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

"Model Predictive Control" - © A. Bemporad. All rights reserved.
• ML = set of algorithms to extract mathematical models from data

• 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, JAX... (python)
LEARNING PREDICTION MODELS FOR MPC
**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 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 \( x_t = \text{value of an inner layer of the network} \) (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)

\[ y_{t+k} = h_k(x_t, u_t, \ldots, u_{t+k-1}), \quad k = 1, \ldots, N \]
• **Approach:** use a (feedforward deep) neural network model for prediction

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

![Diagram of nonlinear MPC](image)

- **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*)
MPC OF ETHYLENE OXIDATION PLANT

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

\[
\begin{align*}
C_2H_4 + \frac{1}{2}O_2 & \rightarrow C_2H_4O \\
C_2H_4 + 3O_2 & \rightarrow 2CO_2 + 2H_2O \\
C_2H_4O + \frac{5}{2}O_2 & \rightarrow 2CO_2 + 2H_2O
\end{align*}
\]

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

\[
\begin{align*}
\dot{x}_1 &= u_1 (1 - x_1 x_4) \\
\dot{x}_2 &= u_1 (u_2 - x_2 x_4) - A_1 e^{x_4} (x_2 x_4)^{\frac{1}{2}} - A_2 e^{x_4} (x_2 x_4)^{\frac{1}{4}} \\
\dot{x}_3 &= -u_1 x_3 x_4 + A_1 e^{x_4} (x_2 x_4)^{\frac{1}{2}} - A_3 e^{x_4} (x_3 x_4)^{\frac{1}{2}} \\
\dot{x}_4 &= \frac{u_1(1-x_4)}{x_1} + B_1 e^{x_4} (x_2 x_4)^{\frac{1}{2}} + B_2 e^{x_4} (x_3 x_4)^{\frac{1}{4}} \\
y &= x_3
\end{align*}
\]

\[u_1 = \text{manipulated variables, } x_3 = \text{controlled output, } u_2 = \text{measured disturbance}\]

\[x_1 = \text{gas density, } x_2 = \text{ethylene concentration, } x_3 = \text{ethylene oxide concentration, } x_4 = \text{temperature in reactor, } u_1 = \text{feed volumetric flow rate, } u_2 = \text{ethylene concentration in feed}\]
• 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
→ **112 coefficients**

• NN model trained by **ODYS Deep Learning** toolset
  (model fitting + Jacobians → neural model in C)

• Model validated on 200 samples.
  \( x_{3,k+1} \) reproduced from \( x_k, u_k \) with max 0.4% error

"Model Predictive Control" - © A. Bemporad. All rights reserved.
MPC OF ETHYLENE OXIDATION PLANT

- **MPC settings:**
  - sampling time \( T_s = 5 \text{ s} \)
  - 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
**Idea:** use autoencoders and artificial neural networks to learn a **nonlinear state-space model** of **desired order** from input/output data.

\[
O_k = \begin{bmatrix} y'_k \cdots y'_{k-m} \end{bmatrix}'
\]

\[
I_k = \begin{bmatrix} y'_k \cdots y'_{k-n_a+1} u'_k \cdots u'_{k-n_b+1} \end{bmatrix}'
\]
**Training problem:** choose \( n_a, n_b, n_x \) and solve

\[
\min_{f,d,e} \alpha \left( \ell_1(\hat{O}_k, O_k) + \ell_1(\hat{O}_{k+1}, O_{k+1}) \right) \\
+ \beta \ell_2(x_{k+1}^*, x_{k+1}) + \gamma \ell_3(O_{k+1}, O^*_{k+1}) \\
\text{s.t.} \quad x_k = e(I_{k-1}), \quad k = k_0, \ldots, N \\
x_{k+1}^* = f(x_k, u_k), \quad k = k_0, \ldots, N - 1 \\
\hat{O}_k = d(x_k), \quad O_k^* = d(x_k^*), \quad k = k_0, \ldots, N
\]

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

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

\[
f(x_k, u_k) = A(x_k, u_k) [x_k^1] + B(x_k, u_k) u_k \\
y_k = C(x_k, u_k) [x_k^1]
\]

(\( A_{ij}, B_{ij}, C_{ij} = \) 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

\[
\begin{align*}
    x_1(t+1) &= x_1(t) - k_1 \sqrt{x_1(t)} + k_2 u(t) \\
    x_2(t+1) &= x_2(t) + k_3 \sqrt{x_1(t)} - k_4 \sqrt{x_2(t)} \\
    y(t) &= x_2(t) + u(t)
\end{align*}
\]

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

www.mathworks.com
• **Alternative:** learn the entire prediction

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

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

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

\[ e_{\text{FIT}} = \max \left\{ 0, 1 - \frac{\| \hat{y} - y \|_2}{\| y - \bar{y} \|_2} \right\} \]

• Closed-loop LTV-MPC results:

• Model complexity reduction: add group-LASSO term with penalty \( \lambda \)

<table>
<thead>
<tr>
<th>( \lambda )</th>
<th>( e_{\text{FIT}} ) (average on all prediction steps)</th>
<th># nonzero weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>.01</td>
<td>0.853</td>
<td>328</td>
</tr>
<tr>
<td>0.005</td>
<td>0.868</td>
<td>363</td>
</tr>
<tr>
<td>0.001</td>
<td>0.901</td>
<td>556</td>
</tr>
<tr>
<td>0.0005</td>
<td>0.911</td>
<td>888</td>
</tr>
<tr>
<td>0</td>
<td>0.917</td>
<td>9000</td>
</tr>
</tbody>
</table>

"Model Predictive Control" - © A. Bemporad. All rights reserved.
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
Can we design an MPC controller \textbf{without} first identifying a model of the open-loop process ?
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 \( u \)
• 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)
• Experimental results: MPC handles soft constraints on $u$, $\Delta u$ 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!
• **Question**: How to choose the reference model $\mathcal{M}$?

• Can we choose $\mathcal{M}$ from data so that $K_p$ is an **optimal controller**?
**Idea**: parameterize desired closed-loop model $M(\theta)$ and optimize

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

Performance index: $W_y (r(t) - y_p(\theta, t))^2 + W_{\Delta u} \Delta u_p^2(\theta, t)$

Identification error: $W_{\text{fit}} (u(t) - u_v(\theta, t))^2$

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

\[
egin{align*}
  y_p(\theta, t) &= M(\theta) r(t) \\
  u_p(\theta, t) &= K_p(\theta)(r(t) - y_p(\theta, t)) \\
  \Delta u_p(\theta, t) &= u_p(\theta, t) - u_p(\theta, t - 1)
\end{align*}
\]

- Optimal $\theta$ obtained by solving a (non-convex) nonlinear programming problem
**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**

\[ y_L(t) = G(z)u(t) \]
\[ y(t) = |y_L(t)| \arctan(y_L(t)) \]

The data-driven controller is **24% better** than LQR based on identified open-loop model!
DATA-DRIVEN OPTIMAL POLICY SEARCH
• 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: \( \pi : \mathbb{R}^{n_s+n_p} \rightarrow \mathbb{R}^{n_u} \) deterministic control policy

\[
u_t = \pi(s_t, p_t)
\]

• Closed-loop performance of an execution is defined as

\[
J_\infty(\pi, s_0, \{p_\ell, d_\ell\}_{\ell=0}^\infty) = \sum_{\ell=0}^\infty \rho(s_\ell, p_\ell, \pi(s_\ell, p_\ell))
\]

\[
\rho(s_\ell, p_\ell, \pi(s_\ell, p_\ell)) = \text{stage cost}
\]
**Optimal Policy Search Problem**

- **Optimal policy:**
  
  \[
  \pi^* = \arg\min_{\pi} J(\pi) \\
  J(\pi) = \mathbb{E}_{s_0,\{p_\ell,d_\ell\}}[J_\infty(\pi, s_0, \{p_\ell, d_\ell\})]
  \]

  expected performance

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

- **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
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 $\pi$ 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 $D(K_{t-1})$
  4. Update policy: $K_t \leftarrow K_{t-1} - \alpha_t D(K_{t-1})$

- If policy is **learned online** and needs to be applied to the process:
  - Compute the nearest policy $K_t^*$ to $K_t$ that stabilizes the local model
    
    $$K_t^* = \arg\min_K \| K - K_t^s \|_2^2$$
    
    s.t. $K$ stabilizes local linear model

- When policy is learned online, **exploration** is guaranteed by the reference $r_t$
Special Case: Output Tracking

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

\[ \Delta u_t = u_t - u_{t-1} \quad \text{control input increment} \]

- Stage cost:
  \[ \| y_{t+1} - r_t \|^2_{Q_y} + \| \Delta u_t \|^2_R + \| q_{t+1} \|^2_{Q_q} \]

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

\[ s_t = \begin{bmatrix} x_t \\ q_t \end{bmatrix}, \quad p_t = r_t. \]

- Linear policy parametrization:
  \[ \pi_K(s_t, r_t) = -K^s \cdot s_t - K^r \cdot r_t, \quad K = \begin{bmatrix} K^s \\ K^r \end{bmatrix} \]
\[
\begin{align*}
x_{t+1} &= \begin{bmatrix} -0.669 & 0.378 & 0.233 \\ -0.288 & -0.147 & -0.638 \\ -0.337 & 0.589 & 0.043 \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix} \\
y_t &= \begin{bmatrix} -1.139 & 0.319 & -0.571 \end{bmatrix} x_t
\end{align*}
\]

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

\[
\begin{array}{c}
Q_y = 1 \\
R = 0.1 \\
Q_q = 1
\end{array}
\]

\[
\begin{array}{c|c|c}
 n_i & n_o & L \\
\hline
 3 & 3 & 20 \\
\hline
 N_0 & N_r & N_q \\
\hline
 50 & 1 & 10
\end{array}
\]

"Model Predictive Control" - © A. Bemporad. All rights reserved.
Evolution of the error $\|K_t - K_{opt}\|_2$:

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

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

Continuously Stirred Tank Reactor (CSTR)

Feed:
- concentration: 10kg mol/m³
- temperature: 298.15K

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

\[
Q_y = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \quad R = 0.1 \quad Q_q = \begin{bmatrix} 0.01 & 0 \\ 0 & 0 \end{bmatrix}
\]
**NONLINEAR EXAMPLE**

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

(Ferrarotti, Bemporad, 2020a) (Ferrarotti, Bemporad, 2020b) (Ferrarotti, Breschi, Bemporad, 2021) "Model Predictive Control" - © A. Bemporad. All rights reserved.
LEARNING OPTIMAL MPC CALIBRATION
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**

\[
    \min_x f(x)
    \]
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)
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)
• **Goal:** solve the global optimization problem

\[
\min_x \ f(x) \\
\text{s.t.} \quad \ell \leq x \leq u \\
\quad g(x) \leq 0
\]

• **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 \( \beta \) 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!
• **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 \( x \in \{x_1, \ldots, x_N\} \)

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

\( \Delta F = \) observed range of \( f(x_i) \)

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

\[
x_{N+1} = \arg \min \ f(x) - \delta z(x)
\]

s.t. \( \ell \leq x \leq u, g(x) \leq 0 \)

\( \delta = \) exploitation vs exploration tradeoff

• Iterate the procedure to get new samples \( x_{N+2}, \ldots, x_{N_{\text{max}}} \)
GLIS VS BAYESIAN OPTIMIZATION

Results computed on 20 runs per test

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

BO = MATLAB's bayesopt fcn

"Model Predictive Control" - © A. Bemporad. All rights reserved.
We want to auto-tune the linear MPC controller

\[
\min_{k=0}^{50-1} \sum_{k=0}^{50-1} (y_{k+1} - r(t))^2 + (W^{\Delta u}(u_k - u_{k-1}))^2
\]

s.t. \[x_{k+1} = Ax_k + Bu_k\]
\[y_c = Cx_k\]
\[-1.5 \leq u_k \leq 1.5\]
\[u_k \equiv u_{N_u}, \forall k = N_u, \ldots, N - 1\]

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:

\[
f(x) = \sum_{t=0}^{T} (y(t) - r(t))^2 + \frac{1}{2}(u(t) - u(t-1))^2 + 2N_u\]

\{track well\} \{smooth control action\} \{small QP\}
• Result: $x^* = [-0.2341, 2.3007]$ 

$W^{\Delta u} = 0.5833, N_u = 2$
MPC AUTOTUNING EXAMPLE

- 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**

MPC optimized for **Raspberry PI**

<table>
<thead>
<tr>
<th>Position (m)</th>
<th>Angle (deg)</th>
<th>Force (N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>$\phi$</td>
<td>$u$</td>
</tr>
<tr>
<td>$p_{ref}$</td>
<td>$\phi_{ref}$</td>
<td>$u_{ref}$</td>
</tr>
</tbody>
</table>

Optimal sample time = 6 ms

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

"Model Predictive Control" - © A. Bemporad. All rights reserved.
**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:

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

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

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

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

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


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

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

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

• Result: \( W^{\Delta u} = 0.6888, N_u = 2 \)
tested combinations of MPC params

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

\[
\begin{align*}
\dot{x} &= v \cos(\theta + \delta) \\
\dot{y} &= v \sin(\theta + \delta) \\
\dot{\theta} &= \frac{1}{L} v \sin(\delta)
\end{align*}
\]

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 $\Delta v, \Delta \delta$
• Preference query window:

\[ T_s = 0.332 \text{ s}, N_u = 16, N_p = 17, \log(q_{u11}) = 0.06, \log(q_{u22}) = 2.02, t_{\text{comp}}: 0.0867 \text{ s} \]

\[ T_s = 0.243 \text{ s}, N_u = 12, N_p = 17, \log(q_{u11}) = 0.19, \log(q_{u22}) = 0.70, t_{\text{comp}}: 0.0846 \text{ s} \]
• 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 $\Delta v = 1.82$
- 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