TUTORIAL SESSION

MODEL PREDICTIVE CONTROL: FUNDAMENTALS AND FRONTIERS

Alberto Bemporad

imt.lu/ab

SCHOOL FOR ADVANCED STUDIES LUCCA

ASCC-2022 - May 5, 2022

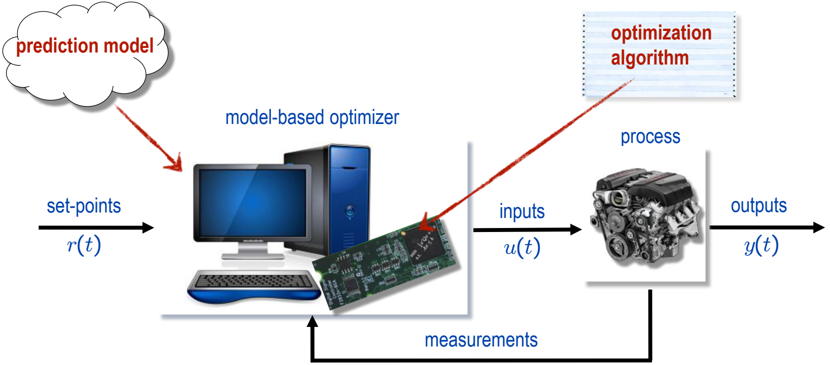
ODYS Advanced Controls & Optimization

WORKSHOP CONTENTS

- **Part 1** (08:45-10:15)
  - **Introduction** and basic principles of MPC, **linear** MPC
  - **Observer** design and integral action
  - Solution methods for linear MPC: **quadratic programming**, **explicit** MPC
- **Part 2** (13:45-15:15)
  - **Linear time-varying** and **nonlinear** MPC
  - **Hybrid** MPC (modeling, control, mixed-integer programming solvers)
  - **Stochastic** MPC based on scenarios
- **Part 3** (15:30-17:00)
  - **Data-driven** linear MPC
  - **Machine-learning** methods for **nonlinear** and **hybrid** MPC
  - Active-learning methods for automatic and preference-based MPC **calibration**

MODEL PREDICTIVE CONTROL: BASIC PRINCIPLES

MODEL PREDICTIVE CONTROL (MPC)



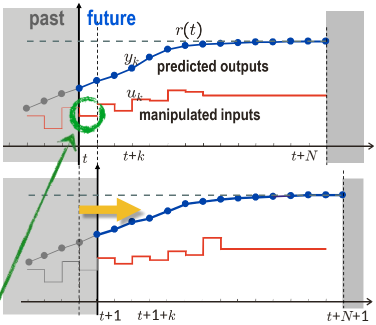
Use a simplified dynamical **model** of the process to **predict** its likely future evolution and choose ~~the "best" control~~ a good action

MODEL PREDICTIVE CONTROL

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

$$
\begin{aligned}
\min_{u_0,\ldots,u_{N-1}} \quad & \sum_{k=0}^{N-1} \|y_k - r(t)\|_2^2 + \rho\|u_k - u_r(t)\|_2^2 \\
\text{s.t.} \quad & x_{k+1} = f(x_k, u_k) && \text{prediction model} \\
& y_k = g(x_k) \\
& u_{\min} \le u_k \le u_{\max} && \text{constraints} \\
& y_{\min} \le y_k \le y_{\max} \\
& x_0 = x(t) && \text{state feedback}
\end{aligned}
$$

→ **numerical optimization problem**



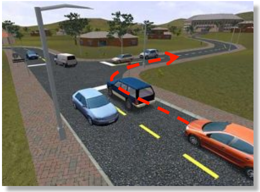
1. **estimate** current state $x(t)$
2. **optimize** wrt $\{u_0, \ldots, u_{N-1}\}$
3. only **apply** optimal u_0 as input $u(t)$

Repeat at all time steps t

DAILY-LIFE EXAMPLES OF MPC

- MPC is like playing chess !

- You use MPC too when you drive !



MPC IN INDUSTRY

- Conceived in the 60's (Rafal, Stevens, 1968) (Propoi, 1963)
- Used in the **process industries** since the 80's (Qin, Badgewell, 2003)
- Now massively spreading to the automotive industry and other sectors
- MPC by **General Motors** and **ODYS** in high-volume production since 2018

  (Bemporad, Bernardini, Long, Verdejo, 2018)

First known mass production of MPC in the automotive industry

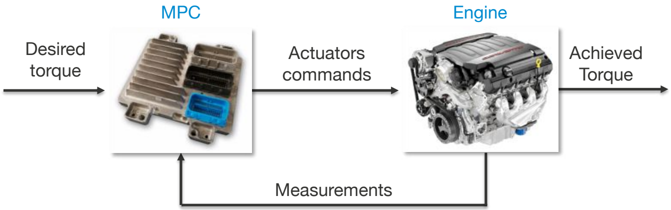
ODYS
Advanced Controls & Optimization

www.odys.it

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

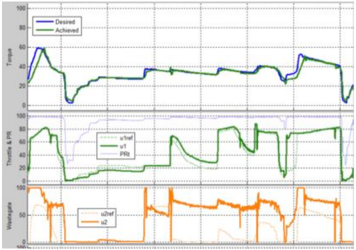
MPC OF GASOLINE TURBOCHARGED ENGINES

- Control **throttle, wastegate, intake & exhaust cams** to make **engine torque** track set-points, with max efficiency and satisfying **constraints**



numerical optimization problem solved in real-time on ECU

(Bemporad, Bernardini, Long, Verdejo, 2018)



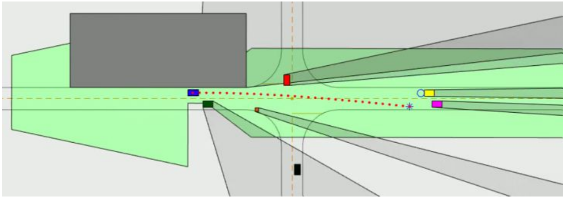
engine operating at low pressure (66 kPa)

(Graf Plessen, Bernardini, Esen, Bemporad, 2018)

- Coordinate **torque request** and **steering** to achieve **safe and comfortable** autonomous driving with no collisions

- MPC combines **path planning**, **path tracking**, and **obstacle avoidance**

- **Stochastic prediction models** used to account for uncertainty (other vehicles/pedestrians, driver's requests)

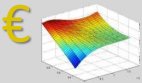


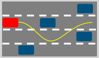
ODYS Advanced Controls & Optimization

TYPICAL USE OF MPC

strategic planner

high-level optimizer





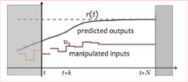
reference optimization

- Static optimizer of steady-state (static I/O models)
- Reference trajectory generator

reference-signal $r(t)$

tactical planner

dynamic optimizer

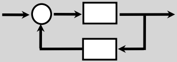


MPC

- optimize transient response
- ensure reference tracking
- handle constraints
- coordinate multiple inputs

measurements $y(t)$ $u(t)$ **actuator set-points**

regulators



low-level controllers

- fast sampling
- single loops

MPC IN INDUSTRY

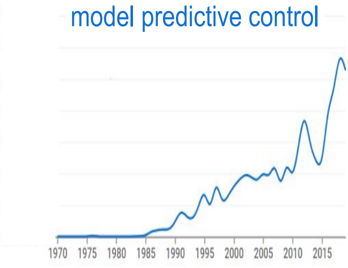
Table 2
The percentage of survey respondents indicating whether a control technology had demonstrated ("Current Impact") or was likely to demonstrate over the next five years ("Future Impact") high impact in practice.

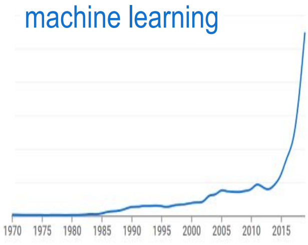
(Samad et al., 2020)

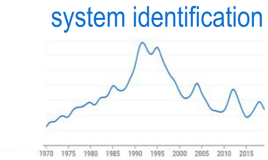
	Current Impact	Future Impact
Control Technology	%High	%High
PID control	91%	78%
System Identification	65%	72%
Estimation and filtering	64%	63%
Model-predictive control	62%	85%
Process data analytics	51%	70%
Fault detection and identification	48%	78%
Decentralized and/or coordinated control	29%	54%
Robust control	26%	42%
Intelligent control	24%	59%
Discrete-event systems	24%	39%
Nonlinear control	21%	42%
Adaptive control	18%	44%
Repetitive control	12%	17%
Hybrid dynamical systems	11%	33%
Other advanced control technology	11%	25%
Game theory	5%	17%

"As can be observed, MPC is clearly considered more impactful, and likely to be more impactful, vis-à-vis other control technologies, especially those that can be considered the "crown jewels" of control theory - robust control, adaptive control, and nonlinear control."

WORD TRENDS







(source: https://books.google.com/ngrams)

LINEAR MODEL PREDICTIVE CONTROL

LINEAR MPC

- Linear prediction model: $\begin{cases} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{cases}$

$$x \in \mathbb{R}^n \quad u \in \mathbb{R}^m \quad y \in \mathbb{R}^p$$

- Constrained optimal control problem (quadratic performance index):

$$\begin{array}{ll} \min\limits_z & x_N' P x_N + \sum\limits_{k=0}^{N-1} x_k' Q x_k + u_k' R u_k \\ \text{s.t.} & u_{\min} \leq u_k \leq u_{\max},\ k = 0, \ldots, N-1 \\ & y_{\min} \leq y_k \leq y_{\max},\ k = 1, \ldots, N \end{array}$$

$$\begin{array}{rcl} R &=& R' \succ 0 \\ Q &=& Q' \succeq 0 \\ P &=& P' \succeq 0 \end{array} \qquad z = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}$$

LINEAR MPC

- Optimization problem (**condensed form**): $x_k = A^k x_0 + \sum_{i=0}^{k-1} A^i B u_{k-1-i}$

$$V(x_0) = \frac{1}{2}x_0' Y x_0 + \quad \min_z \quad \frac{1}{2}z' H z + x_0' F' z$$

(quadratic objective)

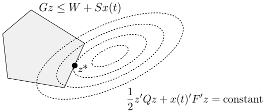
$$H = H' \succ 0$$

$$\text{s.t.} \quad Gz \leq W + Sx_0$$

(linear constraints)

convex Quadratic Program (QP)

- $z = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} \in \mathbb{R}^{Nm}$ is the optimization vector

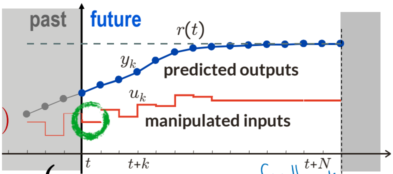


- QP matrices depend on chosen weights, model, and constraints

- **Alternative**: keep also $x_1, \ldots, x_N$ as optimization variables and the equality constraints $x_{k+1} = Ax_k + Bu_k$ (**non-condensed form**, which is **sparse**)

LINEAR MPC ALGORITHM

@ each sampling step t:



- Measure (or estimate) the current state $x(t)$

- Get the solution $z^* = \begin{bmatrix} u_0^* \\ u_1^* \\ \vdots \\ u_{N-1}^* \end{bmatrix}$ of the QP

$$\left\{\begin{array}{ll}\min\limits_{z} & \frac{1}{2}z'Hz + \overbrace{x'(t)F'z}^{\text{feedback}} \\[2ex] \text{s.t.} & Gz \le W + S\underbrace{x(t)}_{\text{feedback}}\end{array}\right.$$

- Apply only $u(t) = u_0^*$, discarding the remaining optimal inputs $u_1^*, \ldots, u_{N-1}^*$

- **Unconstrained MPC**: $\overbrace{Hz + Fx(t)}^{\text{gradient}} = 0 \quad \Rightarrow \quad u(t) = -[I\ 0\ \ldots\ 0]H^{-1}Fx(t)$
linear state feedback!

BASIC CONVERGENCE PROPERTIES

(Keerthi, Gilbert, 1988) (Bemporad, Chisci, Mosca, 1994)

- **Theorem**: Let the MPC law be based on

$$
\begin{aligned}
V^*(x(t)) = \min \quad & \sum_{k=0}^{N-1} x_k' Q x_k + u_k' R u_k \\
\text{s.t.} \quad & x_{k+1} = A x_k + B u_k \\
& u_{\min} \leq u_k \leq u_{\max} \\
& y_{\min} \leq C x_k \leq y_{\max} \\
& x_N = 0 \quad \leftarrow \text{"terminal constraint"}
\end{aligned}
$$

with $R, Q \succ 0, u_{\min} < 0 < u_{\max}, y_{\min} < 0 < y_{\max}$.
If the **optimization problem is feasible at time** $t = 0$ then

$$\lim_{t\to\infty} x(t) = 0, \quad \lim_{t\to\infty} u(t) = 0$$

and the constraints are satisfied at all time $t \geq 0$, for all $R, Q \succ 0$.

- Many more convergence and stability results exist (Mayne, 2014)

LINEAR MPC - TRACKING

- Objective: make the output $y(t)$ track a reference signal $r(t)$
- Let us parameterize the problem using the **input increments**

$$\Delta u(t) = u(t) - u(t-1)$$

- As $u(t) = u(t-1) + \Delta u(t)$ we need to extend the system with a new state $x_u(t) = u(t-1)$

$$\begin{cases} x(t+1) &= Ax(t) + Bu(t-1) + B\Delta u(t) \\ x_u(t+1) &= x_u(t) + \Delta u(t) \end{cases}$$

$$\begin{cases} \begin{bmatrix} x(t+1) \\ x_u(t+1) \end{bmatrix} &= \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \begin{bmatrix} x(t) \\ x_u(t) \end{bmatrix} + \begin{bmatrix} B \\ I \end{bmatrix} \Delta u(t) \\ y(t) &= \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ x_u(t) \end{bmatrix} \end{cases}$$

- Again a linear system with states $x(t), x_u(t)$ and input $\Delta u(t)$

LINEAR MPC - TRACKING

- Optimal control problem (quadratic performance index):

$$
\begin{array}{ll}
\min\limits_{z} & \sum\limits_{k=0}^{N-1} \|W^y(y_{k+1}-r(t))\|_2^2 + \|W^{\Delta u}\Delta u_k\|_2^2 \\
& [\Delta u_k \triangleq u_k - u_{k-1}],\ u_{-1} = u(t-1) \\
\text{s.t.} & u_{\min} \le u_k \le u_{\max},\ k=0,\ldots,N-1 \\
& y_{\min} \le y_k \le y_{\max},\ k=1,\ldots,N \\
& \Delta u_{\min} \le \Delta u_k \le \Delta u_{\max},\ k=0,\ldots,N-1
\end{array}
$$

$$
z = \begin{bmatrix} \Delta u_0 \\ \Delta u_1 \\ \vdots \\ \Delta u_{N-1} \end{bmatrix} \text{ or } z = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}
$$

weight $W^{(\cdot)}$ = diagonal matrix

$$
\begin{array}{ll}
\min\limits_{z} & J(z,x(t)) = \frac{1}{2}z'Hz + [x'(t)\ r'(t)\ u'(t-1)]F'z \\
\text{s.t.} & Gz \le W + S\begin{bmatrix} x(t) \\ r(t) \\ u(t-1) \end{bmatrix}
\end{array}
$$

convex Quadratic Program

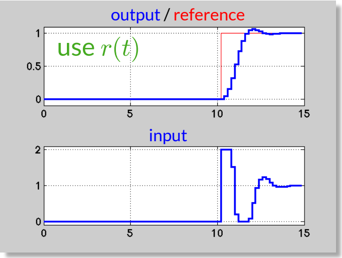
- Add the extra penalty $\|W^u(u_k - u_{\text{ref}}(t))\|_2^2$ to track **input references**
- Constraints may depend on $r(t)$, such as $e_{\min} \le y_k - r(t) \le e_{\max}$

ANTICIPATIVE ACTION (A.K.A. "PREVIEW")

$$\min_{\Delta U} \sum_{k=0}^{N-1} \|W^y(y_{k+1} - r(t+k))\|_2^2 + \|W^{\Delta u}\Delta u(k)\|_2^2$$

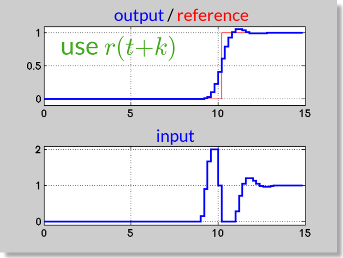
- Reference **not known** in advance (**causal**):

$$r_k \equiv r(t), \forall k = 0, \ldots, N-1$$



- Future refs (partially) **known** in advance (**anticipative action**):

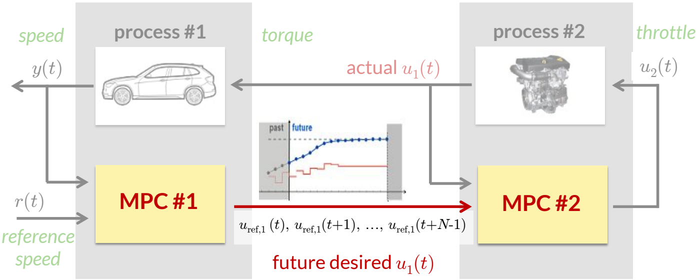
$$r_k = r(t+k), \forall k = 0, \ldots, N-1$$



- Same for previewing **measured disturbances** $x_{k+1} = Ax_k + Bu_k + B_v v(t+k)$

EXAMPLE: CASCADED MPC

- We can use preview also to **coordinate** multiple MPC controllers
- Example: **cascaded MPC**



- MPC #1 commands set-points to MPC #2

OFFSET-FREE TRACKING AND INTEGRAL ACTION

- In control systems, **integral action** occurs if the controller has a transfer-function from the output to the input of the form

$$u(t) = \frac{B(z)}{(z-1)A(z)}y(t), \qquad B(1) \neq 0$$

- One may think that the Δu-formulation of MPC provides integral action ...

  ... is it true ?

- **Example**: we want to regulate the output $y(t)$ to zero of the scalar system

$$\begin{array}{rcl} x(t+1) & = & \alpha x(t) + \beta u(t) \\ y(t) & = & x(t) \end{array}$$

INTEGRAL ACTION AND Δu-FORMULATION

- Design an unconstrained MPC controller with horizon $N = 1$

$$\begin{aligned} \Delta u(t) = \quad & \arg\min_{\Delta u_0} \Delta u_0^2 + \rho y_1^2 \\ \text{s.t.} \quad & u_0 = u(t-1) + \Delta u_0 \\ & y_1 = x_1 = \alpha x(t) + \beta(\Delta u_0 + u(t-1)) \end{aligned}$$

- By substitution, we get

$$\begin{aligned} \Delta u(t) &= \arg\min_{\Delta u_0} \Delta u_0^2 + \rho(\alpha x(t) + \beta u(t-1) + \beta \Delta u_0)^2 \\ &= \arg\min_{\Delta u_0} (1+\rho\beta^2)\Delta u_0^2 + 2\beta\rho(\alpha x(t) + \beta u(t-1))\Delta u_0 \\ &= -\tfrac{\beta\rho\alpha}{1+\rho\beta^2} x(t) - \tfrac{\rho\beta^2}{1+\rho\beta^2} u(t-1) \end{aligned}$$

- Since $x(t) = y(t)$ and $u(t) = u(t-1) + \Delta u(t)$ we get the linear controller

$$u(t) = -\frac{\frac{\rho\beta\alpha}{1+\rho\beta^2} z}{z - \frac{1}{1+\rho\beta^2}} y(t)$$

No pole in $z = 1$

- Reason: MPC gives a feedback gain on both $x(t)$ and $u(t-1)$, not just on $x(t)$

OUTPUT INTEGRATORS AND OFFSET-FREE TRACKING

- Add constant unknown disturbances on measured outputs:

$$\begin{cases} x_{k+1} = Ax_k + Bu_k \\ d_{k+1} = d_k \\ y_k = Cx_k + d_k \end{cases}$$

- Use the extended model to design a **state observer** (e.g., Kalman filter) that estimates both the state $\hat{x}(t)$ and disturbance $\hat{d}(t)$ from $y(t)$

- Why we get offset-free tracking in steady-state (intuitively):

  - the observer makes $C\hat{x}(t) + \hat{d}(t) \to y(t)$ **(estimation error)**
  - the MPC controller makes $C\hat{x}(t) + \hat{d}(t) \to r(t)$ **(predicted tracking error)**
  - the combination of the two makes $y(t) \to r(t)$ **(actual tracking error)**

- In steady state, the term $\hat{d}(t)$ compensates for model mismatch

- See more on survey paper (Pannocchia, Gabiccini, Artoni, 2015)

ERROR FEEDBACK

(Kwakernaak, Sivan, 1972)

- **Idea**: add the integral of the tracking error as an additional state (original idea developed for integral action in state-feedback control)

- Extended prediction model:

$$\begin{cases} x(t+1) & = & Ax(t) + B_u u(t) + 0 \cdot r(t) & \leftarrow \text{r(t) is seen as a meas. disturbance} \\ q(t+1) & = & q(t) + \underbrace{Cx(t) - r(t)}_{\text{tracking error}} & \leftarrow \text{integral action} \\ y(t) & = & Cx(t) \end{cases}$$

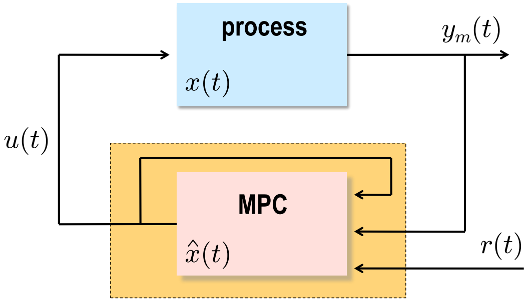
- $\|W^i q\|_2^2$ is penalized in the cost function, otherwise it is useless. W^i is a new tuning knob

- Intuitively, if the MPC closed-loop is asymptotically stable then $q(t)$ converges to a constant, and hence $y(t) - r(t)$ converges to zero.

FREQUENCY ANALYSIS OF MPC (FOR SMALL SIGNALS)

- **Unconstrained MPC** gain + linear observer = linear dynamical system

- Closed-loop MPC analysis can be performed using standard frequency-domain tools (e.g., Bode plots for sensitivity analysis)



CONTROLLER MATCHING

- Given a desired linear controller $u = K_d x$, find a set of weights Q, R, P defining an MPC problem such that

$$-\begin{bmatrix} I\ 0\ \dots\ 0 \end{bmatrix} H^{-1} F = K_d$$

i.e., the **MPC law coincides with K_d** when the constraints are **inactive**

- The above **inverse optimality problem** can be cast to a convex problem

(Di Cairano, Bemporad, 2010)

- Result extended to match any linear controller/observer by LQR/Kalman filter

(Zanon, Bemporad, 2021)

- **MPC Toolbox** (The Mathworks, Inc.): (Bemporad, Ricker, Morari, 1998+)
  - Part of Mathworks' official toolbox distribution
  - All written in **MATLAB** code
  - Great for **education and research**

- **Hybrid Toolbox**: (Bemporad, 2003+)
  - Free download: `http://cse.lab.imtlucca.it/~bemporad/hybrid/toolbox`
  - Great for **research and education**

10,000+ downloads

1.5 downloads/day

- **ODYS Embedded MPC Toolset**: (ODYS, 2013+)
  - Support for **linear & nonlinear MPC** and **extended Kalman filtering**
  - Library-free C code, **MISRA-C 2012 compliant**. Single precision supported
  - **ODYS Deep Learning** supports **neural networks** as prediction models
  - Designed and adopted for **industrial production**

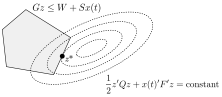
odys.it/embedded-mpc

EMBEDDED QUADRATIC OPTIMIZATION FOR MPC

EMBEDDED LINEAR MPC AND QUADRATIC PROGRAMMING

- MPC based on linear models requires solving a **Quadratic Program (QP)**

$$\begin{array}{ll} \min\limits_{z} & \frac{1}{2}z'Qz + x'(t)F'z + \frac{1}{2}x'(t)Yx(t) \\ \text{s.t.} & Gz \leq W + Sx(t) \end{array} \qquad z = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}$$



ON MINIMIZING A CONVEX FUNCTION SUBJECT TO LINEAR INEQUALITIES

By E. M. L. BEALE

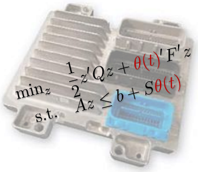
Admiralty Research Laboratory, Teddington, Middlesex

SUMMARY

THE minimization of a convex function of variables subject to linear inequalities is discussed briefly in general terms. Dantzig's Simplex Method is extended to yield finite algorithms for minimizing either a convex quadratic function or the sum of the t largest of a set of linear functions, and the solution of a generalization of the latter problem is indicated. In the last two sections a form of linear programming with random variables as coefficients is described, and shown to involve the minimization of a convex function.

(Beale, 1955)

> A rich set of good QP algorithms is available today



- Not all QP algorithms are suitable for **industrial embedded control**

MPC IN A PRODUCTION ENVIRONMENT

Key requirements for deploying MPC in production:

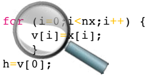
1. **speed (throughput)**
   - **worst-case** execution time less than sampling interval
   - also fast on **average** (to free the processor to execute other tasks)

2. limited **memory and CPU power** (e.g., 150 MHz / 50 kB)

3. **numerical robustness** (single precision arithmetic)

4. **certification** of worst-case execution time

5. **code simple enough** to be validated/verified/certified
   (library-free C code, easy to check by production engineers)

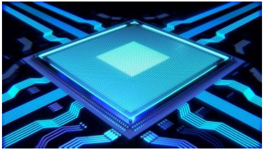


EMBEDDED SOLVERS IN INDUSTRIAL PRODUCTION

- Multivariable MPC controller
- Sampling frequency = 40 Hz (= 1 QP solved every 25 ms)
- Vehicle operating $\approx$1 hr/day for $\approx$360 days/year on average
- Controller running on 10 million vehicles

~520,000,000,000,000 QP/yr

and none of them should fail.



DUAL GRADIENT PROJECTION FOR QP

(Goldstein, 1964) (Levitin, Poljak, 1965) (Combettes, Waijs, 2005)

- Consider the strictly convex QP and its dual

$$\begin{array}{rl} \min & \frac{1}{2}z'Qz + x'F'z \\ \text{s.t.} & Gz \leq W + Sx \end{array} \quad \Rightarrow \quad \begin{array}{rl} \min & \frac{1}{2}y'Hy + (Dx+W)'y \\ \text{s.t.} & y \geq 0 \end{array}$$

with $H = GQ^{-1}G'$, $D = S + GQ^{-1}F$. Take $L \geq \lambda_{\max}(H)$

- Apply **proximal gradient method** to dual QP:

$$\boxed{y^{k+1} = \max\{y^k - \frac{1}{L}(Hy^k + Dx + W), 0\}} \qquad y_0 = 0$$

- The primal solution is related to the dual solution by

$$z^k = -Q^{-1}(Fx + G'y^k)$$

- Convergence is slow: the initial error $f(z^0) - f(z^*)$ reduces as $1/k$

(Nesterov, 1983) (Beck, Teboulle, 2008) (Patrinos, Bemporad, 2014)

- The **fast gradient method** is applied to solve the dual QP problem

$$\begin{array}{ll} \min_z & \frac{1}{2}z'Qz + x'F'z \\ \text{s.t.} & Gz \leq W + Sx \end{array}$$

$$\begin{array}{rcl} K & = & Q^{-1}G' \\ J & = & Q^{-1}F \\ L & \geq & \lambda_{\max}(GQ^{-1}G') \end{array}$$

$$\beta_k = \max\{\tfrac{k-1}{k+2}, 0\}$$

$$\begin{array}{rcl} w^k & = & y^k + \beta_k(y^k - y^{k-1}) \\ z^k & = & -Kw^k - Jx \\ s^k & = & \frac{1}{L}Gz^k - \frac{1}{L}(W + Sx) \\ y^{k+1} & = & \max\left\{w^k + s^k, 0\right\} \end{array}$$

```
while k<maxiter
  beta=max((k-1)/(k+2),0);
  w=y+beta*(y-y0);
  z=-(iMG*w+iMc);
  s=GL*z-bL;

  y0=y;

  % Termination
  if all(s<=epsGL)
    gapL=-w'*s;
    if gapL<=epsVL
      return
    end
  end

  y=w+s;
  k=k+1;
end
```

- Very **simple to code**

FAST GRADIENT PROJECTION FOR (DUAL) QP

- **Termination criteria**: when the following two conditions are met

$$\begin{array}{rcl} s_i^k & \leq & \frac{1}{L}\epsilon_G,\ i=1,\ldots,m \\ -(w^k)'s^k & \leq & \frac{1}{L}\epsilon_f \end{array}$$

primal feasibility

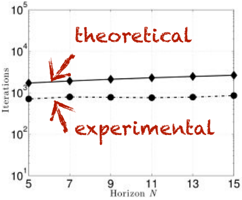
optimality

the solution $z^k = -Kw^k - Jx$ satisfies $G_iz^k - W_i - S_ix \leq \epsilon_G$ and, if $w^k \geq 0$,

$$f(z^k) - f(z^*) \leq f(z^k) - \underbrace{q(w^k)}_{\text{dual fcn}} = -(w^k)'s^kL \leq \epsilon_f$$

- Convergence rate: $f(x^k) - f(x^*) \leq \dfrac{2L}{(k+2)^2}\|z_0 - z^*\|_2^2$

- Tight bounds on maximum number of iterations



ADMM

(Gabay, Mercier, 1976) (Glowinski, Marrocco, 1975) (Douglas, Rachford, 1956) (Boyd et al., 2010)

- Alternating Directions Method of Multipliers for QP

$$\begin{array}{ll}\min & \frac{1}{2}z'Qz + c'z \\ \text{s.t.} & \ell \le Az \le u\end{array}$$

$$\begin{aligned} z^{k+1} &= -(Q+\rho A'A)^{-1}(\rho A'(v^k - s^k) + c) \\ s^{k+1} &= \min\{\max\{Az^{k+1} + v^k, \ell\}, u\} \\ v^{k+1} &= v^k + Az^{k+1} - s^{k+1} \end{aligned}$$

```
while k<maxiter
  k=k+1;
  z=-iM*(c+A'*(rho*(v-s)));
  Az=A*z;
  s=max(min(Az+v,u),ell);
  v=v+Az-s;
end
```

(7 lines EML code)
(≈40 lines of C code)

ρv = dual vector

- Matrix $(Q+\rho A'A)$ must be nonsingular
- The factorization of matrix $(Q+\rho A'A)$ can be done at start and cached
- Very **simple to code**. Sensitive to matrix **scaling** (as gradient projection)
- Used in many applications (control, signal processing, machine learning)

REGULARIZED ADMM FOR QUADRATIC PROGRAMMING

(Stellato, Banjac, Goulart, Bemporad, Boyd, 2020)

- Robust "regularized" ADMM iterations:

$$\begin{aligned} z^{k+1} &= -(Q+\rho A^T A + \epsilon I)^{-1}(c - \epsilon z^k + \rho A^T(v^k - z^k)) \\ s^{k+1} &= \min\{\max\{Az^{k+1}+v^k,\ell\},u\} \\ v^{k+1} &= v^k + Az^{k+1} - s^{k+1} \end{aligned}$$

- Works for any $Q \succeq 0$, A, and choice of $\epsilon > 0$
- **Simple** to code, **fast**, and **robust**
- Only needs to factorize $\begin{bmatrix} Q+\epsilon I & A' \\ A & -\frac{1}{\rho}I \end{bmatrix}$ once
- Implemented in free **osQP solver**

  (Python interface: $\approx$ 1,700,000 downloads)

http://osqp.org

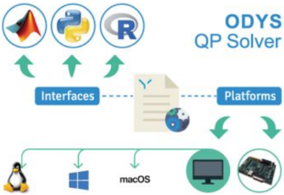
- Extended to solve **mixed-integer quadratic programming** problems

  (Stellato, Naik, Bemporad, Goulart, Boyd, 2018)

ODYS QP SOLVER

- General purpose QP solver designed for **industrial production**

$$
\begin{array}{ll}
\min\limits_{z} & \frac{1}{2}z'Qz+c'z \\
\text{s.t.} & b_\ell \leq Az \leq b_u \\
& \ell \leq z \leq u \\
& Ez = f
\end{array}
$$



- Implements a **proprietary** state-of-the-art method for QP
- Completely written in **ANSI-C** and **MISRA-C 2012** compliant
- **Fast**, **robust** (also in single precision), **low-memory** requirements
- **optimized version for MPC** available ($\approx$ 50% faster)
- Licensed to several automotive OEMs and Tier-1 suppliers
- **Certifiable** execution time

odys.it/qp

PRIMAL-DUAL INTERIOR-POINT METHOD FOR QP

(Nocedal, Wright, 2006) (Gondzio, Terlaki, 1994)

- The Karush-Kuhn-Tucker (KKT) optimality conditions for the convex QP

$$\begin{array}{rl} \min_x & \frac{1}{2}x'Qx + c'x \\ \text{s.t.} & Ax \leq b \\ & Ex = f \end{array} \qquad Q = Q' \succeq 0$$

  are

$$\begin{array}{rclcl} r_Q & = & Qx + c + E'y + A'z & = & 0 \\ r_E & = & Ex - f & = & 0 \\ r_A & = & Ax + s - b & = & 0 \\ r_S & = & [z_1 s_1 \ \ldots \ z_m s_m]' & = & 0 \\ z, s & \geq & 0 & & \end{array}$$

x = primal vars
y = dual vars (eq. constr.)
s = slacks (ineq. constr.)
z = dual vars (ineq. constr.)

- In a nutshell, **interior-point** methods use Newton's method with line search to solve the above nonlinear system of equations

- The complementary slackness constraint is replaced by $z_i s_i = \mu$ and $\mu \to 0$

PRIMAL-DUAL INTERIOR-POINT METHOD FOR QP

(Nocedal, Wright, 2006) (Gondzio, Terlaki, 1994)

- Each interior-point iteration requires solving a linear system of the form

$$\begin{bmatrix} Q & E' & A' & 0 \\ E & 0 & 0 & 0 \\ A & 0 & 0 & I \\ 0 & 0 & S & Z \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta s \end{bmatrix} = \begin{bmatrix} -r_Q \\ -r_E \\ -r_A \\ -r_S \end{bmatrix} \qquad \begin{array}{rcl} Z & = & \mathrm{diag}\ z \\ S & = & \mathrm{diag}\ s \end{array}$$

- In MPC the **structure** $x_{k+1} = Ax_k + Bu_k$ can be heavily exploited to factorize/solve the linear systems efficiently (Rao, Wright, Rawlings, 1998) (Wright, 2018)

- Linear systems tends to become ill-conditioned at convergence

WHICH QP SOLVER TOO CHOOSE FOR MPC ?

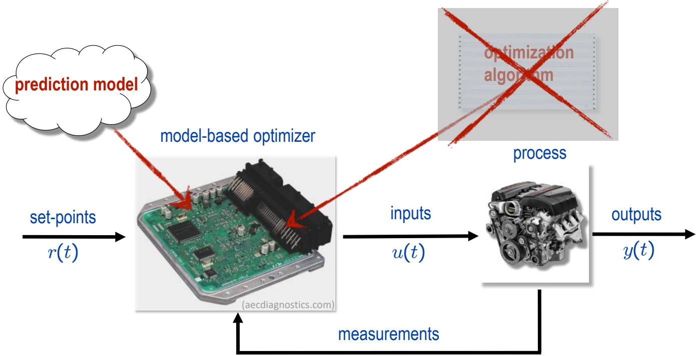
QP solver →	Active-Set	Interior-Point	ADMM	GPAD
CPU time (**small/medium** & **dense**)				
CPU time (**large** & **sparse**)				
Worst-case estimate of CPU time				
Numerical robustness (e.g., in single precision)				
Software complexity (linear algebra libraries)				

small-scale ≈ 20- variables, 50- constraints

large-size ≈ 500+ variables, 2500+ constraints

- AS, ADMM require simpler linear algebra than IP
- IP gives good solutions within 10-15 iterations (usually ...)
- AS iterations tend to increase when both vars and constraints increase

MPC WITHOUT ON-LINE QP

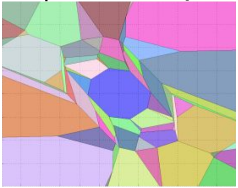


- Can we implement constrained linear MPC **without an on-line QP solver** ?

YES !

EXPLICIT MODEL PREDICTIVE CONTROL

- Continuous & piecewise affine solution of strictly convex multiparametric QP



$$
\begin{aligned}
z^*(x) = \arg\min_z \quad & \tfrac{1}{2} z' Q z + x' F' z \\
\text{s.t.} \quad & Gz \leq W + Sx
\end{aligned}
$$

(Bemporad, Morari, Dua, Pistikopoulos, 2002)

- Corollary: linear MPC is continuous & piecewise affine !

$$
z^* = \begin{bmatrix} \mathbf{u_0} \\ u_1 \\ \vdots \\ u^*_{N-1} \end{bmatrix}
\qquad
u_0^*(x) = \begin{cases} F_1 x + g_1 & \text{if} \quad H_1 x \leq K_1 \\ \vdots & \vdots \\ F_M x + g_M & \text{if} \quad H_M x \leq K_M \end{cases}
$$

- New mpQP solver based on NNLS available (Bemporad, 2015)
  and included in MPC Toolbox since R2014b (Bemporad, Morari, Ricker, 1998-today)

DOUBLE INTEGRATOR EXAMPLE

- Model and constraints: $\begin{cases} x(t+1) &= \left[\begin{smallmatrix}1&1\\0&1\end{smallmatrix}\right] x(t) + \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right] u(t) \\ y(t) &= \left[\begin{smallmatrix}1&0\end{smallmatrix}\right] x(t) \end{cases}$

  $-1 \leq u(t) \leq 1$

- Objective:

$$\min \sum_{k=0}^{\infty} y_k^2 + \frac{1}{100} u_k^2$$

$$u_k = Kx_k,\ \forall k \geq N_u,\ K = \text{LQR gain}$$

$$N_u = N = 2$$

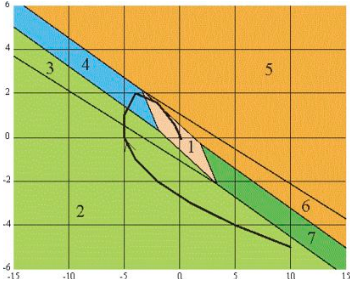
$$\Rightarrow \left(\sum_{k=0}^{1} y_k^2 + \tfrac{1}{100} u_k^2\right) + x_2' \underbrace{\left[\begin{smallmatrix}2.1429 & 1.2246\\ 1.2246 & 1.3996\end{smallmatrix}\right]}_{\text{solution of algebraic Riccati equation}} x_2$$

- QP matrices (cost function normalized by max singular value of H)

$$H = \left[\begin{smallmatrix}0.8365 & 0.3603\\ 0.3603 & 0.2059\end{smallmatrix}\right],\ F = \left[\begin{smallmatrix}0.4624 & 1.2852\\ 0.1682 & 0.5285\end{smallmatrix}\right]$$

$$G = \begin{bmatrix}1 & 0\\ -1 & 0\\ 0 & 1\\ 0 & -1\end{bmatrix},\ W = \begin{bmatrix}1\\1\\1\\1\end{bmatrix},\ S = \begin{bmatrix}0&0\\0&0\\0&0\\0&0\end{bmatrix}$$

DOUBLE INTEGRATOR EXAMPLE - EXPLICIT SOLUTION



$$N_u = 2$$

$$
u(x) = \begin{cases}
\begin{bmatrix} -0.8166 & -1.75 \end{bmatrix} x & \text{if } \begin{bmatrix} -0.8166 & -1.75 \\ 0.8166 & 1.75 \\ 0.6124 & 0.4957 \\ -0.6124 & -0.4957 \end{bmatrix} x \leq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} & \text{(Region \#1)} \\
1 & \text{if } \begin{bmatrix} 0.3864 & 1.074 \\ 0.297 & 0.9333 \end{bmatrix} x \leq \begin{bmatrix} -1 \\ -1 \end{bmatrix} & \text{(Region \#2)} \\
1 & \text{if } \begin{bmatrix} -0.297 & -0.9333 \\ 0.8166 & 1.75 \\ 0.9712 & 2.699 \end{bmatrix} x \leq \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} & \text{(Region \#3)} \\
\begin{bmatrix} -0.5528 & -1.536 \end{bmatrix} x + 0.4308 & \text{if } \begin{bmatrix} -0.9712 & -2.699 \\ 0.3864 & 1.074 \\ 0.6124 & 0.4957 \end{bmatrix} x \leq \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} & \text{(Region \#4)} \\
-1 & \text{if } \begin{bmatrix} -0.3864 & -1.074 \\ -0.297 & -0.9333 \end{bmatrix} x \leq \begin{bmatrix} -1 \\ -1 \end{bmatrix} & \text{(Region \#5)} \\
-1 & \text{if } \begin{bmatrix} 0.297 & 0.9333 \\ -0.8166 & -1.75 \\ -0.9712 & -2.699 \end{bmatrix} x \leq \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} & \text{(Region \#6)} \\
\begin{bmatrix} -0.5528 & -1.536 \end{bmatrix} x - 0.4308 & \text{if } \begin{bmatrix} -0.3864 & -1.074 \\ 0.9712 & 2.699 \\ -0.6124 & -0.4957 \end{bmatrix} x \leq \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} & \text{(Region \#7)}
\end{cases}
$$

go to demo `linear/doubleintexp.m` (Hybrid Toolbox for MATLAB)

(Bemporad, 2003-today)

http://cse.lab.imtlucca.it/~bemporad/hybrid/toolbox

≈10,000+ downloads

≈1.5 downloads/day

DOUBLE INTEGRATOR EXAMPLE - HYBRID TOOLBOX

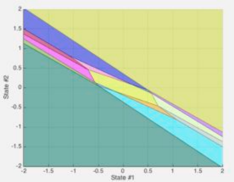
```
Ts=1; % sampling time
model=ss([1 1;0 1],[0;1],[0 1],0,Ts); % prediction model

limits.umin=-1; limits.umax=1; % input constraints

interval.Nu=2; % control horizon
interval.N=2; % prediction horizon

weights.R=.1;
weights.Q=[1 0;0 0];
weights.P='lqr'; % terminal weight = Riccati matrix
weights.rho=+Inf; % hard constraints on outputs, if present

Cimp=lincon(model,'reg',weights,interval,limits); % MPC

range=struct('xmin',[-15 -15],'xmax',[15 15]);
Cexp=expcon(Cimp,range); % explicit MPC

x0=[10,-.3]';
Tstop=40; % simulation time
[X,U,T,Y,I]=sim(Cexp,model,[],x0,Tstop);
```

MPC TOOLBOX

(Bemporad, Morari, Ricker, $\geq$2014)

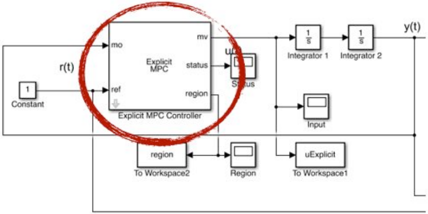
- **`@explicitMPC`** object The MathWorks

```
>> mpcobj = mpc(plant, Ts, p, m);
>> empcobj = generateExplicitMPC(mpcobj, range);
>> empcobj2 = simplify(empcobj, 'exact')
>> [y2,t2,u2] = sim(empcobj,Tf,ref);
>> u = mpcmoveExplicit(empcobj,xmpc,y,ref);
```



- Very simple and robust online PWA evaluation function

```
i=0; imin=0; vmin=Inf; flag=0;
while  found && i<nr
    i=i+1;
    v=max(pwafun(i).H*th-pwafun(i).K);
    if v<=0
        found=true; flag=1;
    else
        if vmin>v
            vmin=v; imin=i;
        end
    end
end
x=pwafun(imin).F*th+pwafun(imin).G;
```



Copyright 1990-2014 The MathWorks, Inc.

APPLICABILITY OF EXPLICIT MPC

- Consider the following general MPC formulation

$$
\begin{array}{ll}
\min\limits_{z} & \displaystyle\sum_{k=0}^{N-1} \frac{1}{2}(y_k - r(t+k))' S (y_k - r(t+k)) + \frac{1}{2}\Delta u_k' T \Delta u_k \\
& \quad + (u_k - u_r(t+k))' V (u_k - u_r(t+k))' + \rho_\epsilon \epsilon^2 \\
\\
\text{subj. to} & x_{k+1} = A x_k + B u_k + B_v v(t+k),\ k = 0, \ldots, N-1 \\
& y_k = C x_k + D u_k + D_v v(t+k),\ k = 0, \ldots, N-1 \\
& u_{\min}(t+k) \leq u_k \leq u_{\max}(t+k),\ k = 0, \ldots, N-1 \\
& \Delta u_{\min}(t+k) \leq \Delta u_k \leq \Delta u_{\max}(t+k),\ k = 0, \ldots, N-1 \\
& y_{\min}(t+k) - \epsilon V_{\min} \leq y_k \leq y_{\max}(t+k) + \epsilon V_{\max},\ k = 1, \ldots, N \\
& x_0 = x(t)
\end{array}
$$

- Everything marked in **red** can be time-varying in explicit MPC
- Not applicable to time-varying models and weights

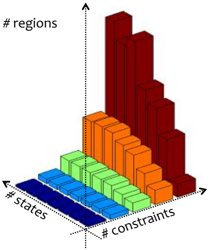
COMPLEXITY OF MULTIPARAMETRIC SOLUTIONS

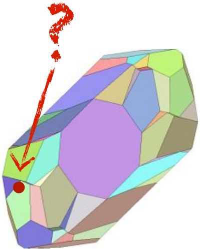
- Number n_r of regions = # optimal combinations of active constraints:
  - mainly depends on the number q of constraints: $n_r \leq \sum_{h=0}^{q} \binom{q}{h} = 2^q$

    (this is a worst-case estimate, most of the combinations are never optimal!)
  - also depends on the number s of free variables
  - weakly depends on the number n of parameters (states + references)

states/horizon	$N=1$	$N=2$	$N=3$	$N=4$	$N=5$
n=2	3	6.7	13.5	21.4	19.3
n=3	3	6.9	17	37.3	77
n=4	3	7	21.65	56	114.2
n=5	3	7	22	61.5	132.7
n=6	3	7	23.1	71.2	196.3
n=7	3	6.95	23.2	71.4	182.3
n=8	3	7	23	70.2	207.9

average on 20 random SISO systems w/ input saturation





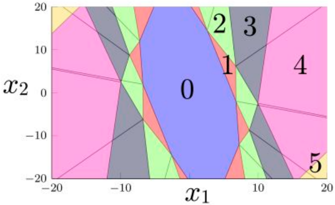
Which is the region the current $x(t)$ belongs to ?

Approaches:

- Store all regions and search linearly through them
- Exploit properties of mpLP solution to locate $x(t)$ from value function (also extended to mpQP) (Baotic, Borrelli, Bemporad, Morari 2008)
- Organize regions on a tree for logarithmic search (Tøndel, Johansen, Bemporad, 2003)
- For mpLP, recast as weighted nearest neighbor problem (logarithmic search) (Jones, Grieder, Rakovic, 2003)
- Exploit reachability analysis (Spjøtvold, Rakovic, Tøndel, Johansen, 2006) (Wang, Jones, Maciejowski, 2007)
- Use bounding boxes and trees (Christophersen, Kvasnica, Jones, Morari, 2007)

COMPLEXITY CERTIFICATION FOR ACTIVE-SET QP SOLVERS

- **Result**: The **number of iterations** to solve the QP via a dual active-set method is a **piecewise constant function** of the parameter x



(Cimini, Bemporad, 2017)

We can **exactly** quantify how many iterations (flops) the QP solver takes in the worst-case !

- Examples (from MPC Toolbox):

	inverted pendulum	DC motor	nonlinear demo	AFTI F16
Explicit MPC				
max flops	3382	1689	9184	16434
max memory (kB)	55	30	297	430
Implicit MPC				
max flops	3809	2082	7747	7807
sqrt	27	9	37	33
max memory (kB)	15	13	20	16

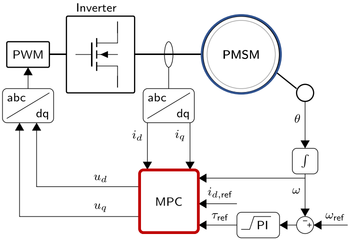
- QP certification algorithm currently used in industrial production projects

(Cimini, Bernardini, Levijoki, Bemporad, 2021)

- MPC of **Permanent Magnet Synchronous Motor**

- **Goal**: control motor torque

- Nonlinear isotropic PMSM model approximated by linear model $@\omega(t)=\omega_0$:

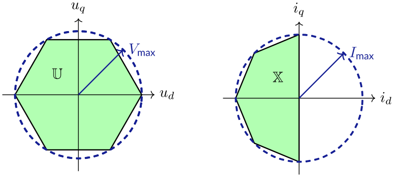


$$\dot{x}=\frac{d}{dt}\begin{bmatrix}i_d(t)\\i_q(t)\end{bmatrix}=\begin{bmatrix}-\frac{R}{L} & \omega_0\\-\omega_0 & -\frac{R}{L}\end{bmatrix}\begin{bmatrix}i_d(t)\\i_q(t)\end{bmatrix}+\begin{bmatrix}\frac{1}{L} & 0\\0 & \frac{1}{L}\end{bmatrix}\begin{bmatrix}u_d(t)\\u_q(t)\end{bmatrix}+\begin{bmatrix}0\\-\frac{\lambda}{L}\end{bmatrix}\omega(t)$$

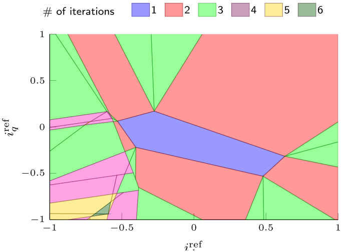
$$y(t)=\begin{bmatrix}i_d(t)\\\tau(t)\end{bmatrix}=\begin{bmatrix}1 & 0\\0 & K_t\end{bmatrix}\begin{bmatrix}i_d(t)\\i_q(t)\end{bmatrix}$$

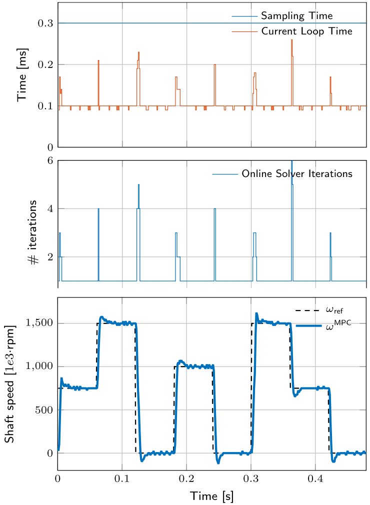
d = direct, q = quadrature

- Voltage/current constraints: (polyhedral approximation)



- **Linear MPC** formulation, solved by **ODYS QP**
- Platform: TI F28335 Delfino 32-bit DSP
  150 MHz CPU, single precision
- **Complexity certification** algorithm guarantees
  2431 flops is the worst-case (=6 QP iters)





- Memory occupancy: 13 kB
  ($\leq$ single-access RAM block of 34 kB)
- Sampling time = 0.3 ms $\Rightarrow$
  real-time QP is 100% feasible

SUBOPTIMAL SOLUTIONS - FUNCTION REGRESSION

- 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)
  - **Nonlinear systems identification** (Canale, Fagiano, Milanese, 2008)
  - **Decision trees**, **random forests**, **K-nearest neighbors**, ...

- Approach works for linear/nonlinear/stochastic/hybrid MPC

LINEAR TIME-VARYING MPC

LINEAR TIME-VARYING MODELS

- **Linear Time-Varying (LTV)** model

$$\begin{cases} x_{k+1} &= A_k(t)x_k + B_k(t)u_k \\ y_k &= C_k(t)x_k \end{cases}$$

- At each time t the model can also change over the prediction horizon k
- Possible measured disturbances are embedded in the model
- On-line optimization is still a QP

$$\begin{array}{ll} \min\limits_{z} & \frac{1}{2}z'H(t)z + \begin{bmatrix} x(t) \\ r(t) \\ u(t-1) \end{bmatrix}' F(t)'z \\ \text{s.t.} & G(t)z \leq W(t) + S(t)\begin{bmatrix} x(t) \\ r(t) \\ u(t-1) \end{bmatrix} \end{array}$$

- The QP matrices cannot be constructed offline

LINEARIZING A NONLINEAR MODEL

- LTV models can be obtained by linearizing a **nonlinear model**

$$\begin{cases} \frac{dx_c(t)}{dt} & = & f(x_c(t), u_c(t)) \\ y_c(t) & = & g(x_c(t)) \end{cases}$$

- At time t, consider the **nominal trajectory**

$$U = \{\bar{u}_c(t), \bar{u}_c(t+T_s), \ldots, \bar{u}_c(t+(N-1)T_s)\}$$

For example U = shifted previous sequence optimized by MPC @$t-1$

- **Integrate** the model from $\bar{x}_c(t)$ and get nominal state/output trajectories

$$\begin{aligned} X &= \{\bar{x}_c(t), \bar{x}_c(t+T_s), \ldots, \bar{x}_c(t+(N-1)T_s)\} \\ Y &= \{\bar{y}_c(t), \bar{y}_c(t+T_s), \ldots, \bar{y}_c(t+(N-1)T_s)\} \end{aligned}$$

For example $\bar{x}_c(t) =$ current state

LINEARIZING A NONLINEAR MODEL

- **Linearize** the nonlinear model around the nominal states and inputs:

$$
\begin{aligned}
\frac{dx_c}{dt} &= f(x_c,u_c) \approx \underbrace{f(\bar{x}_c,\bar{u}_c)}_{\frac{d\bar{x}_c}{dt}} + \underbrace{\left.\frac{\partial f}{\partial x_c}\right|_{\bar{x}_c,\bar{u}_c}}_{\text{Jacobian matrix } A_c}(x_c-\bar{x}_c) + \underbrace{\left.\frac{\partial f}{\partial u_c}\right|_{\bar{x}_c,\bar{u}_c}}_{\text{Jacobian matrix } B_c}(u_c-\bar{u}_c)\\
y &= g(x_c) \approx \underbrace{g(\bar{x}_c)}_{\bar{y}_c} + \underbrace{\left.\frac{\partial g}{\partial x_c}\right|_{\bar{x}_c}}_{\text{Jacobian matrix } C}(x_c-\bar{x}_c)
\end{aligned}
$$

- Define $x \triangleq x_c - \bar{x}_c, u \triangleq u_c - \bar{u}_c, y \triangleq y_c - \bar{y}_c$ and get the linear system

$$
\frac{dx}{dt} = A_c x + B_c u \qquad\qquad y = Cx
$$

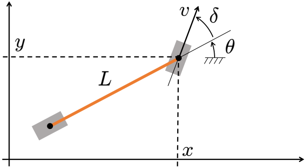
- Convert linear model to **discrete-time** and get matrices (A_k, B_k, C_k)

- **Alternative**: compute (A_k, B_k, C_k) (a.k.a. **sensitivities**) during integration

AUTONOMOUS DRIVING EXAMPLE

- **Goal**: Control longitudinal acceleration and steering angle of the vehicle simultaneously for autonomous driving with obstacle avoidance

- **Approach**: MPC based on a bicycle-like kinematic model of the vehicle in Cartesian coordinates



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

(x, y)	Cartesian position of front wheel	v	velocity at front wheel
θ	vehicle orientation	δ	steering input
L	vehicle length = 4.5 m		

AUTONOMOUS DRIVING EXAMPLE

- Let $x_n, y_n, \theta_n, v_n, \delta_n$ nominal states/inputs satisfying

$$\begin{bmatrix} \dot{x}_n \\ \dot{y}_n \\ \dot{\theta}_n \end{bmatrix} = \begin{bmatrix} v_n \cos(\theta_n + \delta_n) \\ v_n \sin(\theta_n + \delta_n) \\ \frac{v_n}{L} \sin(\delta_n) \end{bmatrix}$$

feasible nominal trajectory

- Linearize the model around the nominal trajectory:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \approx \begin{bmatrix} \dot{x}_n \\ \dot{y}_n \\ \dot{\theta}_n \end{bmatrix} + A_c \begin{bmatrix} x - x_n \\ y - y_n \\ \theta - \theta_n \end{bmatrix} + B_c \begin{bmatrix} v - v_n \\ \delta - \delta_n \end{bmatrix}$$

linearized model

where A_c, B_c are the **Jacobian matrices**

$$A_c = \begin{bmatrix} 0 & 0 & -v_n \sin(\theta_n + \delta_n) \\ 0 & 0 & v_n \cos(\theta_n + \delta_n) \\ 0 & 0 & 0 \end{bmatrix} \qquad B_c = \begin{bmatrix} \cos(\theta_n + \delta_n) & -v_n \sin(\theta_n + \delta_n) \\ \sin(\theta_n + \delta_n) & v_n \cos(\theta_n + \delta_n) \\ \frac{1}{L} \sin(\delta_n) & \frac{v_n}{L} \cos(\delta_n) \end{bmatrix}$$

- Use first-order Euler method to discretize model:

$$A = I + T_s A_c, \quad B = T_s B_c, \quad T_s = 50\,\mathrm{ms}$$

AUTONOMOUS DRIVING EXAMPLE

- Constraints on inputs and input variations $\Delta v_k = v_k - v_{k-1}, \Delta\delta_k = \delta_k - \delta_{k-1}$:

$$\begin{array}{lll} -20 \leq v \leq 70 & \text{km/h} & \text{velocity constraint} \\ -45 \leq \delta \leq 45 & \text{deg} & \text{steering angle} \\ -5 \leq \Delta\delta \leq 5 & \text{deg} & \text{steering angle rate} \end{array}$$

- Stage cost to minimize:

$$(x - x_{\text{ref}})^2 + (y - y_{\text{ref}})^2 + \Delta v^2 + \Delta\delta^2$$

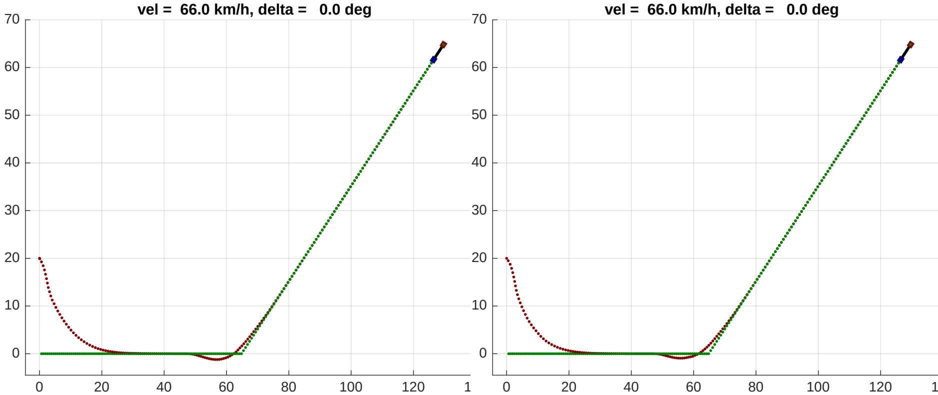
- Prediction horizon: $N = 30$ (prediction distance = $NT_s v$, for example 25 m at 60 km/h)

- Control horizon: $N_u = 4$

- Preview on reference signals available

AUTONOMOUS DRIVING EXAMPLE

- Closed-loop simulation results



Linear Parameter-Varying (LPV) MPC

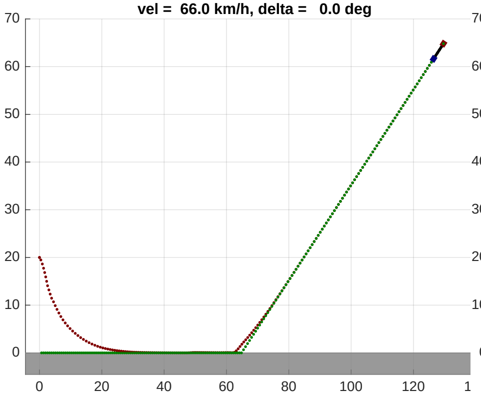
Model linearized @t and used @$t + k, \forall k$

Linear Time-Varying (LTV) MPC

Model linearized @$t + k, \forall k$

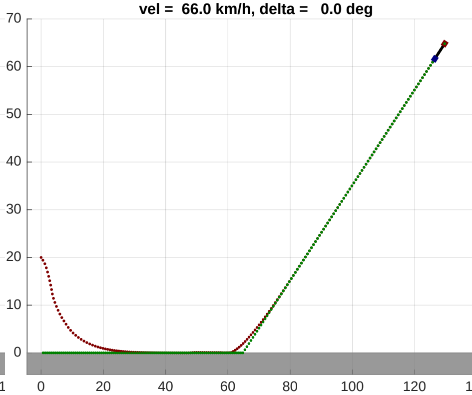
AUTONOMOUS DRIVING EXAMPLE

- Add position constraint $y \geq 0\,\mathrm{m}$



LPV-MPC

Model linearized @t



LTV-MPC

Model linearized @$t+k$, $k = 0, \ldots, N-1$

NONLINEAR MPC

NONLINEAR MPC

(Mayne, Rawlings, Diehl, 2017)

- Nonlinear prediction model

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

- Nonlinear constraints $h(x_k, u_k) \leq 0$
- Nonlinear performance index $\min \ \ell_N(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k)$
- Optimization problem: **nonlinear programming problem (NLP)**

$$\begin{array}{rl} \min_z & F(z, x(t)) \\ \text{s.t.} & G(z, x(t)) \leq 0 \\ & H(z, x(t)) = 0 \end{array} \qquad z = \begin{bmatrix} u_0 \\ \vdots \\ u_{N-1} \\ x_1 \\ \vdots \\ x_N \end{bmatrix}$$

NONLINEAR OPTIMIZATION

- (Nonconvex) NLP is harder to solve than QP
- Convergence to a **global optimum** may not be guaranteed
- Several NLP solvers exist (such as **Sequential Quadratic Programming (SQP)**)

  (Nocedal, Wright, 2006)
- NLP can be useful to deal with strong dynamical nonlinearities and/or nonlinear constraints/costs
- NL-MPC is less used in practice than linear MPC

FAST NONLINEAR MPC

(Lopez-Negrete, D'Amato, Biegler, Kumar, 2013)

- **Fast MPC**: exploit **sensitivity analysis** to compensate for the computational delay caused by solving the NLP
- **Key idea**: pre-solve the NLP between time $t-1$ and t based on the predicted state $x^*(t) = f(x(t-1), u(t-1))$ in background
- Get $u^*(t)$ and sensitivity $\left.\dfrac{\partial u^*}{\partial x}\right|_{x^*(t)}$ within sample interval $[(t-1)T_s, tT_s)$
- At time t, get $x(t)$ and compute

$$u(t) = u^*(t) + \frac{\partial u^*}{\partial x}(x(t) - x^*(t))$$

- A.k.a. **advanced-step MPC** (Zavala, Biegler, 2009)
- Note that still one NLP must be solved within the sample interval

FROM LTV-MPC TO NONLINEAR MPC

- Can we use the LTV-MPC machinery to handle nonlinear MPC ?

- **Key idea**: Solve a **sequence of LTV-MPC** problems at each time t

For $h = 0$ to $h_{\max} - 1$ do:

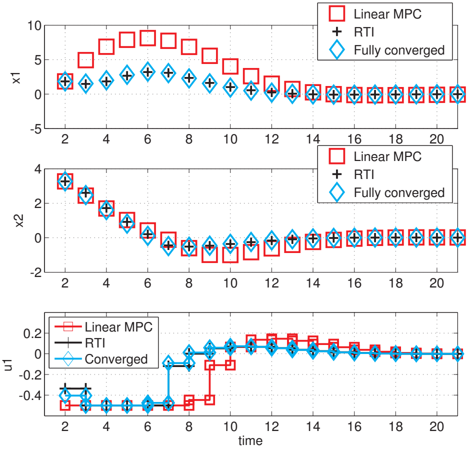
1. **Simulate** from $x(t)$ with inputs U_h and get state trajectory X_h
2. **Linearize** around (X_h, U_h) and **discretize** in time
3. Get U^*_{h+1} = **QP solution** of corresponding LTV-MPC problem
4. **Line search**: find optimal step size $\alpha_h \in (0, 1]$;
5. Set $U_{h+1} = (1 - \alpha_h)U_h + \alpha_h U^*_{h+1}$;

Return solution $U_{h_{\max}}$

- Special case: just solve one iteration with $\alpha = 1$ (a.k.a. **Real-Time Iteration**)
  (Diehl, Bock, Schloder, Findeisen, Nagy, Allgower, 2002) = LTV-MPC

(Gros, Zanon, Quirynen, Bemporad, Diehl, 2020)

- Example



OUTPUT FEEDBACK - EXTENDED KALMAN FILTER

- For **state estimation**, an **Extended Kalman Filter** (EKF) can be used based on the same nonlinear model (with additional noise)

$$\begin{array}{rcl} x(k+1) & = & f(x(k), u(k), \xi(k)) \\ y(k) & = & g(x(k)) + \zeta(k) \end{array}$$

- **measurement update**:

$$\begin{array}{rcl} C(k) & = & \frac{\partial g}{\partial x}(\hat{x}_{k|k-1}) \\ M(k) & = & P(k|k-1)C(k)'[C(k)P(k|k-1)C(k)' + R(k)]^{-1} \\ \text{consumed by MPC} \rightarrow \hat{x}(k|k) & = & \hat{x}(k|k-1) + M(k)\left(y(k) - g(\hat{x}(k|k-1))\right) \\ P(k|k) & = & (I - M(k)C(k))P(k|k-1) \end{array}$$

- **time update**:

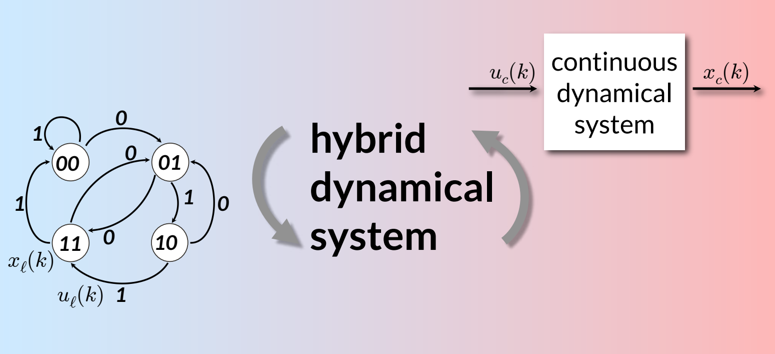
$$\hat{x}(k+1|k) = f(\hat{x}(k|k), u(k))$$

$$A(k) = \frac{\partial f}{\partial x}(\hat{x}_{k|k}, u(k), E[\xi(k)]), \; G(k) = \frac{\partial f}{\partial \xi}(\hat{x}_{k|k}, u(k), E[\xi(k)])$$

$$P(k+1|k) = A(k)P(k|k)A(k)' + G(k)Q(k)G(k)'$$

HYBRID MPC

HYBRID DYNAMICAL SYSTEMS

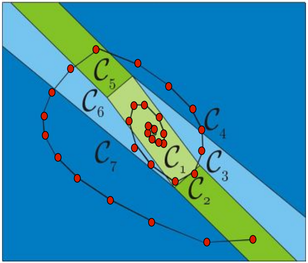


- Variables are **binary-valued**
  $x_\ell \in \{0,1\}^{n_\ell}$, $u_\ell \in \{0,1\}^{m_\ell}$
- Dynamics = **finite state machine**
- **Logic constraints**

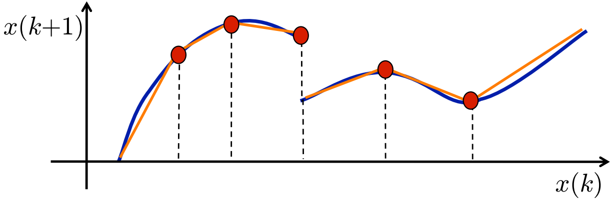
- Variables are **real-valued**
  $x_c \in \mathbb{R}^{n_c}$, $u_c \in \mathbb{R}^{m_c}$
- **Difference/differential equations**
- **Linear inequality** constraints

PIECEWISE AFFINE SYSTEMS

$$\begin{array}{rcl} x(k+1) & = & A_{i(k)}x(k) + B_{i(k)}u(k) + f_{i(k)} \\ y(k) & = & C_{i(k)}x(k) + D_{i(k)}u(k) + g_{i(k)} \\ \\ i(k) & \text{s.t.} & H_{i(k)}x(k) + J_{i(k)}u(k) \leq K_{i(k)} \end{array}$$

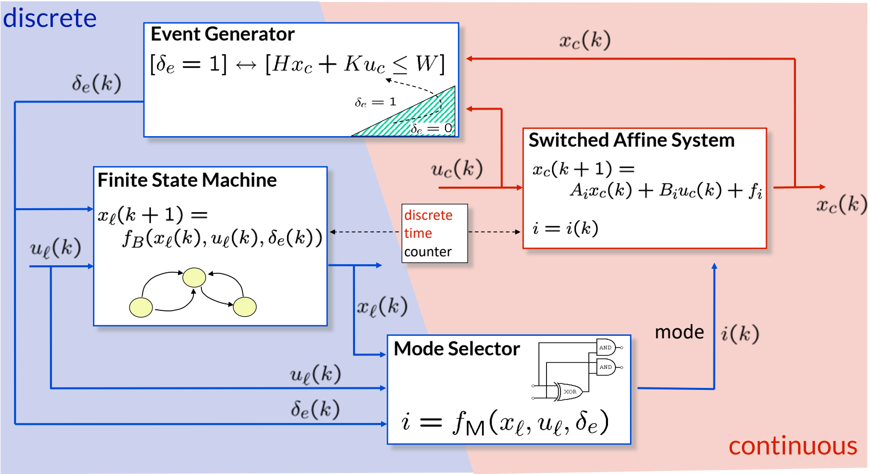


- PWA systems can approximate nonlinear dynamics arbitrarily well (even discontinuous ones)



DISCRETE HYBRID AUTOMATON (DHA)

(Torrisi, Bemporad, 2004)



$x_\ell \in \{0,1\}^{n_\ell}$ = **binary state** $\qquad x_c \in \mathbb{R}^{n_c}$ = **real-valued state**

$u_\ell \in \{0,1\}^{m_\ell}$ = **binary input** $\qquad u_c \in \mathbb{R}^{m_c}$ = **real-valued input**

$\delta_e \in \{0,1\}^{n_e}$ = **event variable** $\qquad i \in \{1,\ldots,s\}$ = **current mode**

CONVERSION OF LOGIC FORMULAS TO LINEAR INEQUALITIES

(Glover, 1975) (Williams, 1977) (Hooker, 2000)

- Key observation: $X_1 \vee X_2 = \texttt{true}$ $\Rightarrow$ $\delta_1 + \delta_2 \geq 1, \delta_1, \delta_2 \in \{0, 1\}$
- We want to impose the Boolean statement

$$F(X_1, \ldots, X_n) = \texttt{true}$$

- Convert the formula to **Conjunctive Normal Form (CNF)**

$$\bigwedge_{j=1}^{m} \left(\bigvee_{i \in P_j} X_i \bigvee_{i \in N_j} \bar{X}_i \right) = \texttt{true}, \quad P_j \cup N_j \subseteq \{1, \ldots, n\}$$

- Transform the CNF into the equivalent linear inequalities

$$\left\{ \begin{array}{rcl} \sum_{i \in P_1} \delta_i + \sum_{i \in N_1} (1 - \delta_i) & \geq & 1 \\ \vdots & & \vdots \\ \sum_{i \in P_m} \delta_i + \sum_{i \in N_m} (1 - \delta_i) & \geq & 1 \end{array} \right. \Rightarrow \quad A\delta \leq b, \ \delta \in \{0, 1\}^n$$

polyhedron

Any logic proposition can be translated into integer linear inequalities

BIG-M TECHNIQUE (IFF)

- Consider the **if-and-only-if** condition

$$[\delta = 1] \leftrightarrow [a'x_c - b \leq 0] \qquad \begin{array}{l} x_c \in \mathcal{X} \\ \delta \in \{0, 1\} \end{array}$$

- Assume $\mathcal{X} \subset \mathbb{R}^{n_c}$ bounded. Let M and m such that $\forall x_c \in \mathcal{X}$

$$\begin{array}{rcl} M & > & a'x_c - b \\ m & < & a'x_c - b \end{array}$$

- The if-and-only-if condition is equivalent to

$$\left\{ \begin{array}{rcl} a'x_c - b & \leq & M(1 - \delta) \\ a'x_c - b & > & m\delta \end{array} \right.$$

- We can replace the second constraint with $a'x_c - b \geq \epsilon + (m - \epsilon)\delta$ to avoid strict inequalities, where $\epsilon > 0$ is a small number (e.g., the machine precision)

BIG-M TECHNIQUE (IF-THEN-ELSE)

- Consider the **if-then-else** condition

$$z = \begin{cases} a_1' x_c - b_1 & \text{if } \delta = 1 \\ a_2' x_c - b_2 & \text{otherwise} \end{cases}$$

$$x_c \in \mathcal{X}$$
$$\delta \in \{0,1\}$$
$$z \in \mathbb{R}$$

- Assume $\mathcal{X} \subset \mathbb{R}^{n_c}$ bounded. Let M_1, M_2 and m_1, m_2 such that $\forall x_c \in \mathcal{X}$

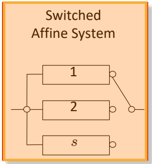
$$\begin{array}{ccccc} M_1 & > & a_1' x_c - b_1 & > & m_1 \\ M_2 & > & a_2' x_c - b_2 & > & m_2 \end{array}$$

- The if-then-else condition is equivalent to

$$\begin{cases} (m_1 - M_2)(1-\delta) + z & \leq & a_1' x_c - b_1 \\ (m_2 - M_1)(1-\delta) - z & \leq & -(a_1' x_c - b_1) \\ (m_2 - M_1)\delta + z & \leq & a_2' x_c - b_2 \\ (m_1 - M_2)\delta - z & \leq & -(a_2' x_c - b_2) \end{cases}$$

SWITCHED AFFINE SYSTEM

- The state-update equation of a SAS can be rewritten as



$$x_c(k+1) = \sum_{i=1}^{s} z_i(k) \qquad z_i(k) \in \mathbb{R}^{n_c}$$

with

$$z_1(k) = \begin{cases} A_1 x_c(k) + B_1 u_c(k) + f_1 & \text{if } \delta_1(k) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\vdots$$

$$z_s(k) = \begin{cases} A_s x_c(k) + B_s u_c(k) + f_s & \text{if } \delta_s(k) = 1 \\ 0 & \text{otherwise} \end{cases}$$

and with $\delta_i(k) \in \{0,1\}$ subject to the **exclusive or** condition

$$\sum_{i=1}^{s} \delta_i(k) = 1 \text{ or equivalently } \begin{cases} \sum_{i=1}^{s} \delta_i(k) & \geq & 1 \\ \sum_{i=1}^{s} \delta_i(k) & \leq & 1 \end{cases}$$

- Output eqs $y_c(k) = C_i x_c(k) + D_i u_c(k) + g_i$ admit similar transformation

$X_1 \vee X_2 =$TRUE $\longrightarrow$ $\delta_1 + \delta_2 \geq 1, \quad \delta_1, \delta_2 \in \{0,1\}$

Any logic statement

$f(X) =$TRUE

$\bigwedge_{j=1}^{m} \left(\vee_{i \in P_j} X_i \vee_{i \in N_j} \neg X_i \right)$ (CNF)

$N_j, P_j \subseteq \{1, \ldots, n\}$

$$\left\{ \begin{array}{l} 1 \leq \sum_{i \in P_1} \delta_i + \sum_{i \in N_1} (1 - \delta_i) \\ \vdots \\ 1 \leq \sum_{i \in P_m} \delta_i + \sum_{i \in N_m} (1 - \delta_i) \end{array} \right.$$

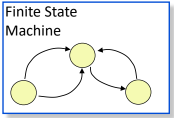
$[\delta_e^i(k) = 1] \leftrightarrow [H^i x_c(k) \leq W^i]$

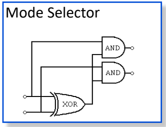
$$\left\{ \begin{array}{l} H^i x_c(k) - W^i \leq M^i (1 - \delta_e^i(k)) \\ H^i x_c(k) - W^i > m^i \delta_e^i(k) \end{array} \right.$$

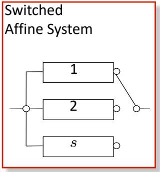
IF $[\delta = 1]$ THEN $z = a_1^T x + b_1^T u + f_1$

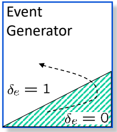
ELSE $z = a_2^T x + b_2^T u + f_2$

$$\left\{ \begin{array}{rcl} (m_1 - M_2)(1 - \delta) + z & \leq & a_1 x + b_1 u + f_1 \\ (m_2 - M_1)(1 - \delta) - z & \leq & -a_1 x - b_1 u - f_1 \\ (m_2 - M_1)\delta + z & \leq & a_2 x + b_2 u + f_2 \\ (m_1 - M_2)\delta - z & \leq & -a_2 x - b_2 u - f_2 \end{array} \right.$$





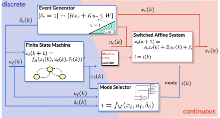




MIXED LOGICAL DYNAMICAL (MLD) SYSTEMS

(Bemporad, Morari, 1999)

- By converting logic relations into mixed-integer linear inequalities a DHA can be rewritten as the **Mixed Logical Dynamical (MLD)** system



$$
\begin{cases}
x(k+1) = Ax(k) + B_1 u(k) + B_2\delta(k) + B_3 z(k) + B_5 \\
y(k) = Cx(k) + D_1 u(k) + D_2\delta(k) + D_3 z(k) + D_5 \\
E_2\delta(k) + E_3 z(k) \le E_4 x(k) + E_1 u(k) + E_5
\end{cases}
$$

$$
x \in \mathbb{R}^{n_c} \times \{0,1\}^{n_b},\ u \in \mathbb{R}^{m_c} \times \{0,1\}^{m_b}
$$
$$
y \in \mathbb{R}^{p_c} \times \{0,1\}^{p_b},\ \delta \in \{0,1\}^{r_b},\ z \in \mathbb{R}^{r_c}
$$

- The translation from DHA to MLD can be automatized, see e.g. the language **HYSDEL** (HYbrid Systems DEscription Language) (Torrisi, Bemporad, 2004)

- MLD models allow solving MPC, verification, state estimation, and fault detection problems via **mixed-integer programming**

EQUIVALENCE OF HYBRID MODELS

- **MLD** and **PWA** systems are equivalent (Bemporad, Ferrari-Trecate, Morari, 2000)

  Proof: For a given combination (x_ℓ, u_ℓ, δ) of an MLD model, the state and output equation are linear and valid in a polyhedron.

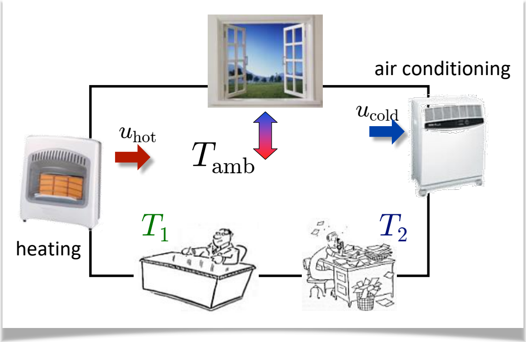
  Conversely, a PWA system can be modeled as MLD system (see next slide)

- Efficient **conversion algorithms** from MLD to PWA form exist

  (Bemporad, 2004) (Geyer, Torrisi, Morari, 2003)

- Further equivalences exist with other classes of hybrid dynamical systems, such as **Linear Complementarity (LC)** systems (Heemels, De Schutter, Bemporad, 2001)

EXAMPLE: ROOM TEMPERATURE CONTROL



discrete dynamics

- #1 = cold $\rightarrow$ heater = on
- #2 = cold $\rightarrow$ heater = on **unless** #1 hot
- A/C activation has similar rules

continuous dynamics

$$\frac{dT_i}{dt} = -\alpha_i(T_i - T_{\rm amb}) + k_i(u_{\rm hot} - u_{\rm cold})$$

$i = 1, 2$

go to demo `demos/hybrid/heatcool.m`

EXAMPLE: ROOM TEMPERATURE CONTROL

```
SYSTEM heatcool {

INTERFACE {
    STATE { REAL T1 [-10,50];
            REAL T2 [-10,50];
          }
    INPUT { REAL Tamb [-10,50];
          }
    PARAMETER {
        REAL Ts, alpha1, alpha2, k1, k2;
        REAL Thot1, Tcold1, Thot2, Tcold2, Uc, Uh;
          }
}
IMPLEMENTATION {
        AUX { REAL uhot, ucold;
              BOOL hot1, hot2, cold1, cold2;
          }
        AD  { hot1 = T1>=Thot1;
              hot2 = T2>=Thot2;
              cold1 = T1<=Tcold1;
              cold2 = T2<=Tcold2;
          }
        DA  { uhot = {IF cold1 | (cold2 & ~hot1) THEN Uh ELSE 0};
              ucold = {IF hot1 | (hot2 & ~cold1) THEN Uc ELSE 0};
          }
        CONTINUOUS { T1 = T1+Ts*(-alpha1*(T1-Tamb)+k1*(uhot-ucold));
                     T2 = T2+Ts*(-alpha2*(T2-Tamb)+k2*(uhot-ucold));
          }
      }
}
```

```
>> S=mld('heatcoolmodel',Ts);
```

get the MLD model in MATLAB

```
>> [XX,TT]=sim(S,x0,U);
```

simulate the MLD model

EXAMPLE: ROOM TEMPERATURE CONTROL

- MLD model of the room temperature system

$$\begin{cases} x(k+1) &=& Ax(k) + B_1u(k) + B_2\delta(k) + B_3z(k) + B_5 \\ y(k) &=& Cx(k) + D_1u(k) + D_2\delta(k) + D_3z(k) + D_5 \\ E_2\delta(k) &+& E_3z(k) \leq E_4x(k) + E_1u(k) + E_5 \end{cases}$$

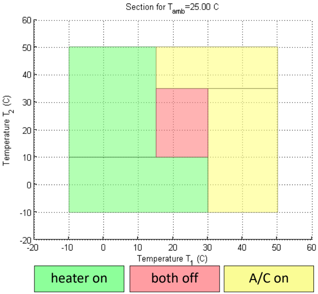
  - 2 continuous states (temperature T_1, T_2)
  - 1 continuous input (room temperature T_{amb})
  - 2 auxiliary continuous vars (power flows $u_{\text{hot}}, u_{\text{cold}}$)
  - 6 auxiliary binary vars (4 threshold events + 2 for the OR condition)
  - 20 mixed-integer inequalities

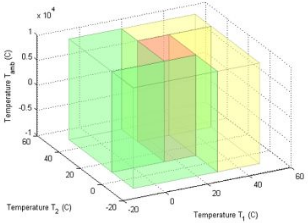
- In principle, we have $2^6 = 64$ possible combinations of binary variables

- PWA model of the room temperature system

$$\begin{aligned} x(k+1) &= A_{i(k)}x(k) + B_{i(k)}u(k) + f_{i(k)} \\ y(k) &= C_{i(k)}x(k) + D_{i(k)}u(k) + g_{i(k)} \\ i(k) \quad \text{s.t.} \quad & H_{i(k)}x(k) + J_{i(k)}u(k) \le K_{i(k)} \end{aligned}$$

```
>> P=pwa(S);
```





5 polyhedral regions

(partition does not depend on input)

2 continuous states (T_1, T_2)

1 continuous input $(T_{\rm amb})$

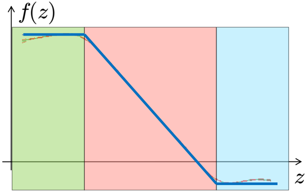
IDENTIFICATION OF HYBRID SYSTEMS

PWA REGRESSION PROBLEM

- **Problem**: Given input/output pairs $\{x(k), y(k)\}, k = 1, \ldots, N$ and number s of models, compute a **piecewise affine** (PWA) approximation $y \approx f(x)$

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

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



- Need to learn **both** the parameters $\{F_i, g_i\}$ of the affine submodels **and** the partition $\{H_i, K_i\}$ of the PWA map from data (**offline learning**)

- Possibly update model+partition as new data are available (**online learning**)

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

APPROACHES TO PWA IDENTIFICATION

- Mixed-integer linear or quadratic programming (Roll, Bemporad, Ljung, 2004)
- Partition of infeasible set of inequalities (Bemporad, Garulli, Paoletti, Vicino, 2005)
- K-means clustering in a feature space (Ferrari-Trecate, Muselli, Liberati, Morari, 2003)
- Bayesian approach (Juloski, Wieland, Heemels, 2004)
- Kernel-based approaches (Pillonetto, 2016)
- Hyperplane clustering in data space (Münz, Krebs, 2002)
- **Recursive multiple least squares & PWL separation** (Breschi, Piga, Bemporad, 2016)
- **Piecewise affine regression and classification** (PARC) (Bemporad, 2021)

PWA REGRESSION ALGORITHM

(Breschi, Piga, Bemporad, 2016)

1. Estimate models $\{F_i,\ g_i\}$ **recursively**. Let $e_i(k) = y(k) - F_i x(k) - g_i$ and only update model $i(k)$ such that

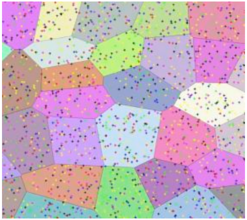
$$i(k) \leftarrow \arg\min_{i=1,\ldots,s} \quad \underbrace{e_i(k)' \Lambda_e^{-1} e_i(k)}_{\text{one-step prediction error of model \#i}} \quad + \underbrace{(x(k) - c_i)' R_i^{-1} (x(k) - c_i)}_{\text{proximity to centroid of cluster \#i}}$$

using **recursive LS** and **inverse QR decomposition** (Alexander, Ghirnikar, 1993)

This also splits the data points $x(k)$ in **clusters** $C_i = \{x(k) :\ i(k) = i\}$

2. Compute a polyhedral partition $\{H_i,\ K_i\}$ of the regressor space via **multi-category linear separation**

$$\phi(x) = \max_{i=1,\ldots,s} \{w_i' x - \gamma_i\}$$



(Breschi, Piga, Bemporad, 2016)

- Identification of **piecewise-affine ARX** model

$$\begin{bmatrix} y_1(k) \\ y_2(k) \end{bmatrix} = \begin{bmatrix} -0.83 & 0.20 \\ 0.30 & -0.52 \end{bmatrix} \begin{bmatrix} y_1(k-1) \\ y_2(k-1) \end{bmatrix} + \begin{bmatrix} -0.34 & 0.45 \\ -0.30 & 0.24 \end{bmatrix} \begin{bmatrix} u_1(k-1) \\ u_2(k-1) \end{bmatrix}$$
$$+ \begin{bmatrix} 0.20 \\ 0.15 \end{bmatrix} + \max \left\{ \begin{bmatrix} 0.20 & -0.90 \\ 0.10 & -0.42 \end{bmatrix} \begin{bmatrix} y_1(k-1) \\ y_2(k-1) \end{bmatrix} \right.$$
$$\left. + \begin{bmatrix} 0.42 & 0.20 \\ 0.50 & 0.64 \end{bmatrix} \begin{bmatrix} u_1(k-1) \\ u_2(k-1) \end{bmatrix} + \begin{bmatrix} 0.40 \\ 0.30 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\} + e_o(k),$$

- **Quality of fit**: best fit rate (BFR) = $\max \left\{ 1 - \frac{\|y_{o,i} - \hat{y}_i\|_2}{\|y_{o,i} - \bar{y}_{o,i}\|_2}, 0 \right\}, i = 1, 2$

  y_o = measured, $\hat{y}$ = open-loop simulated, $\bar{y}$ = sample mean of y_o

		$N = 4000$	$N = 20000$	$N = 100000$
y_1	(offline) RLP	96.0 %	96.5 %	99.0 %
	(Offline) RPSN	96.2 %	96.4 %	98.9 %
	(Online) ASGD	86.7 %	95.0 %	96.7 %
y_2	(offline) RLP	96.2 %	96.9 %	99.0 %
	(offline) RPSN	96.3 %	96.8 %	99.0 %
	(online) ASGD	87.4 %	95.2 %	96.4 %

BFR on validation data, open-loop validation

RLP = Robust linear programming
(Bennett, Mangasarian, 1994)

RPSN = Piecewise-smooth Newton method
(Bemporad, Bernardini, Patrinos, 2015)

ASGD = Averaged stochastic gradient descent
(Bottou, 2012)

- **CPU time for computing the partition**:

	$N = 4000$	$N = 20000$	$N = 100000$
(Offline) RLP	0.308 s	3.227 s	112.435 s
(Offline) RPSN	0.016 s	0.086 s	0.365 s
(Online) ASGD	0.013 s	0.023 s	0.067 s

PARC - PIECEWISE AFFINE REGRESSION AND CLASSIFICATION

(Bemporad, 2021)

- 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**

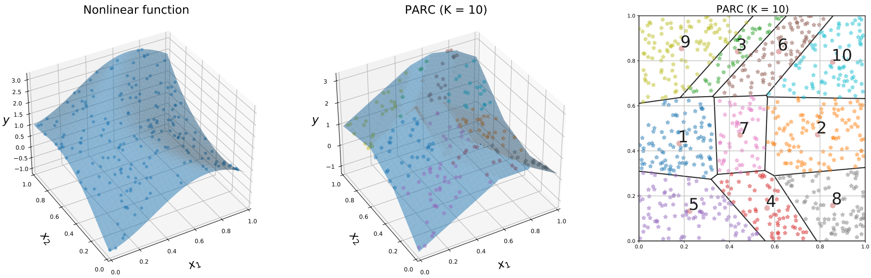
$$\Phi(z) = \omega^{j(z)} z + \gamma^{j(z)}, \qquad j(z) = \min\left\{\arg\max_{j=1,\ldots,K}\{\omega^j z + \gamma^j\}\right\}$$

- 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

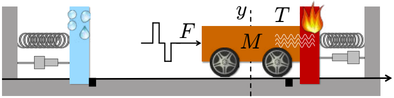
(Bemporad, 2021)

- 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/`

PARC - CART & BUMPERS EXAMPLE



- **Example**: moving cart and bumpers + heat transfer during bumps.

  Spring and viscous forces are **nonlinear**.

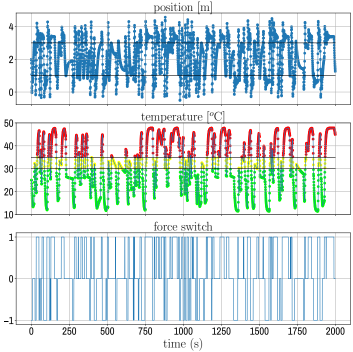
- Categorical input $F \in \{-\bar{F}, 0, \bar{F}\}$ and categorical output $c \in \{green, yellow, red\}$

- Continuous-time system simulated for 2,000 s, sample time = 0.5 s (=4000 training samples)

- Feature vector $z_k = [y_k, \dot{y}_k, T_k, F_k]$

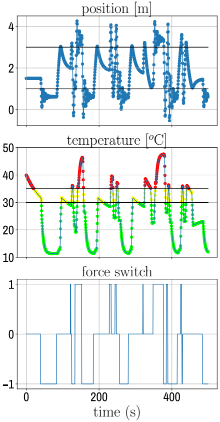
- Target vector $v_k = [y_{k+1}, \dot{y}_{k+1}, T_{k+1}, c_k]$

- Hybrid model learned by **PARC** ($K = 5$ regions)

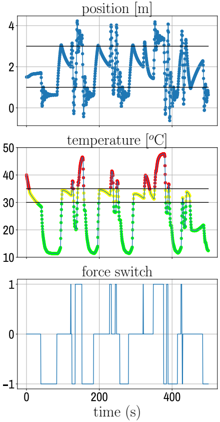


PARC - CART & BUMPERS EXAMPLE

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



continuous-time system



discrete-time PWA model

- Model fit is good enough for MPC design purposes (see later ...)

HYBRID MODEL PREDICTIVE CONTROL

MIQP FORMULATION OF HYBRID MPC

(Bemporad, Morari, 1999)

- Finite-horizon optimal control problem (regulation)

$$
\begin{array}{ll}
\min & \displaystyle\sum_{k=0}^{N-1} y_k' Q y_k + u_k' R u_k \\
\text{s.t.} & \left\{\begin{array}{rcl}
x_{k+1} & = & Ax_k + B_1 u_k + B_2 \delta_k + B_3 z_k + B_5 \\
y_k & = & Cx_k + D_1 u_k + D_2 \delta_k + D_3 z_k + D_5 \\
E_2 \delta_k & + & E_3 z_k \leq E_4 x_k + E_1 u_k + E_5 \\
x_0 & = & x(t)
\end{array}\right.
\end{array}
$$

$Q = Q' \succ 0, R = R' \succ 0$

- Treat u_k, δ_k, z_k as free decision variables, $k = 0, \ldots, N-1$

- Predictions can be constructed **exactly as in the linear case**

$$
x_k = A^k x_0 + \sum_{j=0}^{k-1} A^j (B_1 u_{k-1-j} + B_2 \delta_{k-1-j} + B_3 z_{k-1-j} + B_5)
$$

MIQP FORMULATION OF HYBRID MPC

(Bemporad, Morari, 1999)

- After substituting x_k, y_k the resulting optimization problem becomes the following **Mixed-Integer Quadratic Programming (MIQP)** problem

$$\begin{array}{rl} \min_\xi & \frac{1}{2}\xi' H\xi + x'(t)F'\xi + \frac{1}{2}x'(t)Yx(t) \\ \text{s.t.} & G\xi \leq W + Sx(t) \end{array}$$

- The optimization vector $\xi = [u_0, \ldots, u_{N-1}, \delta_0, \ldots, \delta_{N-1}, z_0, \ldots, z_{N-1}]$ has **mixed real and binary** components

$$\begin{array}{l} u_k \in \mathbb{R}^{m_c} \times \{0,1\}^{m_b} \\ \delta_k \in \{0,1\}^{r_b} \\ z_k \in \mathbb{R}^{r_c} \end{array} \quad \Longrightarrow \quad \xi \in \mathbb{R}^{N(m_c+r_c)} \times \{0,1\}^{N(m_b+r_b)}$$

- **Closed-loop convergence**, **asymptotic stability** can be guaranteed by terminal cost/constraints (Bemporad, Morari, 1999) (Lazar, Heemels, Weiland, Bemporad, 2006)

(Bemporad, Borrelli, Morari, 2000)

- Finite-horizon optimal control problem using infinity norms

$$\min_{\xi} \quad \sum_{k=0}^{N-1} \|Qy_k\|_\infty + \|Ru_k\|_\infty$$

$$\text{s.t.} \quad \begin{cases} x_{k+1} & = & Ax_k + B_1u_k + B_2\delta_k + B_3z_k + B_5 \\ y_k & = & Cx_k + D_1u_k + D_2\delta_k + D_3z_k + D_5 \\ E_2\delta_k & + & E_3z_k \leq E_4x_k + E_1u_k + E_5 \\ x_0 & = & x(t) \end{cases}$$

$Q \in \mathbb{R}^{m_y \times n_y}$
$R \in \mathbb{R}^{m_u \times n_u}$

- Introduce additional variables $\epsilon_k^y, \epsilon_k^u, k = 0, \ldots, N-1$

$$\begin{cases} \epsilon_k^y & \geq & \|Qy_k\|_\infty \\ \epsilon_k^u & \geq & \|Ru_k\|_\infty \end{cases} \Rightarrow \begin{cases} \epsilon_k^y & \geq & \pm Q^i y_k \\ \epsilon_k^u & \geq & \pm R^i u_k \end{cases}$$

Q^i = ith row of Q

MILP FORMULATION OF HYBRID MPC

(Bemporad, Borrelli, Morari, 2000)

- After substituting x_k, y_k the resulting optimization problem becomes the following **Mixed-Integer Linear Programming (MILP)** problem

$$\begin{array}{ll} \min_\xi & \displaystyle\sum_{k=0}^{N-1} \epsilon_k^y + \epsilon_k^u \\ \text{s.t.} & G\xi \leq W + Sx(t) \end{array}$$

- $\xi = [u_0, \ldots, u_{N-1}, \delta_0, \ldots, \delta_{N-1}, z_0, \ldots, z_{N-1}, \epsilon_0^y, \epsilon_0^u, \ldots, \epsilon_{N-1}^y, \epsilon_{N-1}^u]$ is the optimization vector, with **mixed real and binary** components

$$\begin{array}{l} u_k \in \mathbb{R}^{m_c} \times \{0,1\}^{m_b} \\ \delta_k \in \{0,1\}^{r_b} \\ z_k \in \mathbb{R}^{r_c} \\ \epsilon_k^y, \epsilon_k^u \in \mathbb{R} \end{array} \quad \Longrightarrow \quad \xi \in \mathbb{R}^{N(m_c+r_c+2)} \times \{0,1\}^{N(m_b+r_b)}$$

- Same approach applies to any **convex piecewise affine** stage cost

HYBRID MPC – TEMPERATURE CONTROL

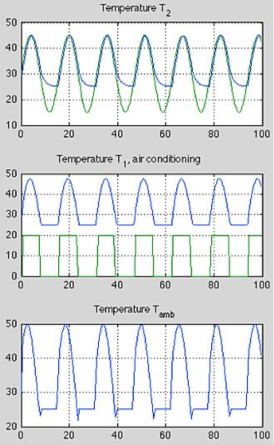
```
>> refs.x=2;            % just weight state #2
>> Q.x=1;               % unit weight on state #2
>> Q.rho=Inf;           % hard constraints
>> Q.norm=Inf;          % infinity norms
>> N=2;                 % prediction horizon
>> limits.xmin=[25;-Inf];
```

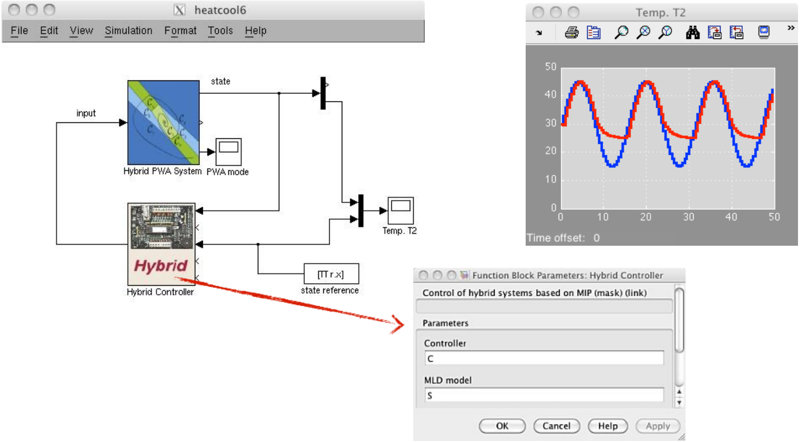
```
>> C=hybcon(S,Q,N,limits,refs);
```

```
>> C

Hybrid controller based on MLD model S <heatcoolmodel.hys> [Inf-norm]

2 state measurement(s)
0 output reference(s)
0 input reference(s)
1 state reference(s)
0 reference(s) on auxiliary continuous z-variables

20 optimization variable(s) (8 continuous, 12 binary)
46 mixed-integer linear inequalities
sampling time = 0.5, MILP solver = 'glpk'

Type "struct(C)" for more details.
>>
```

```
>> [XX,UU,DD,ZZ,TT]=sim(C,S,r,x0,Tstop);
```

$$\min \sum_{k=1}^{2} \|x_{2k} - r(t)\|_\infty$$
$$\text{s.t.} \begin{cases} x_{1k} \geq 25, \; k = 1,2 \\ \text{MLD model} \end{cases}$$





- Average CPU time to solve MILP: $\approx$ 1 ms/step

  (Macbook Pro 3GHz Intel Core i7 using GLPK)

MIXED-INTEGER PROGRAMMING SOLVERS

- Binary constraints make Mixed-Integer Programming (MIP) a hard problem ($\mathcal{NP}$-complete)

- However, excellent general purpose **branch & bound** / **branch & cut** solvers available for MILP and MIQP (Gurobi, CPLEX, FICO Xpress, GLPK, CBC, ...)

  (more solvers/benchmarks: see `http://plato.la.asu.edu/bench.html`)

- MIQP approaches tailored to embedded hybrid MPC applications:
  - B&B + (dual) active set methods for QP

    (Leyffer, Fletcher, 1998) (Axehill, Hansson, 2006) (Bemporad, 2015) (Bemporad, Naik, 2018)
  - B&B + interior point methods: (Frick, Domahidi, Morari, 2015)
  - B&B + fast gradient projection: (Naik, Bemporad, 2017)
  - B&B + ADMM: (Stellato, Naik, Bemporad, Goulart, Boyd, 2018)

- No need to reach global optimum (see convergence proof), although performance may deteriorate

BRANCH & BOUND METHOD FOR MIQP

(Dakin, 1965)

- We want to solve the following MIQP

$$\begin{array}{rl} \min & V(z) \triangleq \frac{1}{2}z'Qz + c'z \\ \text{s.t.} & Az \leq b \\ & z_i \in \{0,1\},\ \forall i \in I \end{array} \qquad \begin{array}{l} z \in \mathbb{R}^n \\ Q = Q' \succeq 0 \\ I \subseteq \{1,\ldots,n\} \end{array}$$

- **Branch & Bound (B&B)** is the simplest (and most popular) approach to solve the problem to optimality
- **Key idea**:
  - for each binary variable $z_i, i \in I$, either set $z_i = 0$, or $z_i = 1$, or $z_i \in [0,1]$
  - solve the corresponding **QP relaxation** of the MIQP problem
  - use QP result to decide the next combination of fixed/relaxed variables

FAST GRADIENT PROJECTION FOR MIQP

(Naik, Bemporad, 2017)

- Consider again the MIQP problem with Hessian $Q = Q' \succ 0$

$$
\begin{aligned}
\min_{z} \quad & V(z) \triangleq \frac{1}{2} z'Qz + c'z \\
\text{s.t.} \quad & \ell \leq Az \leq u \\
& Gz = g \\
& \bar{A}_i z \in \{\bar{\ell}_i, \bar{u}_i\},\ i = 1, \dots, p
\end{aligned}
$$

$$
\begin{aligned}
w^k &= y^k + \beta_k (y^k - y^{k-1}) \\
z^k &= -Kw^k - Jx \\
s^k &= \dots \\
y_i^{k+1} &= \max\left\{w_i^k + s_i^k, 0\right\},\ i \in I_{\text{ineq}}
\end{aligned}
$$

- Use B&B and **fast gradient projection** to solve dual of QP relaxation

constraint is relaxed	$\bar{A}_i z \leq \bar{u}_i$	$\rightarrow$	$y_i^{k+1} = \max\left\{w_i^k + s_i^k, 0\right\}$	$(y_i \geq 0)$
constraint is fixed	$\bar{A}_i z = \bar{u}_i$	$\rightarrow$	$y_i^{k+1} = w_i^k + s_i^k$	$(y_i \lessgtr 0)$
constraint is ignored	$\bar{A}_i z = \bar{\ell}_i$	$\rightarrow$	$y_i^{k+1} = 0$	$(y_i = 0)$

FAST GRADIENT PROJECTION FOR MIQP

(Naik, Bemporad, 2017)

- **Same dual QP matrices** at each node, **preconditioning** computed only once
- **Warm-start** exploited, **dual cost** used to stop QP relaxations earlier
- Criterion based on Farkas lemma to detect **QP infeasibility**
- Numerical results (time in ms):

n	m	p	q	miqpGPAD	GUROBI
10	100	2	2	15.6	6.56
50	25	5	3	3.44	8.74
50	150	10	5	63.22	46.25
100	50	2	5	6.22	26.24
100	200	15	5	164.06	188.42
150	100	5	5	31.26	88.13
150	200	20	5	258.80	274.06
200	50	15	6	35.08	144.38

n = # variables
m = # inequality constraints
p = # binary constraints
q = # equality constraints

CPU time measured on Intel Core i7-4700MQ CPU 2.40 GHz

EXPLICIT HYBRID MPC (MLD FORMULATION)

$$\min_\xi J(\xi, x(t)) = \sum_{k=0}^{N-1} \|Qy_k\|_\infty + \|Ru_k\|_\infty$$

$$\text{subject to} \quad \left\{ \begin{array}{rcl} x_{k+1} & = & Ax_k + B_1 u_k + B_2 \delta_k + B_3 z_k + B_5 \\ y_k & = & Cx_k + D_1 u_k + D_2 \delta_k + D_3 z_k + D_5 \\ E_2 \delta_k + E_3 z_k & \leq & E_4 x_k + E_1 u_k + E_5 \\ x_0 & = & x(t) \end{array} \right.$$

- **Online optimization**: solve the problem for a **given state** $x(t)$ as the **MILP**

$$\begin{array}{ll} \min_\xi & \displaystyle\sum_{k=0}^{N-1} \epsilon_k^y + \epsilon_k^u \\ \text{s.t.} & G\xi \leq W + S x(t) \end{array}$$

- **Offline optimization**: solve the MILP in advance **for all states** $x(t)$

  → **multiparametric Mixed-Integer Linear Program (mp-MILP)**

MULTIPARAMETRIC MILP

- Consider the mp-MILP

$$\begin{array}{rl} \min_{\xi_c,\xi_d} & f_c'\xi_c + f_d'\xi_d \\ \text{s.t.} & G_c\xi_c + G_d\xi_d \le W + Sx \end{array} \qquad \begin{array}{l} \xi_c \in \mathbb{R}^{n_c} \\ \xi_d \in \{0,1\}^{n_d} \\ x \in \mathbb{R}^m \end{array}$$

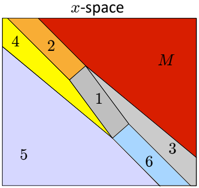
- A mp-MILP can be solved by alternating MILPs and mp-LPs

  (Dua, Pistikopoulos, 1999)

- The multiparametric solution $\xi^*(x)$ is **PWA** (but possibly discontinuous)

- The MPC controller is piecewise affine in $x = x(t)$

$$u(x) = \begin{cases} F_1x + g_1 & \text{if} \quad H_1x \le K_1 \\ \vdots & \vdots \\ F_Mx + g_M & \text{if} \quad H_Mx \le K_M \end{cases}$$



(More generally, the parameter vector x includes states and reference signals)

EXPLICIT HYBRID MPC (PWA FORMULATION)

- Consider the MPC formulation using a PWA prediction model

$$
\begin{array}{rl}
\min_\xi J(\xi, x(t)) & = \displaystyle\sum_{k=0}^{N-1} \|Qy_k\|_\infty + \|Ru_k\|_\infty \\
\text{subject to} & \left\{\begin{array}{rcl}
x_{k+1} & = & A_{i(k)}x_k + B_{i(k)}u_k + f_{i(k)} \\
y_k & = & C_{i(k)}x_k + D_{i(k)}u_k + g_{i(k)} \\
 & & i(k) \text{ such that } H_{i(k)}x_k + W_{i(k)}u_k \leq K_{i(k)} \\
x_0 & = & x(t)
\end{array}\right.
\end{array}
$$

- **Method #1**: The explicit solution can be obtained by using a combination of **dynamic programming (DP)** and **mpLP** (Borrelli, Baotic, Bemporad, Morari, 2005)

- Clearly the explicit hybrid MPC law is again piecewise affine, as PWA systems $\equiv$ MLD systems

EXPLICIT HYBRID MPC (PWA FORMULATION)

- **Method #2:** (Bemporad, Hybrid Toolbox, 2003)

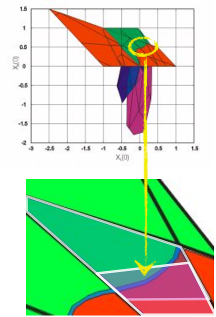
  (Alessio, Bemporad, 2006) (Mayne, ECC 2001) (Mayne, Rakovic, 2002)

  1 Use backwards (=DP) **reachability analysis** for enumerating all feasible mode sequences $I = \{i(0), i(1), \ldots, i(N)\}$

  2 For each fixed sequence I, solve the explicit finite-time optimal control problem for the corresponding linear time-varying system (**mpQP** or **mpLP**)

  3a **Case of 1 / ∞-norms** or **convex PWA costs**: Compare value functions and **split regions**

  3b **Case of quadratic costs**: the partition may not be fully polyhedral, better **keep overlapping polyhedra** and compare online quadratic cost functions when overlaps are detected

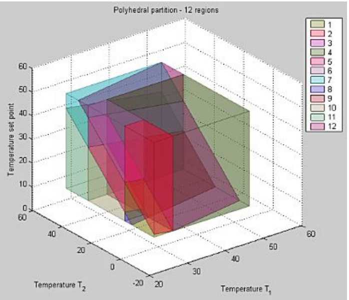


- Comparison of quadratic costs can be avoided by lifting the parameter space (Fuchs, Axehill, Morari, 2015)

EXPLICIT HYBRID MPC – TEMPERATURE CONTROL

```
>> E=expcon(C,range,options);
```

```
>> E

Explicit controller (based on hybrid controller C)
3 parameter(s)
1 input(s)
12 partition(s)
sampling time = 0.5

The controller is for hybrid systems (tracking)
This is a state-feedback controller.

Type "struct(E)" for more details.
>>
```

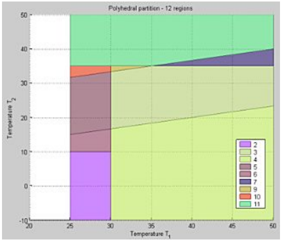


384 numbers to store in memory

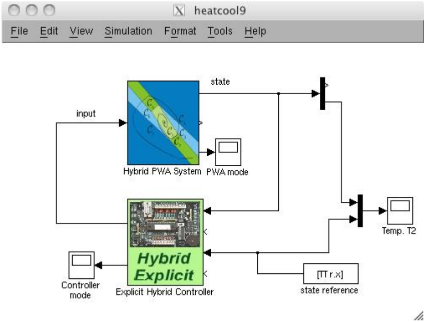
$$\min \sum_{k=0}^{2} \|x_{2k} - r(t)\|_\infty$$

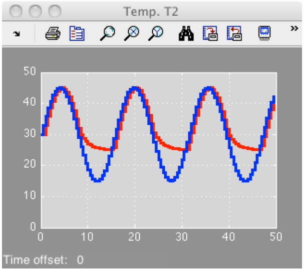
$$\text{s.t.} \begin{cases} x_{1k} \geq 25, \ k = 1, 2 \\ \text{hybrid model} \end{cases}$$

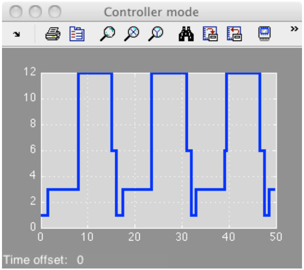
(T_1, T_2) section for $T_{\rm ref} = 30$



EXPLICIT HYBRID MPC – TEMPERATURE CONTROL







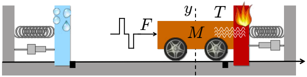
generated C-code

utils/expcon.h

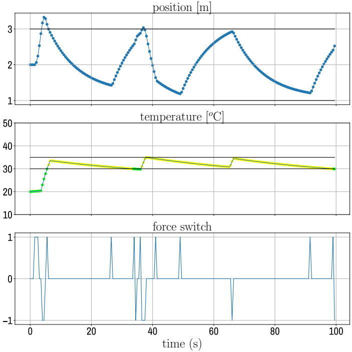
```
#define EXPCON_REG 12
#define EXPCON_NTH 3
#define EXPCON_NYM 2
#define EXPCON_NH 72
#define EXPCON_NF 12
static double EXPCON_F[]={
    -1,0,0,0,-1,0,
    -1,-1,-1,-1,-1,-1,0,-3,-3,
    -3,0,-3,0,0,0,0,0,
    0,0,4,4,4,0,4,0,0,
    0,0,0,0};

static double EXPCON_G[]={
    101.6,1.6,1.6,-1.6,98.4001,0,100,51.6,
    101.6,51.6,48.4,50};

static double EXPCON_H[]={
    0,0,0,-0.00999999,0,-0.0333333,
    0.02,0.00999999,-0.02,0,0,-0.0333333,0.02,0.00999999,
    0,0,-0.02,0.02,0,-1,0.00999999,0,
```

PARC - CART & BUMPERS EXAMPLE

- MPC problem with prediction horizon $N = 9$:



$$\begin{array}{ll} \min_{F_0,\ldots,F_{N-1}} & \displaystyle\sum_{k=0}^{N-1} |c_k - 1| + 0.25|F_k| \\ \text{s.t.} & F_k \in \{-\bar{F}, 0, \bar{F}\} \\ & \text{PWA model equations} \end{array}$$

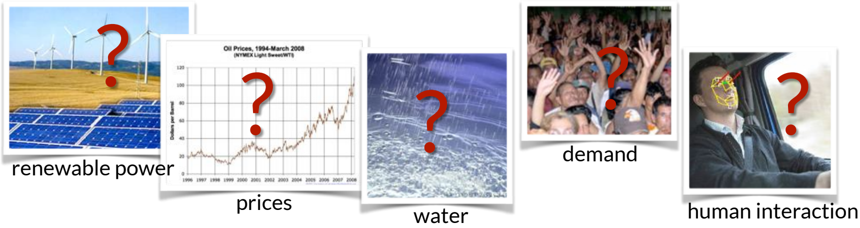


- MILP solution time: 0.15-0.29 s (CPLEX)
- **Data-driven hybrid MPC** controller can keep temperature in yellow zone
- **Approximate explicit MPC**: fit a **decision tree** on 10,000 samples (accuracy: 99.9%). CPU time = 52÷67 μs. Closed-loop trajectories very similar.

STOCHASTIC MODEL PREDICTIVE CONTROL

OPTIMIZE DECISIONS UNDER UNCERTAINTY

- In many control problems decisions must be taken under **uncertainty**



- **Robust** control approaches do not model uncertainty (only assume that is bounded) and pessimistically consider the worst case

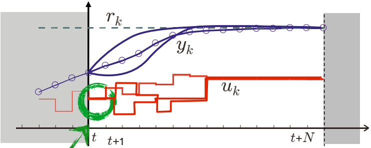
- **Stochastic** models provide instead additional information about uncertainty

- **Optimality** is often sought (ex: minimize expected economic cost)

STOCHASTIC MODEL PREDICTIVE CONTROL

- At time t: solve a **stochastic optimal control** problem over a finite future horizon of N steps:

$$
\begin{array}{ll}
\min_u & E_w\left[\sum_{k=0}^{N-1} \ell(y_k, u_k, w_k)\right] \\
\text{s.t.} & x_{k+1} = A(w_k)x_k + B(w_k)u_k + f(w_k) \\
& y_k = C(w_k)x_k + D(w_k)u_k + g(w_k) \\
& u_{\min} \le u_k \le u_{\max} \\
& y_{\min} \le y_k \le y_{\max}, \forall w \quad \text{robustness} \\
& x_0 = x(t) \quad \text{feedback}
\end{array}
$$



$x(t)$ = process state

$u(t)$ = manipulated vars

$y(t)$ = controlled output

$w(t)$ = **stochastic disturbances**

- Solve stochastic optimal control problem w.r.t. future input sequence
- Apply the first optimal move $u(t)=u_0^*$, throw the rest of the sequence away
- At time $t+1$: Get new measurements, repeat the optimization. And so on …

LINEAR STOCHASTIC MODEL W/ DISCRETE DISTURBANCE

- **Linear stochastic** prediction model

$$\begin{cases} x_{k+1} = A(w_k)x_k + B(w_k)u_k + f(w_k) \\ y_k = C(w_k)x_k + g(w_k) \end{cases}$$

possibly subject to stochastic output constraints $y_{\min}(w_k) \le y_k \le y_{\max}(w_k)$

- **Stochastic discrete disturbance**

$$w_k \in \{w^1, \dots, w^s\}$$

with discrete probabilities $p_j = \Pr\left[w_k = w^j\right], p_j \ge 0, \sum_{j=1}^{s} p_j = 1$

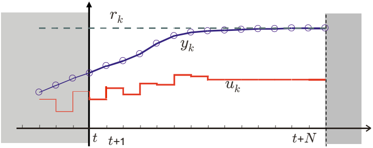
- (A, B, C) can be sparse matrices (e.g., network of interacting subsystems)

- Often w_k is low-dimensional (e.g., driver's power request, obstacle velocities, electricity price, weather, ...)

COST FUNCTIONS FOR SMPC TO MINIMIZE

- **Expected performance**

$$\min_u \sum_{k=0}^{N-1} E_w\left[(y_k - r_k)^2\right]$$



- Tradeoff between **expectation & risk**

$$\min_u \sum_{k=0}^{N-1} (E_w\left[y_k - r_k\right])^2 + \alpha \mathrm{Var}_w\left[y_k - r_k\right] \qquad \alpha \geq 0$$

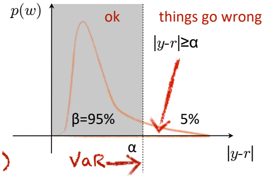
- Note that they coincide for α=1, since

$$\mathrm{Var}_w\left[y_k - r_k\right] = E_w\left[(y_k - r_k)^2\right] - (E_w\left[y_k - r_k\right])^2$$

- Conditional Value-at-Risk (**CVaR**) (Rockafellar, Uryasev, 2000)

$$\min_{u,\alpha} \sum_{k=0}^{N-1} \alpha_k + \frac{1}{1-\beta} E_w[\max\{|y_k - r_k| - \alpha_k, 0\}]$$

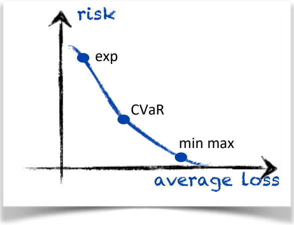
= minimize expected loss when things go wrong (convex !)



= expected shortfall

- **Min-max** = minimize worst case performance

$$\min_{u} \sum_{k=0}^{N-1} \max_{w} |y_k - r_k|$$



- CVaR optimization (Rockafellar, Uryasev, 2000)

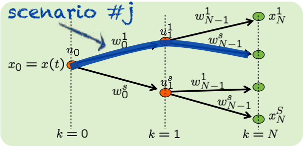
$$\min_{u,\alpha} \sum_{k=0}^{N-1} \alpha_k + \frac{1}{1-\beta} E_w\left[\max\left\{|y_k - r_k| - \alpha_k, 0\right\}\right]$$

$$\begin{array}{ll}
\min_{u,z,\alpha} & \displaystyle\sum_{k=0}^{N-1} \alpha_k + \frac{1}{1-\beta} \sum_{j=1}^{S} p^j z_k^j \\
\text{s.t.} & z_k^j \geq y_k^j - r_k^j - \alpha_k \\
& z_k^j \geq r_k^j - y_k^j - \alpha_k \\
& z_k^j \geq 0
\end{array}$$

CVaR optimization becomes a linear programming problem

STOCHASTIC OPTIMAL CONTROL PROBLEM

- Enumerate all possible scenarios $\{w_0^j, w_1^j, \dots, w_{N-1}^j\}, j = 1, \dots, S$
- Scenario = path on the tree
- Number S of scenarios = number of leaf nodes



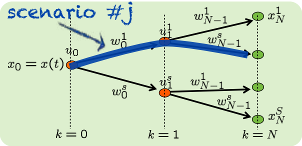
- Each scenario has probability $p_j = \displaystyle\prod_{k=0}^{N-1} \mathbf{Pr}[w_k = w_k^j]$

STOCHASTIC OPTIMAL CONTROL PROBLEM

- Each scenario has its own evolution

$$x_{k+1}^j = A(w_k^j)x_k^j + B(w_k^j)u_k^j + f(w_k^j)$$

(=linear time-varying system)



- Expectations become simple sums!

Example: $\min E_w \left[x'_N P x_N + \sum_{k=0}^{N-1} x'_k Q x_k + u'_k R u_k \right]$

$$\min \sum_{j=1}^{S} p^j \left((x_N^j)' P x_N^j + \sum_{k=0}^{N-1} (x_k^j)' Q x_k^j + (u_k^j)' R u_k^j \right)$$

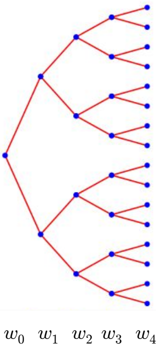
Expectations of quadratic costs remain quadratic costs

SCENARIO TREE GENERATION FROM DATA

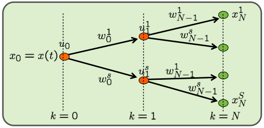
- Scenario trees can be generated by **clustering** sample paths
- Paths can be obtained by **Monte Carlo simulation** of (estimated) models, or from **historical data**
- The **number of nodes** can be decided a priori

Heuristic
Multilevel
Clustering

(Heitsch, Römisch, 2009)



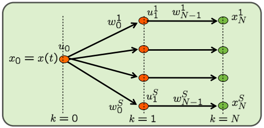
- **Alternatives** (simpler but less accurate): use histograms (only for $w_k \in \mathbb{R}$) or **K-means** (also in higher dimensions), within a recursive algorithm



Stochastic control (scenario tree)

Causality constraints: $u_k^j = u_k^h$ when scenarios j and h share the same node at prediction time k (in particular, $u_0^j \equiv u_0$ at root node $k=0$)

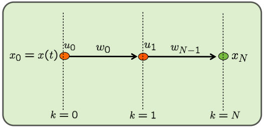
Decision u_k only depends on past disturbance realizations $w_0, \dots, w_{k-1}$



Stochastic control (scenario fan)

No causality in prediction: only $u_0^j \equiv u_0$ at root node.

Decision u_k depends on future disturbance realizations.

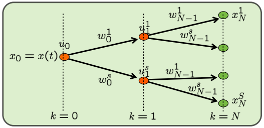


Deterministic control (single disturbance sequence)

- frozen-time: $w_k \equiv w(t), \forall k$ (causal prediction)
- prescient: $w_k = w(t+k)$ (non-causal)
- certainty equivalence: $w_k = E[w(t+k|t)]$ (causal)

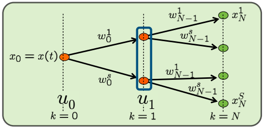
Tradeoff between **complexity** (=number of nodes) and **performance** (=accuracy of stochastic modeling)

OPEN-LOOP VS CLOSED-LOOP PREDICTION



closed-loop prediction

A different move u_k is optimized to counteract each outcome of the disturbance w_k



open-loop prediction

Only a sequence of inputs $u_0, \ldots, u_{N-1}$ is optimized, the same u_k must be good for all possible disturbances w_k

- Intuitively: OL prediction is more conservative than CL in handling constraints
- OL problem = CL problem + additional constraints (=less degrees of freedom)

LINEAR STOCHASTIC MPC FORMULATION

- A rich literature on stochastic MPC is available

  (Schwarme, Nikolaou, 1999) (Munoz de la Pena, Bemporad, Alamo, 2005) (Primbs, 2007) (Oldewurtel, Jones, Morari, 2008) (Wendt, Wozny, 2000) (Couchman, Cannon, Kouvaritakis, 2006) (Ono, Williams, 2008) (Batina, Stoorvogel, Weiland, 2002) (van Hessem, Bosgra 2002) (Bemporad, Di Cairano, 2005) (Bernardini, Bemporad, 2012)

  See also the survey paper (Mesbah, 2016)

- Performance index: $\min E_w \left[x'_N P x_N + \sum_{k=0}^{N-1} x'_k Q x_k + u'_k R u_k \right]$

- **Goal**: ensure **mean-square convergence** $\lim_{t\to\infty} E[x'(t)x(t)] = 0 \quad (f(w(t)) = 0)$

- Mean-square stability ensured by **stochastic Lyapunov function** $V(x) = x'Px$

$$E_{w(t)} \left[V(x(t+1)) \right] - V(x(t)) \leq -x'(t)Lx(t), \ \forall t \geq 0$$

$$P = P' \succ 0$$
$$L = L' \succ 0$$

(Morozan, 1983)

STABILIZING STOCHASTIC MPC

(Bernardini, Bemporad, 2012)

- Impose stochastic stability constraint in SMPC problem (=quadratic constraint w.r.t. u_0)

$$\begin{array}{ll}\min\limits_{u} & E_w\left[\sum_{k=0}^{N-1}\ell(x_k,u_k)\right] \\ \text{s.t.} & x_{k+1}=A(w_k)x_k+B(w_k)u_k \\ & E\left[V(A(w_0)x_0+B(w_0)u_0)\right]\leq x_0'(Q^{-1}-L)x_0 \\ & x_0=x(t)\end{array}$$

performance and stability are decoupled

- SMPC approach:
  1. Solve LMI problem off-line to find stochastic Lyapunov fcn $V(x)=x'Q^{-1}x$
  2. Optimize stochastic performance based on scenario tree

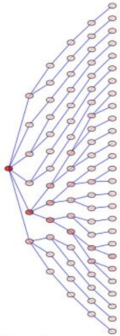
Theorem: The closed-loop system is as. stable in the mean-square sense

- SMPC can be generalized to handle **input and state constraints**

Note: *recursive feasibility* guaranteed by backup solution $u(k)=Kx(k)$

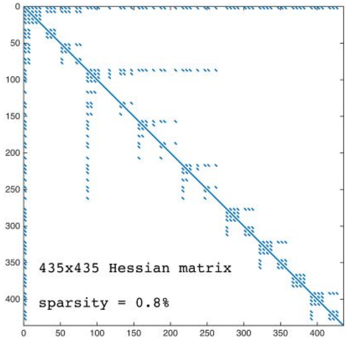
COMPLEXITY OF STOCHASTIC OPTIMIZATION PROBLEM

- #optimization variables = #nodes x #inputs (in condensed version)
- Problems are **very sparse** (well exploited by **interior point methods**)
- Example: SMPC with quadratic cost and linear constraints

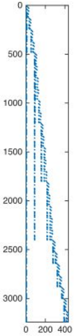


Tree=87 nodes

Branching factor M=[6 3 2 2 2]



435 free variables (5 inputs x node)



3240x435 constraint matrix
sparsity = 1.1%

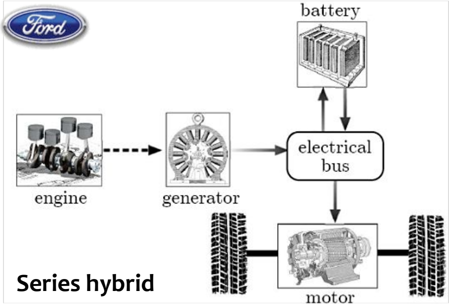
Control problem:

(Bichi, Ripaccioli, Di Cairano, Bernardini, Bemporad, Kolmanovsky, CDC 2010)

Decide optimal generation of **mechanical power** (from engine) and **electrical power** (from battery) to satisfy **driver's power request**

What will the future power request from the driver be ?

$P_{req}(w(t))$ = driver's power request



$$P_{req}(k) = P_{el}(k) + P_{mec}(k) - P_{br}(k)$$

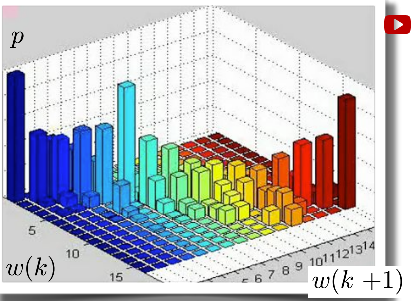
LEARNING A STOCHASTIC MODEL OF THE DRIVER

- The driver action on the vehicle is modeled by the **stochastic** process $w(k)$
- Assume that the realization $w(k)$ can be **measured** at every time step k
- Depending on the **application**, $w(k)$ may represent different quantities (e.g., power request in an HEV, acceleration, velocity, steering wheel angle, ...)

Good model for control purposes: $w(k)$ = **Markov chain**

$$[T]_{ij} = \mathbf{P}[w(k+1) = w_j | w(k) = w_i]$$

Number of states in Markov chain determines the **trade-off** between complexity *and* accuracy



Transition probability matrix T is easily estimated from driver's data

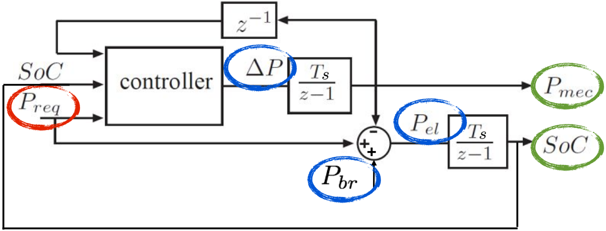
Several model improvements are possible (e.g., multiple Markov chains)

Manipulates inputs

$$\Delta P(k),\ P_{el}(k),\ P_{br}(k)$$

Uncertainty

$$P_{req}(w(k))$$



sample time T_s=1 s

Controlled output

$$P_{req}(k) = P_{el}(k) + P_{mec}(k) - P_{br}(k)$$

Constraints

$$\begin{array}{rclcl}
SoC_{min} & \leq & SoC(k) & \leq & SoC_{max} \\
0 & \leq & P_{mec}(k-1) & \leq & P_{mec,max} \\
P_{el,min} & \leq & P_{el}(k) & \leq & P_{el,max} \\
\Delta P_{min} & \leq & \Delta P & \leq & \Delta P_{max} \\
0 & \leq & P_{br}(k) & &
\end{array}$$

State-space equations

$$SoC(k+1) = SoC(k) - KT_sP_{el}(k)$$

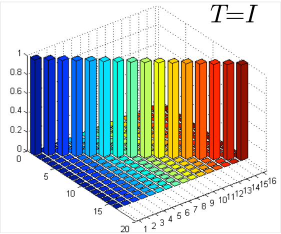
$$P_{mec}(k+1) = P_{mec}(k) + \Delta P(k)$$

COMPARISON WITH DETERMINISTIC MPC

"Frozen-time" MPC (FTMPC)

No stochastic disturbance model,
simply ZOH along prediction horizon

$$P_{req}(w(t+k|k)) = P_{req}(w(k))$$



"Prescient" MPC (PMPC)

Future disturbance sequence $P_{req}(w(t+k|k))$
known in advance

SIMULATION RESULTS: CONTROLLER COMPARISON

Comparison on different driving cycles

SHEV ENERGY MANAGEMENT SIMULATION RESULTS ON STANDARD DRIVING CYCLES

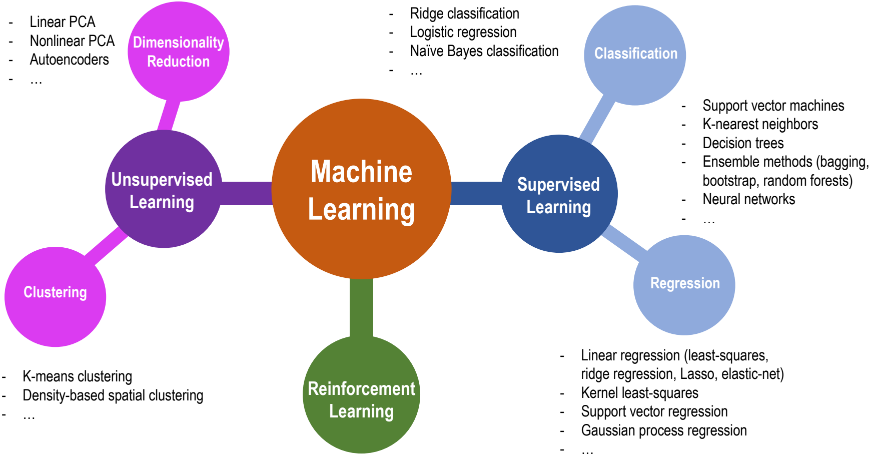
	$\|\Delta P\|$	Fuel cons.	ΔSoC gain/loss	Equiv. fuel cons.	impr. wrt FTMPC
			NEDC		
FTMPC	37.57kW	204g	0.35%	197g	–
→ SMPCL	16.28kW	166g	-0.82%	184g	6.45%
PMPC	15.25kW	196g	0.84%	177g	9.97%
			FTP-75		
FTMPC	89.28kW	348g	0.64%	334g	–
→ SMPCL	26.07kW	292g	0.08%	290g	13.10%
PMPC	32.30kW	307g	0.89%	286g	14.20%
			FTP-Highway		
FTMPC	39.33kW	267g	0.64%	253g	–
→ SMPCL	16.84kW	281g	2.12%	235g	7.26%
PMPC	16.33kW	254g	0.91%	234g	7.32%

pretty close to having the crystal ball.
But we don't, we just model uncertainty carefully

LEARNING-BASED NONLINEAR MPC

MACHINE LEARNING (ML)

- Massive set of techniques to **extract mathematical models from data**



MACHINE LEARNING (ML)

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

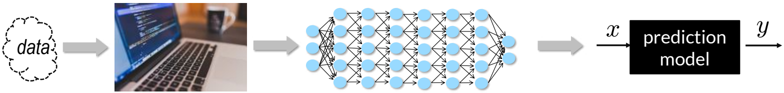
- **Works very well** in practice (despite training is most often a nonconvex optimization problem ...)

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

- Availability of excellent open-source **software tools** also explains success
  `scikit-learn`, **`TensorFlow/Keras`**, **`PyTorch`**, **`JAX`**, **`Flux.jl`**, ... python julia

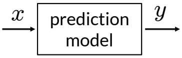
LEARNING PREDICTION MODELS FOR MPC

- **Black-box modeling**: purely data-driven. Use training data to fit a prediction model that can explain them



- **Physics-based modeling**: use physical principles to create a prediction model (e.g.: weather forecast, chemical reaction, mechanical laws, ...)

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



- **Gray-box modeling** is a mix of the two. It can be quite effective

"All models are wrong, but some are useful."

(George E. P. Box)

MODELS FOR CONTROL SYSTEMS DESIGN

- Prediction models for **model predictive control**:
  - Complex model = complex controller
    $\rightarrow$ model must be as **simple** as possible
  - Easy to **linearize** (to get Jacobian matrices for nonlinear optimization)

- Prediction models for **state estimation**:
  - Complex model = complex Kalman filter
  - Easy to linearize

- Models for **virtual sensing**:
  - No need to use simple models (except for computational reasons)

- Models for **diagnostics**:
  - Usually a **classification problem** to solve
  - Complexity is also less of an issue

Linear models

- linear I/O models (ARX, ARMAX,...)
- subspace linear SYS-ID
- linear regression (ridge, elastic-net, Lasso)

Piecewise linear models

- decision-trees
- neural nets + (leaky)ReLU
- K-means + linear models

Nonlinear linear models

- basis functions + linear regression
- neural networks
- ~~K-nearest neighbors~~
- ~~support vector machines~~
- ~~kernel methods~~
- ~~random forests~~

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 $y_t \approx \mathcal{N}(y_{t-1}, \ldots, y_{t-n_a}, u_{t-1}, \ldots, u_{t-n_b})$

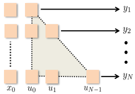
- **Neural state-space** models:

  - **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)

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

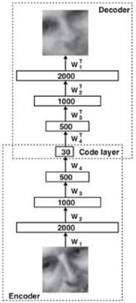
$$y_{t+k} = h_k(x_t, u_t, \ldots, u_{t+k-1}), \; k = 1, \ldots, N$$



- **Recurrent neural networks** are more appropriate for accurate open-loop predictions, but more difficult to train (see later ...)

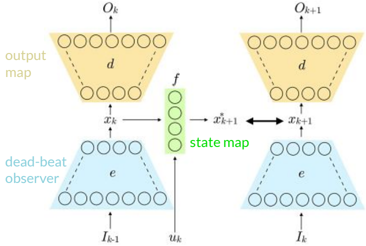
NONLINEAR STATE-SPACE MODELS VIA AUTOENCODERS

- **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)



$$
\begin{aligned}
O_k &= [y_k' \ \ldots \ y_{k-m}']' \\
I_k &= [y_k' \ \ldots \ y_{k-n_a+1}' \ u_k' \ \ldots \ u_{k-n_b+1}']'
\end{aligned}
$$

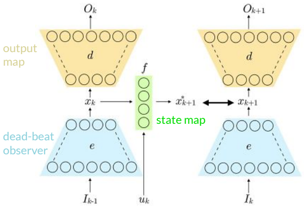
(Masti, Bemporad, 2021)

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

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

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

$$
\begin{aligned}
\min_{f,d,e} \quad & \sum_{k=k_0}^{N-1} \alpha \left(\ell_1(\hat{O}_k, O_k) + \ell_1(\hat{O}_{k+1}, O_{k+1})\right) \\
& +\beta\ell_2(x^\star_{k+1}, x_{k+1}) + \gamma\ell_3(O_{k+1}, O^\star_{k+1}) \\
\text{s.t.} \quad & x_k = e(I_{k-1}),\ k = k_0, \ldots, N \\
& x^\star_{k+1} = f(x_k, u_k),\ k = k_0, \ldots, N-1 \\
& \hat{O}_k = d(x_k),\ O^\star_k = d(x^\star_k),\ k = k_0, \ldots, N
\end{aligned}
$$



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

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

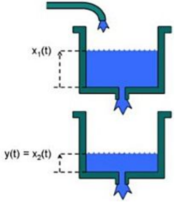
$$
\begin{aligned}
f(x_k, u_k) &= A(x_k, u_k) \begin{bmatrix} x_k \\ 1 \end{bmatrix} + B(x_k, u_k) u_k \\
y_k &= C(x_k, u_k) \begin{bmatrix} x_k \\ 1 \end{bmatrix}
\end{aligned}
$$

(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

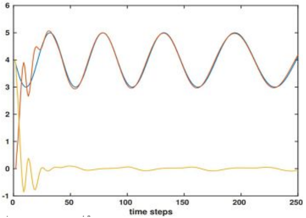


www.mathworks.com

$$\begin{cases} x_1(t+1) = x_1(t) - k_1\sqrt{x_1(t)} + k_2u(t) \\ x_2(t+1) = x_2(t) + k_3\sqrt{x_1(t)} - k_4\sqrt{x_2(t)} \\ y(t) = x_2(t) + u(t) \end{cases}$$

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

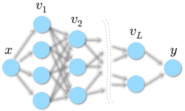


LTV-MPC results

TRAINING FEEDFORWARD NEURAL NETWORKS

- **Feedforward neural network** model:

$$y_k = f_y(x_k,\theta) = \begin{cases} v_{1k} &= A_1 x_k + b_1 \\ v_{2k} &= A_2 f_1(v_{1k}) + b_2 \\ \vdots & \quad \vdots \\ v_{Lk} &= A_{L_y} f_{L-1}(v_{(L-1)k}) + b_L \\ \hat{y}_k &= f_L(v_{Lk}) \end{cases}$$

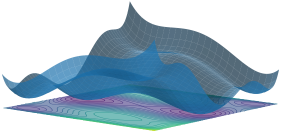


$$\theta = (A_1, b_1, \ldots, A_L, b_L)$$

Examples: x_k = measured state, or $x_k = (y_{k-1},\ldots,y_{k-n_a},u_{k-1},\ldots,u_{k-n_b})$

- **Training problem**: given a dataset $\{x_0,y_0,\ldots,x_{N-1},y_{N-1}\}$ solve

$$\min_{\theta} r(\theta) + \sum_{k=0}^{N-1} \ell(y_k, f(x_k,\theta))$$



- It is a nonconvex, unconstrained, nonlinear programming problem that can be solved by **stochastic gradient descent**, **quasi-Newton** methods, ... and **EKF** !

TRAINING FEEDFORWARD NEURAL NETWORKS BY EKF

(Singhal, Wu, 1989) (Puskorius, Feldkamp, 1994)

- **Key idea**: treat parameter vector θ of the feedforward neural network as a **constant state**

$$\begin{cases} \theta_{k+1} &= \theta_k + \eta_k \\ y_k &= f(x_k, \theta_k) + \zeta_k \end{cases}$$

  and use EKF to estimate θ_k **on line** from a streaming dataset $\{x_k, y_k\}$

- Ratio $\mathrm{Var}[\eta_k]/\,\mathrm{Var}[\zeta_k]$ is related to the **learning-rate**

- Initial matrix $(P_{0|-1})^{-1}$ is related to **quadratic regularization** on θ

- Implemented in **ODYS Deep Learning** library

- Extended to rather **arbitrary convex loss functions/regularization** terms

  (Bemporad, 2021 - `https://arxiv.org/abs/2111.02673`)

RECURRENT NEURAL NETWORKS

- **Recurrent Neural Network** (RNN) model:

$$\begin{aligned} x_{k+1} &= f_x(x_k, u_k, \theta_x) \\ y_k &= f_y(x_k, \theta_y) \end{aligned}$$

f_x, f_y = feedforward neural network

- **Training problem**: given a dataset $\{u_0, y_0, \ldots, u_{N-1}, y_{N-1}\}$ solve

$$\begin{aligned} \min_{\substack{\theta_x, \theta_y \\ x_0, x_1, \ldots, x_{N-1}}} \quad & r(x_0, \theta_x, \theta_y) + \sum_{k=0}^{N-1} \ell(y_k, f_y(x_k, \theta_y)) \\ \text{s.t.} \quad & x_{k+1} = f_x(x_k, u_k, \theta_x) \end{aligned}$$

- **Main issue**: x_k are **hidden states**, i.e., are **unknowns** of the problem

TRAINING RNNS ONLINE BY EKF

(Bemporad, 2021 - `https://arxiv.org/abs/2111.02673`)

- Estimate both hidden states x_k and parameters θ_x, θ_y by EKF based on

$$\left\{\begin{array}{rcl} x_{k+1} & = & f_x(x_k, u_k, \theta_{xk}) + \xi_k \\ \begin{bmatrix}\theta_{x(k+1)} \\ \theta_{y(k+1)}\end{bmatrix} & = & \begin{bmatrix}\theta_{xk} \\ \theta_{yk}\end{bmatrix} + \eta_k \\ y_k & = & f_y(x_k, \theta_{yk}) + \zeta_k \end{array}\right.$$

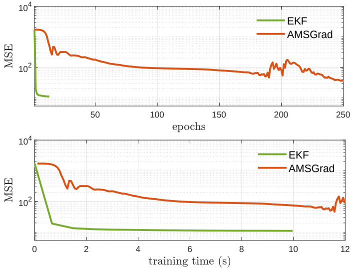
- RNN and its hidden state x_k can be estimated **on line** from a streaming dataset $\{u_k, y_k\}$, and/or **offline** by processing multiple epochs of a given dataset

- Can handle **general smooth strictly convex** loss functions/regularization terms

- Can add ℓ_1-penalty $\lambda \left\| \begin{bmatrix}\theta_x \\ \theta_y\end{bmatrix} \right\|_1$ to **sparsify** θ_x, θ_y by changing EKF update into

$$\begin{bmatrix}\hat{x}(k|k) \\ \theta_x(k|k) \\ \theta_y(k|k)\end{bmatrix} = \begin{bmatrix}\hat{x}(k|k-1) \\ \theta_x(k|k-1) \\ \theta_y(k|k-1)\end{bmatrix} + M(k)e(k) - \lambda P(k|k-1) \begin{bmatrix} 0 \\ \mathrm{sign}(\theta_x(k|k-1)) \\ \mathrm{sign}(\theta_y(k|k-1))\end{bmatrix}$$

- Dataset: 3499 I/O data of **magneto-rheological fluid damper** (Wang et al., 2009)
- N=2000 data used for training, 1499 for testing the model
- Same data used in NNARX modeling demo of SYS-ID Toolbox for MATLAB
- **RNN model**: **4** hidden states
  shallow state-update and output functions
  6 neurons each, **leaky-ReLU** activation
- Compare with gradient descent (AMSGrad)

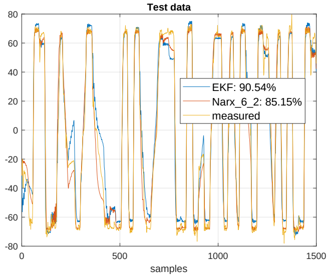


- Training time measured on MATLAB+CasADi implementation of EKF/AMSGrad

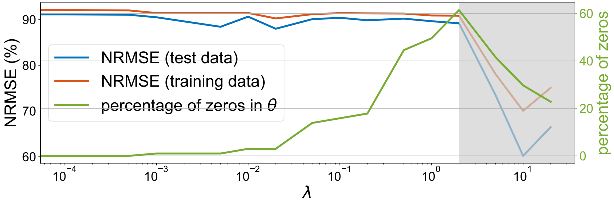
TRAINING RNNS BY EKF - EXAMPLES

- Compare NRMSE[1] wrt NNARX model (SYS-ID TBX):

EKF = **91.97**, AMSGrad = **85.58**, NNARX(6,2) = **88.18** (**training**)
EKF = **90.54**, AMSGrad = **80.95**, NNARX(6,2) = **85.15** (**test**)



- Repeat training with ℓ_1-penalty $\lambda \left\| \begin{bmatrix} \theta_x \\ \theta_y \end{bmatrix} \right\|_1$



[1] normalized root-mean-square error

TRAINING RNNS BY EKF - EXAMPLES

- Dataset: 2000 I/O data of linear system with **binary outputs**

$$x(k+1) = \begin{bmatrix} .8 & .2 & -.1 \\ 0 & .9 & .1 \\ .1 & -.1 & .7 \end{bmatrix} x(k) + \begin{bmatrix} -1 \\ .5 \\ 1 \end{bmatrix} u(k) + \xi(k) \qquad \text{Var}[\xi_i(k)] = \sigma^2$$

$$y(k) = \begin{cases} 1 & \text{if } [\,-2\ 1.5\ 0.5\,]\,x(k) - 2 + \zeta(k) \geq 0 \\ 0 & \text{otherwise} \end{cases} \qquad \text{Var}[\zeta(k)] = \sigma^2$$

- N=1000 data used for training, 1000 for testing the model

- Train **linear state-space model** with 3 states and **sigmoidal output** function

$$f_1^y(y) = 1/(1 + e^{-A_1^y [x'(k)\ u(k)]' - b_1^y})$$

σ	accuracy [%] training	test
0.000	99.20	98.90
0.001	99.30	98.90
0.010	99.20	98.70
0.100	96.50	97.00
0.200	93.00	93.80

- Training loss: (modified) **cross-entropy** loss

$$\ell_{\text{CE}\epsilon}(y(k), \hat{y}) = \sum_{i=1}^{n_y} -y_i(k) \log(\epsilon + \hat{y}_i) - (1 - y_i(k)) \log(1 + \epsilon - \hat{y}_i)$$

TRAINING RNNS BY SEQUENTIAL LEAST-SQUARES

(Bemporad, 2021 - `http://arxiv.org/abs/2112.15348`)

- RNN training problem = **optimal control** problem:

$$\begin{array}{rl} \min_{\theta_x,\theta_y,x_0,x_1,\ldots,x_{N-1}} & r(x_0,\theta_x,\theta_y)+\displaystyle\sum_{k=0}^{N-1}\ell(y_k,\hat{y}_k) \\ \text{s.t.} & x_{k+1}=f_x(x_k,u_k,\theta_x) \\ & \hat{y}_k=f_y(x_k,\theta_y) \end{array}$$

  - θ_x,θ_y,x_0 = manipulated variables, $\hat{y}_k$ = output, y_k = reference signal
  - $r(x_0,\theta_x,\theta_y)$ = input penalty, $\ell(y_k,\hat{y}_k)$ = output penalty
  - N = prediction horizon, control horizon = 1

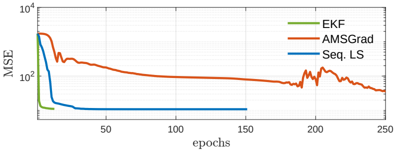
- Linearized model:

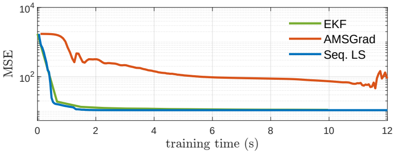
$$\begin{array}{rcl} \Delta x_{k+1} & = & (\nabla_x f_x)'\Delta x_k+(\nabla_{\theta_x} f_x)'\Delta\theta_x \\ \Delta y_k & = & (\nabla_{x_k} f_y)'\Delta x_k+(\nabla_{\theta_y} f_y)'\Delta\theta_y \end{array}$$

- **Idea**: take $2^{\rm nd}$-order expansions of the loss ℓ and regularization term r and use **sequential least-squares** + line search to minimize wrt x_0,θ_x,θ_y

(Bemporad, 2021 - `http://arxiv.org/abs/2112.15348`)

- Fluid-damper example:





- We want to also handle **non-smooth** (and **non-convex**) regularization terms

$$\begin{array}{rl}\min_{\theta_x,\theta_y,x_0} & r(x_0,\theta_x,\theta_y)+\sum_{k=0}^{N-1}\ell(y_k,f_y(x_k,\theta_y))+g(\theta_x,\theta_y)\\ \text{s.t.} & x_{k+1}=f_x(x_k,u_k,\theta_x)\end{array}$$

- **Idea**: use **alternating direction method of multipliers** (ADMM) by splitting

$$\begin{array}{rl}\min_{\theta_x,\theta_y,x_0,\nu_x,\nu_y} & r(x_0,\theta_x,\theta_y)+\sum_{k=0}^{N-1}\ell(y_k,f_y(x_k,\theta_y))+g(\nu_x,\nu_y)\\ \text{s.t.} & x_{k+1}=f_x(x_k,u_k,\theta_x)\\ & \left[\begin{smallmatrix}\nu_x\\ \nu_y\end{smallmatrix}\right]=\begin{bmatrix}\theta_x\\ \theta_y\end{bmatrix}\end{array}$$

(Bemporad, 2021 - `http://arxiv.org/abs/2112.15348`)

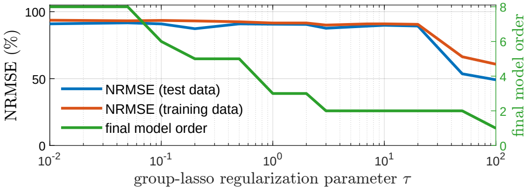
- ADMM + Seq. LS = **NAILS** algorithm (Nonconvex ADMM Iterations and Sequential LS)

$$\begin{bmatrix} x_0^{t+1} \\ \theta_x^{t+1} \\ \theta_y^{t+1} \end{bmatrix} = \arg\min_{x_0,\theta_x,\theta_y} V(x_0,\theta_x,\theta_y) + \frac{\rho}{2}\left\| \begin{bmatrix} \theta_x - \nu_x^t + w_x^t \\ \theta_y - \nu_y^t + w_y^t \end{bmatrix} \right\|_2^2 \quad \text{(sequential) LS}$$

$$\begin{bmatrix} \nu_x^{t+1} \\ \nu_y^{t+1} \end{bmatrix} = \text{prox}_{\frac{1}{\rho}g}(\theta_x^{t+1} + w_x^t, \theta_y^{t+1} + w_y^t) \quad \text{proximal step}$$

$$\begin{bmatrix} w_x^{t+1} \\ w_y^{t+1} \end{bmatrix} = \begin{bmatrix} w_x^h + \theta_x^{t+1} - \nu_x^{t+1} \\ w_y^h + \theta_y^{t+1} - \nu_y^{t+1} \end{bmatrix} \quad \text{update dual vars}$$

- Fluid-damper example: **group-Lasso regularization** $g(\nu_i^g) = \tau \sum_{i=1}^{n_x} \|\nu_i^g\|_2$ to zero entire rows and columns and **reduce state-dimension** automatically



TRAINING RNNS BY SEQUENTIAL LS AND ADMM

(Bemporad, 2021 - `http://arxiv.org/abs/2112.15348`)

- Fluid-damper example: **quantization** of θ_x, θ_y for simplifying model arithmetic +ReLU activation function

$$g(\theta_i) = \begin{cases} 0 & \text{if } \theta_i \in \mathcal{Q} \\ +\infty & \text{otherwise} \end{cases}$$

$\mathcal{Q}$ = multiples of 0.1 between -0.5 and 0.5

  - NRMSE = **83.10** (training), **80.51** (test)
  - NRMSE = **8.83** (training), **2.69** (test) $\leftarrow$ **no ADMM, just quantize after training**
  - Training time: ≈ 5 s

- **Note**: no convergence to a global minimum is guaranteed

- **NAILS** = very flexible & efficient learning algorithm for **control-oriented RNNs**

TRAINING RNNS

- Computation time (Intel Core i9-10885H CPU @2.40GHz):

language	autodiff	EKF /time step CPU time	seq. LS /epoch CPU time
Python 3.8.1	PyTorch	≈ 30 ms	(N/A)
Python 3.8.1	JAX	≈ 9 ms	≈ 1.0 s
Julia 1.7.1	Flux.jl	≈ 2 ms	≈ 0.8 s

- Several **sparsity patterns** can be exploited in EKF updates (supported by **ODYS EKF** and **ODYS Deep Learning** libraries)

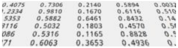
- **Note**: Extension to **gray-box** identification + state-estimation is immediate

- **Note**: RNN training by EKF can be used to generalize **output disturbance models** for offset-free set-point tracking to nonlinear I/O disturbance models

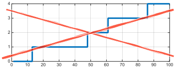
NONLINEAR MPC BASED ON NEURAL NETWORKS

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

- **Neural nonlinear MPC** requires **advanced technical software** to run efficiently and reliably (**model learning**, **problem construction**, **optimization**)

DEEP NONLINEAR MPC FOR AUTONOMOUS DRIVING

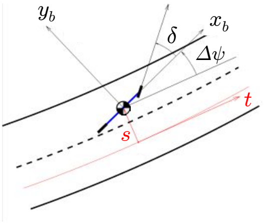
- **Goal**: track desired longitudinal speed (v_y), lateral displacement (e_y) and orientation ($\Delta\Psi$)

- **Inputs**: wheel torque T_w and steering angle δ

- **Constraints**: on e_y and lateral displacement s (for obstacle avoidance) and manipulated inputs T_w, δ

- **Sampling time**: 100 ms

- **Model**: gray-box bicycle model



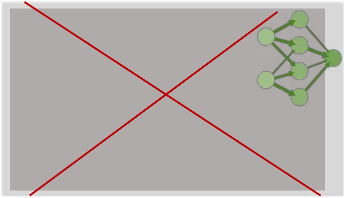
$$\dot{s} = \frac{v_x \cos\Delta\psi - v_y \sin\Delta\psi}{1 - \kappa e_y}$$
$$\dot{e}_y = v_x \sin\Delta\psi + v_y \cos\Delta\psi$$
$$\Delta\dot{\psi} = \omega - \kappa\dot{s}$$

- **kinematics** is simple to model (white box)

- **tire forces** harder to model + **stiff** wheel slip ratio dynamics (k_f, k_r) $\Rightarrow$ small integration step required

- learn a **black-box neural-network model** !

(Boni, Capelli, Frascati @ODYS, 2021)



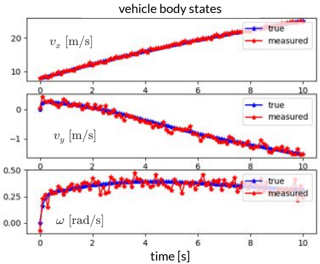
DEEP NONLINEAR MPC FOR AUTONOMOUS DRIVING

- **ODYS Deep Learning Toolset** used to learn a neural-network with input $(v_x, v_y, \omega, k_f, k_r, T_w, \delta)$ @k and output $(v_x, v_y, \omega, k_f, k_r)$ @$k+1$

- Data generated from high-fidelity simulation model with noisy measurements, sampled @10Hz

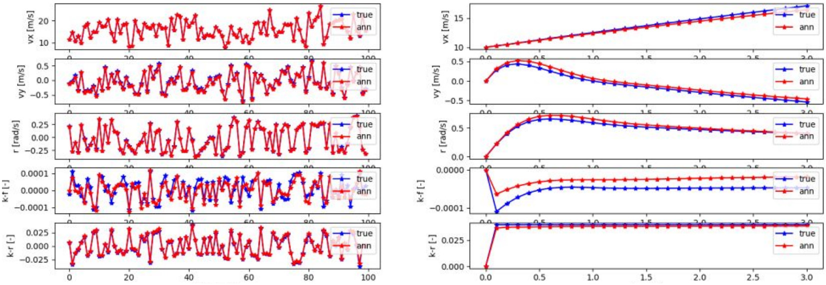
- Neural network model: **2 hidden layers, 55 neurons each**

- Advantages of black-box (neural network) model:
  - No physical model required describing tire-road interaction
  - directly learn the model in discrete-time ($T_s = 100$ ms)



- Model validation on test data:

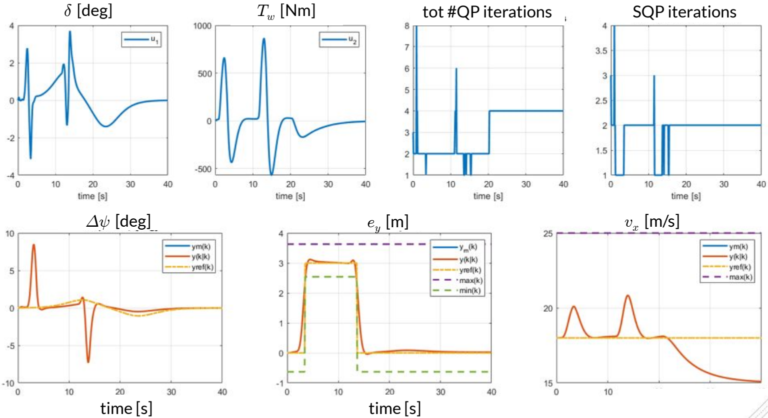
one-step ahead prediction on test data | open-loop predictions



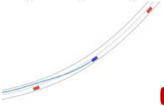
sample | time [s]

- C-code (network+Jacobians) automatically generated for ODYS MPC

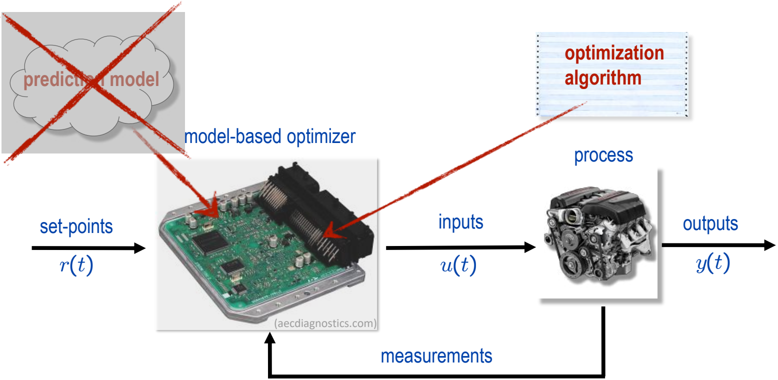
- **Closed-loop MPC**: overtake vehicle #1, keep safety distance from vehicle #2



- Good reference tracking, constraints on e_y, v_x satisfied, smooth command action

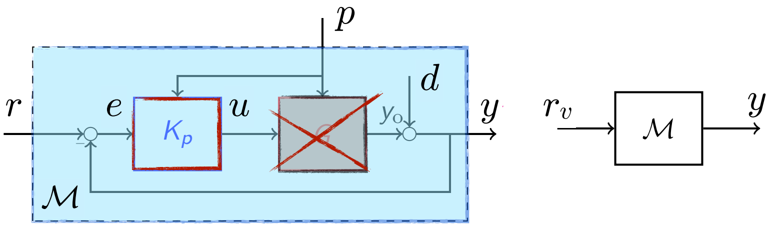


DIRECT DATA-DRIVEN MPC



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

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

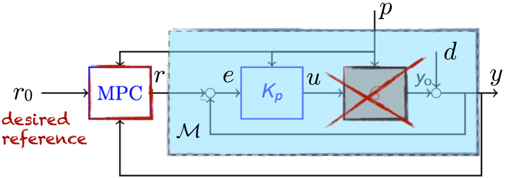


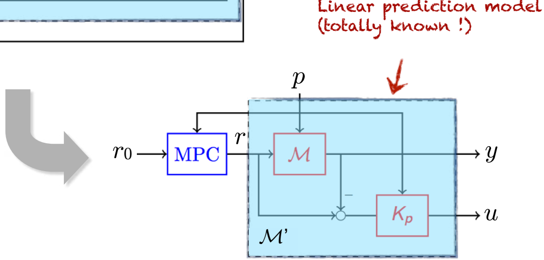
- Collect a set of **data** $\{u(t), y(t), p(t)\}, t = 1, \ldots, N$
- Specify a **desired closed-loop linear model** $\mathcal{M}$ from r to y
- Compute $r_v(t) = \mathcal{M}^{\#} y(t)$ from **pseudo-inverse model** $\mathcal{M}^{\#}$ of $\mathcal{M}$
- **Identify** linear (LPV) model K_p from $e_v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

DIRECT DATA-DRIVEN MPC

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

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





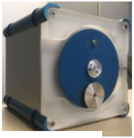
- MPC designed to handle input/output **constraints** and improve **performance**

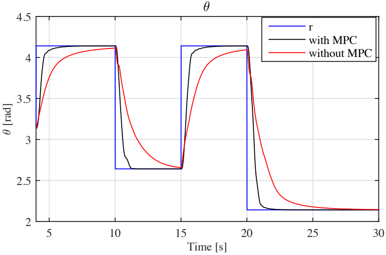
  (Piga, Formentin, Bemporad, 2017)

DIRECT DATA-DRIVEN MPC - AN EXAMPLE

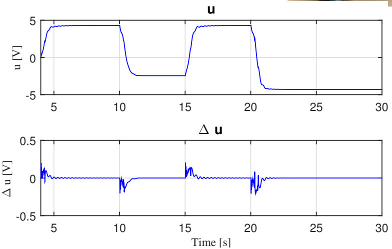
- Experimental results: MPC handles soft constraints on u, Δu and y

  (motor equipment by courtesy of TU Delft)





desired tracking performance achieved

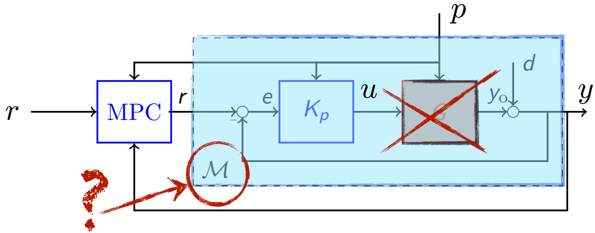


constraints on input increments satisfied

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

OPTIMAL DIRECT DATA-DRIVEN MPC

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



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

OPTIMAL DIRECT DATA-DRIVEN MPC

(Selvi, Piga, Bemporad, 2018)

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

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

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

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

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

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

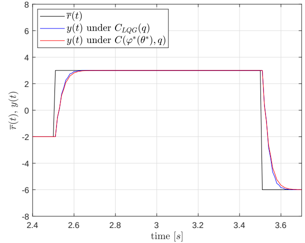
OPTIMAL DIRECT DATA-DRIVEN MPC

(Selvi, Piga, Bemporad, 2018)

- **Results**: linear process

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

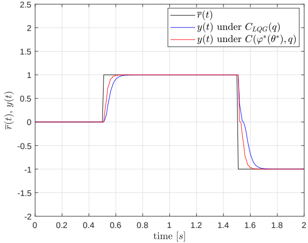
Data-driven controller **only 1.3% worse** than model-based LQR (=SYS-ID on same data + LQR design)



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

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

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



LEARNING OPTIMAL MPC CALIBRATION

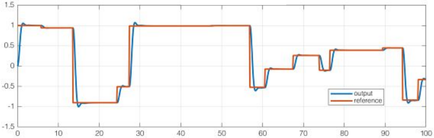
MPC CALIBRATION PROBLEM

- 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**

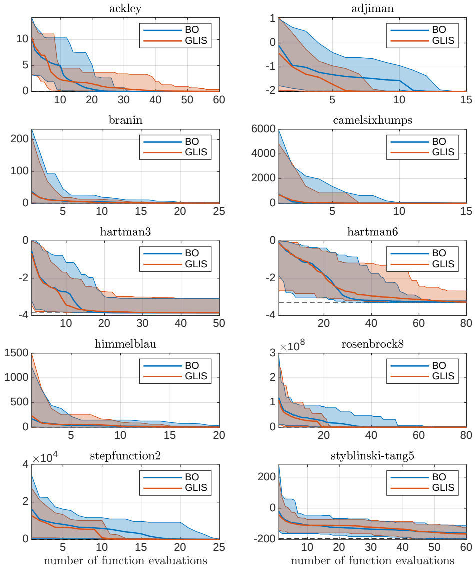
$$\min_x f(x)$$

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`

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

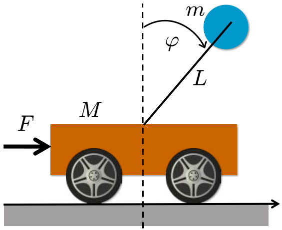
Results computed on 20 runs per test

BO = MATLAB's **bayesopt** fcn

MPC AUTOTUNING EXAMPLE

(Forgione, Piga, Bemporad, 2020)

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



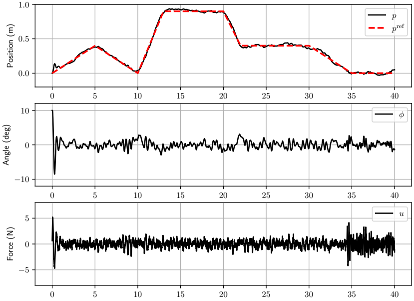
- **sample time**
- **weights** on outputs and input increments
- prediction and control **horizons**
- **covariance** matrices of Kalman filter
- absolute and relative **tolerances** of QP solver

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

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

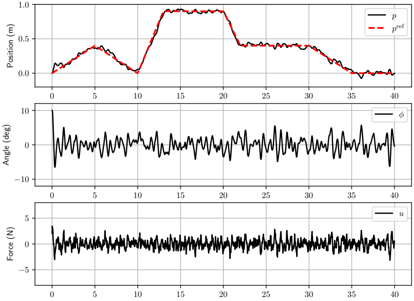
MPC AUTOTUNING EXAMPLE

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:

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

- 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

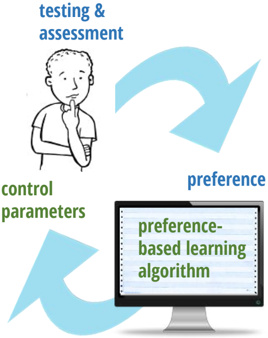
- Use **preference-based optimization** (**GLISp**) algorithm for **semi-automatic tuning** of MPC (Zhu, Bemporad, Piga, 2021)

- 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**

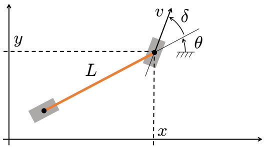


PREFERENCE-BASED TUNING: MPC EXAMPLE

(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:

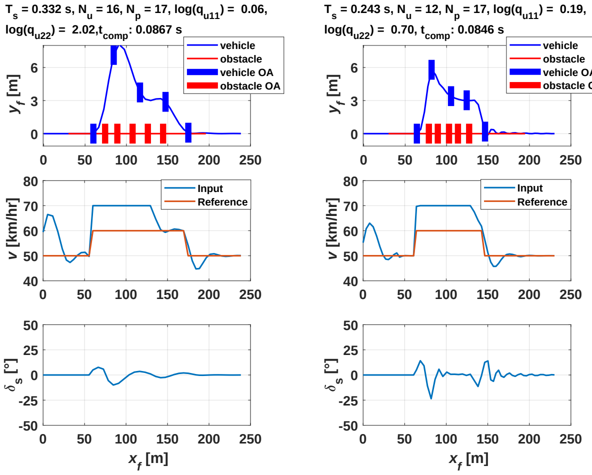
  "optimal obstacle avoidance", *"pleasant drive"*, *"keep CPU time small"*, ...

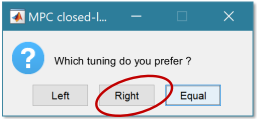
  → **not easy to quantify in a single function**

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

PREFERENCE-BASED TUNING: MPC EXAMPLE

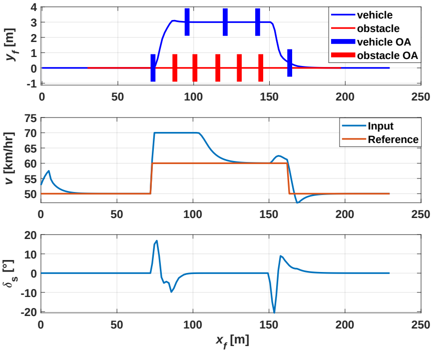
- Preference query window:





PREFERENCE-BASED TUNING: MPC EXAMPLE

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



Optimal MPC parameters:

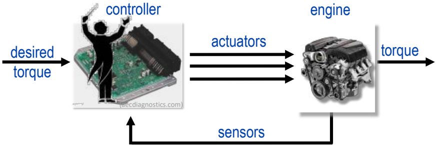
- sample time = 85 ms (CPU time = 80.8 ms)
- prediction horizon = 16
- control horizon = 5
- weight on Δv = 1.82
- weight on $\Delta\delta$ = 8.28

- **Note**: no need to define a closed-loop performance index explicitly!
- Extended to handle also **unknown constraints** (Zhu, Piga, Bemporad, 2021)

CONCLUSIONS

Perspective of the automotive industry:

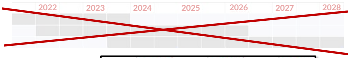
- Increasingly demanding **requirements** (emissions/consumption, passenger safety and comfort, ...)



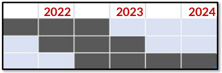
- Better control performance only achieved by better **coordination** of actuators:
  - **increasing number** of actuators (e.g., due to electrification)

  - take into account **limited range** of actuators

  - resilience in case of some **actuator failure**

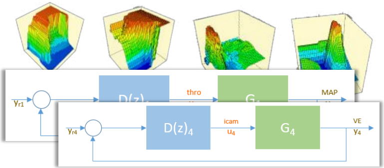


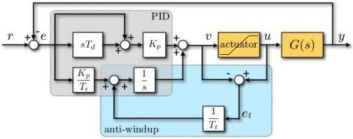
- **Shorter development time** for control solution (market competition, changing legislation)



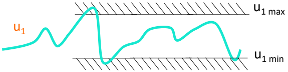
DO WE REALLY NEED MPC?

- Classical control approach:
  - many **single PID loops**
  - **anti-windup** for actuator saturation
  - many **lookup tables**





- Long design & calibration time due to:
  - **complexity** of anti-windup due to **interactions**
  - difficulty to recover from **actuator failure**
  - design space increases **exponentially**
    (e.g.: 5 inputs, 10 values each $\rightarrow 10^5$ entries)
  - hard to **coordinate** multiple actuators optimally
  - porting to different vehicle models may require **substantial recalibration**



(courtesy of J. Verdejo)

CONCLUSIONS

- MPC is a **universal control methodology**:
  - to coordinate **multiple inputs/outputs**, arbitrary **models** (linear, nonlinear, ...)
  - to **optimize performance** index subject to **constraints**
  - it is intuitive to **design/calibrate** and easy to **reconfigure**

- After a long history of success in the **process industries**, MPC is now a mature technology for the **automotive** industry too:
  - modern ECUs can solve MPC problems in **real-time**
  - increasingly tight **requirements** ask for advanced multivariable control solutions
  - advanced MPC **software tools** are available for design/calibration/deployment

CONCLUSIONS

- **Learning-based MPC** is a formidable combination for advanced control:
  - **MPC** / on-line optimization is an extremely powerful control methodology
  - **ML** extremely useful to get **control-oriented models** and **control laws** from **data**
- Ignoring **ML** tools would be a mistake (a lot to "learn" from machine learning)
- **ML** cannot replace control engineering:
  - **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
  - Pure AI-based **reinforcement learning** methods can be also a failure
- A wide spectrum of research opportunities and new practices is open !