1, \ldots, N$)
- Design a **linear observer** for each model Σ_j to summarize all past input/outputs into state estimates $x^1, x^2, \ldots, x^N$ (one per model)

LEARNING VIRTUAL SENSORS - DATA COMPRESSION AND PREDICTOR

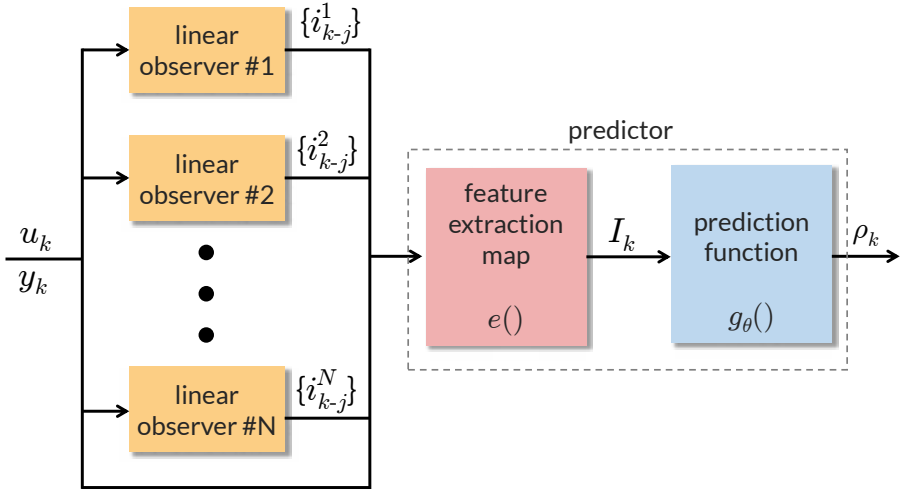
- Compress past input/outputs into sum of output prediction errors

$$\nu_k^j = \sum_{r=k-\ell}^{k} (y_r^j - \hat{y}_r^j)^2$$

ℓ = window of past estimates (hyperparameter)

- Two-layer structure

$$\begin{aligned} I_k &= (u_k, y_k, \nu_k^1, \ldots, \nu_k^N) \\ \hat{\rho}_k &= g_\theta(I_k) \end{aligned}$$



- g_θ = feedforward **artificial neural network (ANN)** w/ rectified linear unit (ReLU) activation function, or **decision tree (DT)**, or **random forest (RF)** regression

- Training objective: $\min \sum_k \|\hat{\rho}_k - \rho_k\|_2^2$

EXAMPLE: NONLINEAR PARAMETER-VARYING SYSTEM

- True (unknown) underlying nonlinear system ($\alpha = 1$)

$$\begin{cases} x_{k+1} &= Ax_k + \frac{\alpha}{2}\operatorname{atan}(x_k) + \log(\rho_k + 1)Bu_k \\ y_k &= -(1 + e^{\rho_k})\begin{bmatrix} 0 & 0 & 0 & 0 & 1 \end{bmatrix} x_k \end{cases} \qquad \begin{aligned} A &\in \mathbb{R}^{5\times 5} \\ B &\in \mathbb{R}^{5\times 2} \end{aligned}$$

- $u_k \sim \mathcal{N}(0, 1)$, the scheduling signal ρ_k is generated by setting

$$\begin{aligned} p_k &= 0.999\rho_k + 0.03\omega_k, \qquad \omega_k \sim \mathcal{N}(0, 1) \\ \rho_{k+1} &= \begin{cases} p_k \text{ if } p_k \in [-0.95, 0.95] \\ \frac{p_k}{2} \text{ otherwise} \end{cases} \end{aligned}$$

- Input, output, and ρ measurements are affected by noise in $\mathcal{N}(0, 0.03^2)$

- Training dataset = up to 25,000 samples, testing data set = 5,000 samples

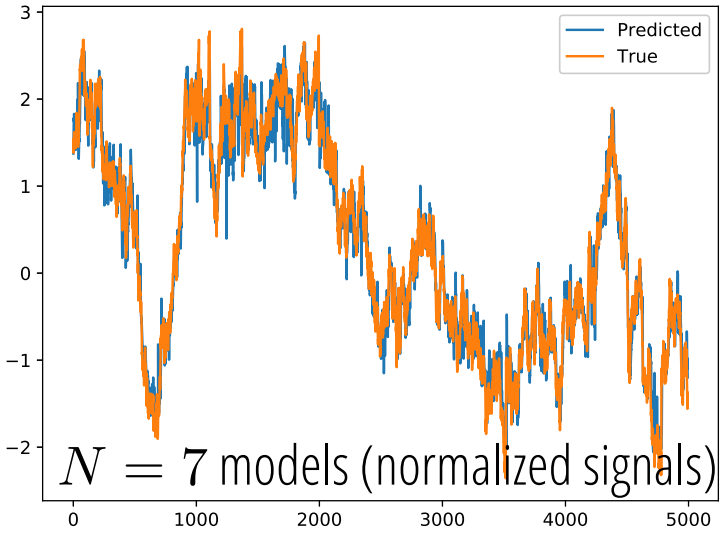
EXAMPLE: NONLINEAR PARAMETER-VARYING SYSTEM

- Local models identified by **recursive ARX estimator** based on Kalman filtering (**3 past outputs, 3 past inputs** $\Rightarrow \gamma_k \in \mathbb{R}^6$)
- N clusters created by running **K-means** in γ-space
- **Deadbeat observer** for each centroidal model $\Sigma(\bar{\gamma}_j), j = 1, \ldots, N$ (equivalent to $\hat{y}_k^j = \bar{[}\hat{y}_{k-1}\ \hat{y}_{k-2}\ \hat{y}_{k-3}\ u_{k-1}\ u_{k-2}\ u_{k-3}]'\gamma_j$)
- Input to ANN is $I_k = (u_k, y_k, \nu_k^1, \ldots, \nu_k^N), \nu_k^i = \sum_{r=k-4}^{k} (y_r^i - \hat{y}_r^i)^2 \quad (\ell = 4)$
- **ANN** with 2 ReLU layers, 64 neurons + linear output function
- Alternative: **DT** or **RF**, both with max depth = 10 nodes

EXAMPLE: NONLINEAR PARAMETER-VARYING SYSTEM

- Quality of reconstruction $\hat{\rho}$ of ρ is measured by

$$\mathrm{FIT} = 1 - \frac{\|\rho_k - \hat{\rho}_k\|_2}{\|\rho_k - \bar{\rho}_k\|_2}$$



- Experiments are repeated 10 times with different ρ and noise realizations

# models	2	3	5	7
mean (std)	0.686 (0.033)	0.766 (0.027)	0.779 (0.026)	0.784 (0.027)

(RF predictor)

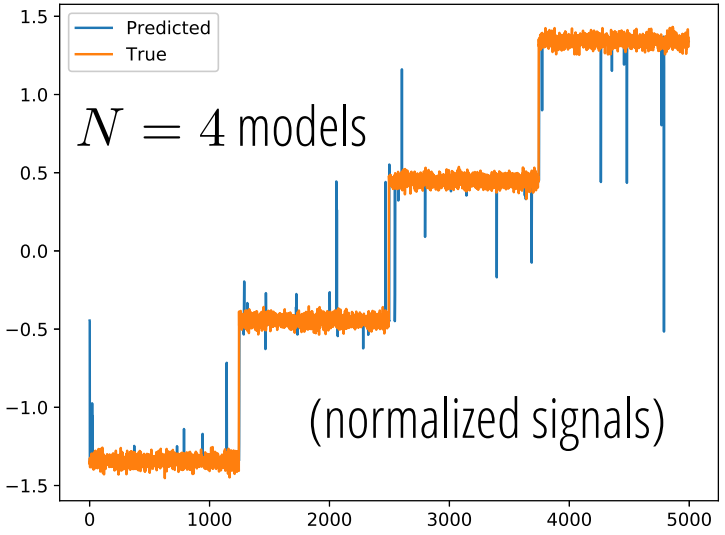
# training samples	5 models DT	RF	ANN
25000	0.757 (0.026)	0.779 (0.026)	0.778 (0.026)
15000	0.733 (0.031)	0.762 (0.028)	0.763 (0.031)
5000	0.613 (0.191)	0.682 (0.150)	0.685 (0.090)

EXAMPLE: MODE RECONSTRUCTION

- True (unknown) **switching** linear system ($\alpha = 0$) with 4 modes

$$\rho_k \in \{0, 0.5, 1, 1.5\} \quad \textbf{(discrete)}$$

- Experiments are repeated 10 times with different ρ and noise realizations



# models	2	3	4	5
mean (std)	0.773 (0.018)	0.876 (0.012)	0.931 (0.009)	0.931 (0.009)

- **Note**: no penalty on ρ_k switching used.
  Cf. Hidden Markov Models and Jump Models (Bemporad, Breschi, Piga, Boyd, 2018)

- **Note**: a **classifier** rather than a function regressor may improve fit quality

EXAMPLE: NONLINEAR STATE ESTIMATION (BATTERY SOC)

- Lithium-ion battery model (**unknown**) (Ali et al., 2017)

$$\begin{cases} \dot{x}_1(t) &= \dfrac{-i(t)}{C_c} \\ \dot{x}_2(t) &= \dfrac{-x_2(t)}{R_{ts}(x_1)C_{ts}(x_1)} + \dfrac{i(t)}{C_{ts}(x_1)} \\ \dot{x}_3(t) &= \dfrac{-x_3(t)}{R_{tl}(x_1)C_{tl}(x_1)} + \dfrac{i(t)}{C_{ts}(x_1)} \\ y(t) &= E_0(x_1) - x_2(t) - x_3(t) - i(t)R_s(x_1) \end{cases}$$

renewableenergyworld.com

- Only voltage $y(t)$ and current $i(t)$ are measurable

- **Goal**: estimate the state of charge (SoC) $x_1(t)$

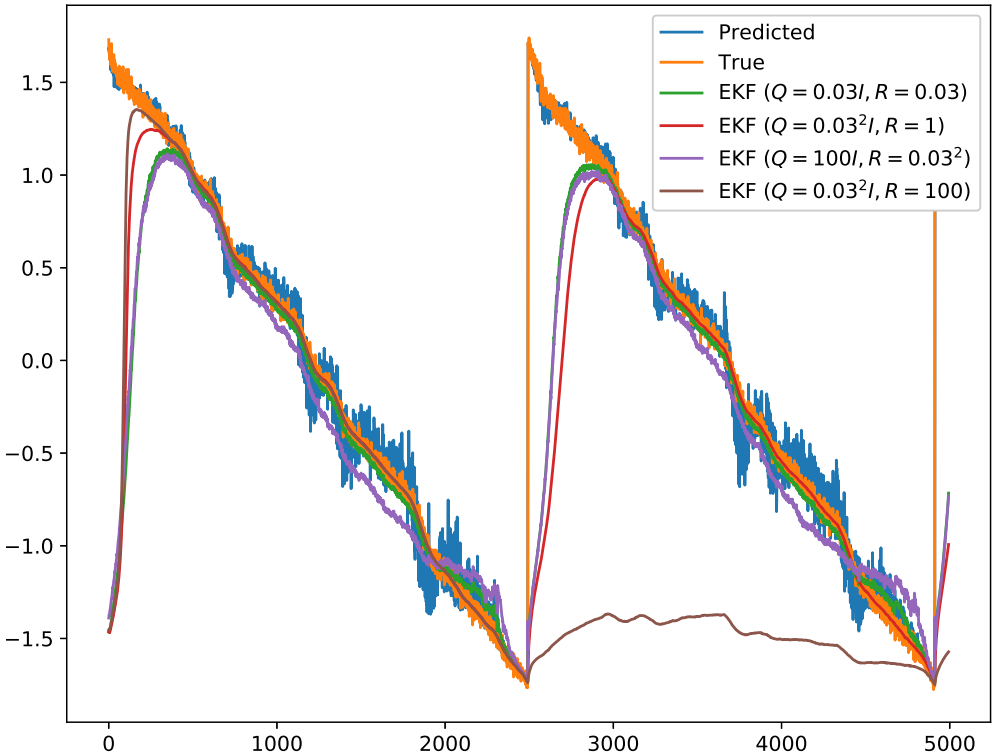
EXAMPLE: NONLINEAR STATE ESTIMATION (BATTERY SOC)

- Virtual sensor architecture:
  - 5 linear observers
  - predictor: ANN, DT, or RF
  - 25,000 training samples
  - 5,000 validation samples

- Requirements:

	Memory (single floats)	CPU time (μs)
ANN	$\approx$ 1,350	6
DT	$\approx$ 3,000	0.1
RF	$\approx$ 30,000	1

(Intel Core i5 6200U)



Comparison with **model-based** extended Kalman filter (EKF) with different covariance matrices

CONCLUSIONS

- **MPC** + **ML** together can have a tremendous impact in the design and implementation of nonlinear control systems:
  - **MPC** and on-line optimization is an extremely powerful control methodology
  - **ML** extremely useful to get **control-oriented models** (system identification) and **control laws** (reinforcement learning) from **data**
- Ignoring **ML** tools would be a mistake (a lot to "learn" from machine learning)
- **ML** alone is not enough to replace control:
  - Black-box modeling can be a failure.
    Better use gray-box models when possible.
  - Approximating the control law can be a failure.
    Don't abandon on-line optimization.