Embedded Convex Quadratic Optimization for Model Predictive Control

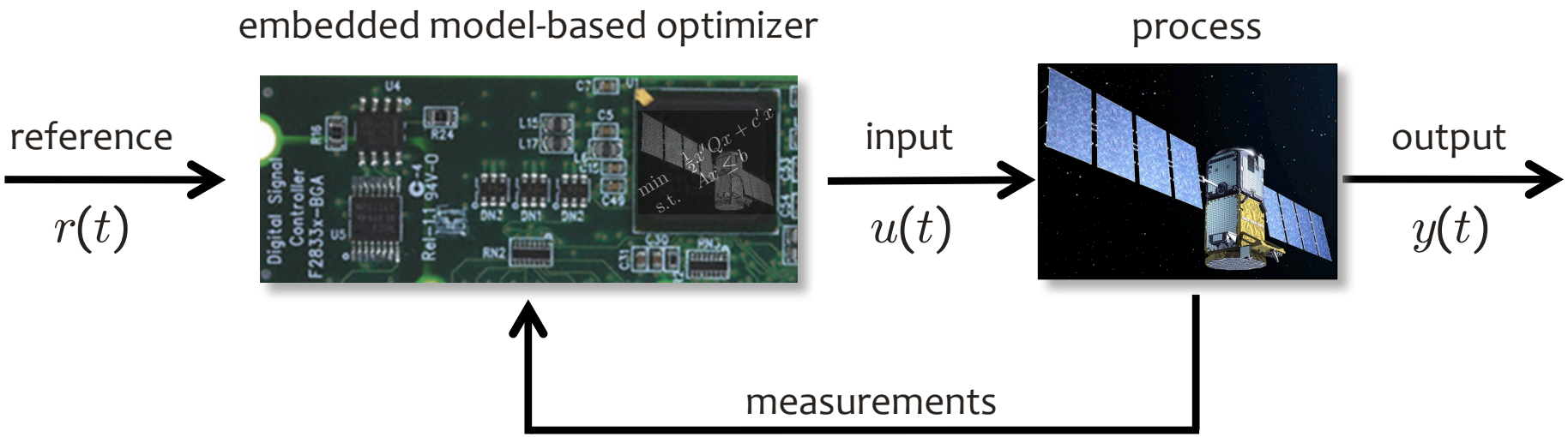
Alberto Bemporad

http://imtlucca.it/alberto.bemporad

(joint work with D. Bernardini, G. Cimini, A. Guiggiani, P. Patrinos, L. Stella)

IMT INSTITUTE FOR ADVANCED STUDIES LUCCA

Model Predictive Control (MPC)



rough — likely

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

Model Predictive Control (MPC)

- At time t: consider optimal control problem over a future horizon of N steps

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

$$\text{s.t. } x_{k+1} = Ax_k + Bu_k$$

$$y_k = Cx_k$$

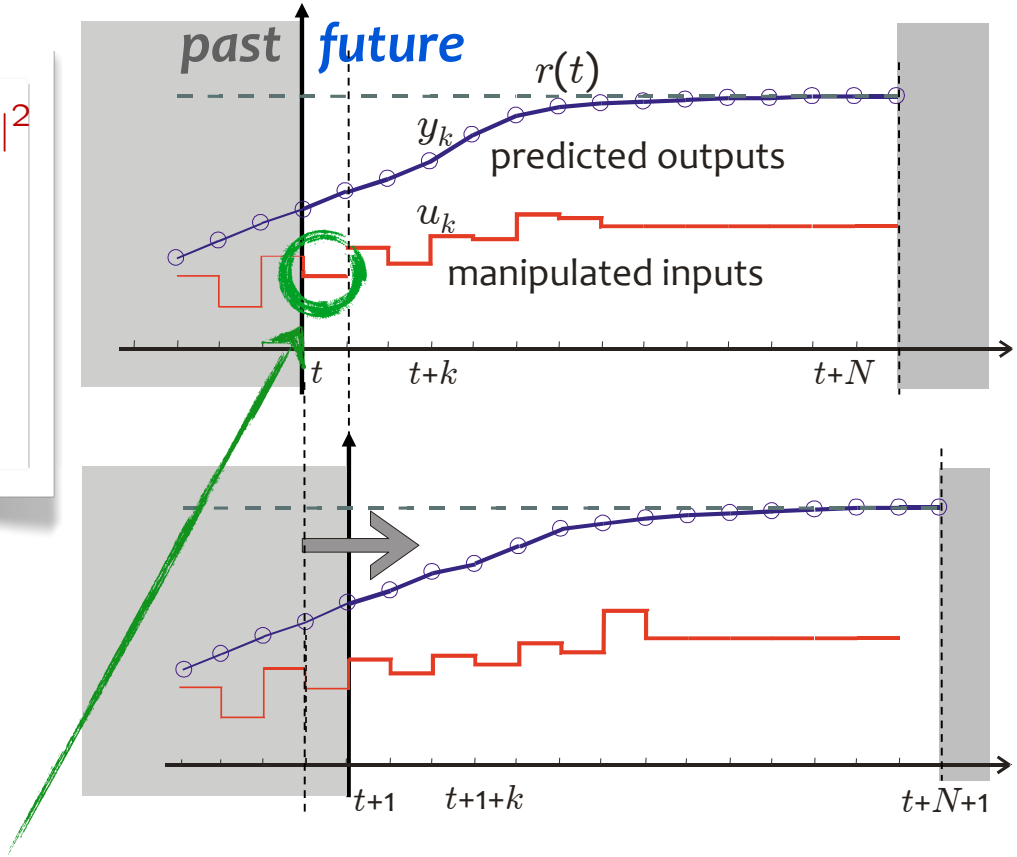
$$u_{\min} \le u_k \le u_{\max}$$

$$y_{\min} \le y_k \le y_{\max}$$

$$x_0 = x(t)$$

← feedback !

→ **Quadratic Program (QP)**



- Solve QP w.r.t. $\{u_0,\ldots,u_{N-1}\}$
- 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 …

Used in process industries since the 80's. Now spreading in auto & aero

Automotive applications of MPC

Bemporad, Bernardini, Borrelli, Di Cairano, Giorgetti, Hrovat, Kolmanovsky, Ripaccioli, Trimboli, Tseng, Yanakiev (2001-2014)

Powertrain
- direct-inj. engine control
- A/F ratio control
- magnetic actuators
- robotized gearbox
- power MGT in HEVs
- cabin heat control in HEVs

Vehicle dynamics
- traction control
- active steering
- semiactive suspensions

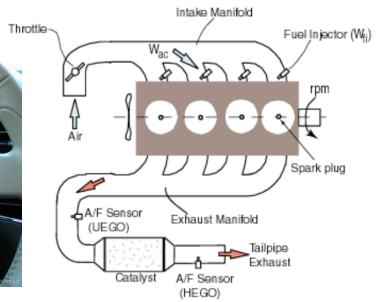
JAGUAR (Assadian, Dextreit)

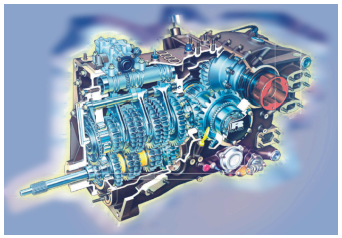
CENTRO RICERCHE FIAT (Borodani, Mannelli)

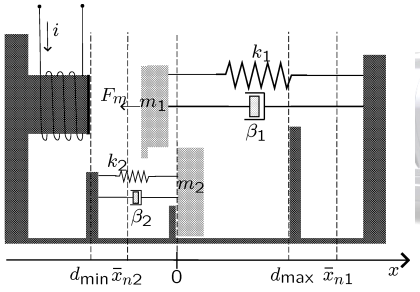
DENSO (Esen,Tashiro)

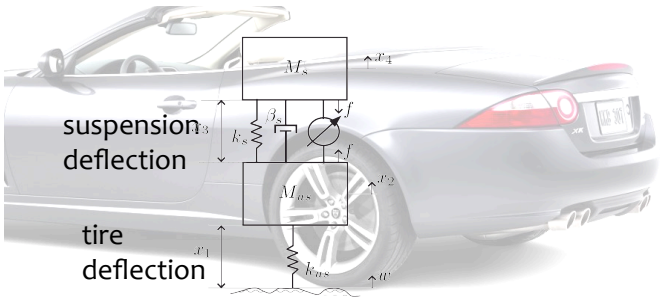
Homogeneous

Stratified





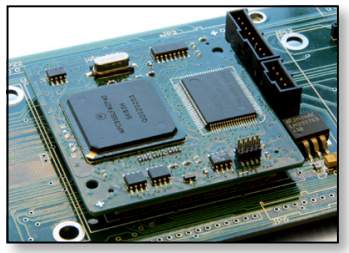




Requirements for embedded optimization code

1. **Speed (throughput)**: fast enough to provide the control decision within short sampling intervals (e.g., 10-100 ms)

2. Require **simple/cheap hardware** (microcontroller, microprocessor, FPGA) and **little memory** to store problem **data *and* code**



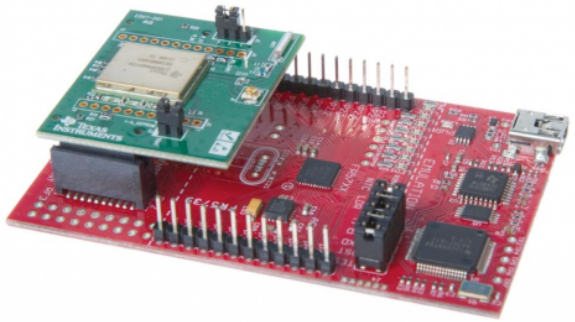
3. **Worst-case execution time** must be (tightly) estimated for embedding the controller in a **real-time platform**

4. **Code simple** enough to be **verifiable/certifiable** (or at least understandable by production engineers)

```
    # Main iteration loop
    while keepgoing & (nu<maxiter):

        # Compute theta, beta
        beta=th*(1/th0-1)

        w=y+beta*(y-y0)
        zhat=-(dot(MG,w)+gP)
        if -onlyzhat:
            z=(1-th)*z+th*zhat
        th0=th # Update th0
        th2=th**2
        th=(sqrt(th2**2+4.0*th2)-th2)/2.0
```

Contents of my talk

- Explicit MPC and **multiparametric quadratic programming**
- Recent advances in embedded **quadratic programming**
  - Feasibility-based methods
  - Fixed-point implementations
- Simulations and **hardware-in-the-loop** results, comparing different embedded QP solvers

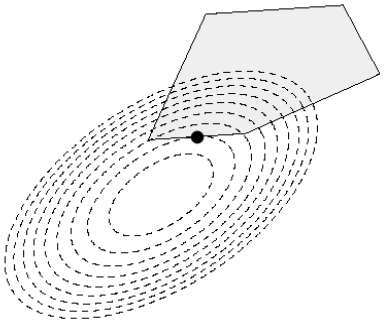


Embedded Linear MPC

- Linear MPC requires solving a **Quadratic Program (QP)**

$$\begin{aligned} \min_z \quad & \frac{1}{2}z'Hz + x'(t)F'z + \frac{1}{2}x'(t)Yx(t) \\ \text{s.t.} \quad & Gz \le W + Sx(t) \end{aligned}$$

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



The control law $u(t)=u_0^*(x(t))$ is **implicitly** defined by the QP solver

- A variety of algorithms exist to solve the QP **on-line** given $x(t)$:

  active set (AS), interior point (IP), gradient projection (GP), alternating direction method of multipliers (ADMM), proximal methods, ...

- **Explicit MPC**: pre-solve the QP **off-line for all** $x(t)$ to find the control law $u_0^*(x)$ **explicitly** via **multiparametric quadratic programming** (mpQP)

Properties of multiparametric QP

Theorem: Assume $H \succ 0$, $\begin{bmatrix} H & F \\ F' & Y \end{bmatrix} \succeq 0$ (always satisfied in QPs from MPC).

$$X^* = \{x \in \mathbb{R}^n : \ z^*(x) \text{ exists}\} \longrightarrow \text{polyhedron}$$

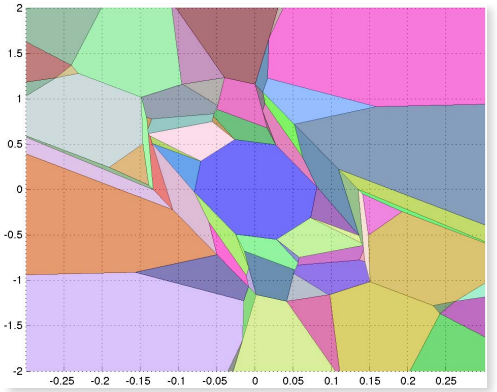
$$z^*(x) = \arg\min_z \ \tfrac{1}{2} z'Hz + x'F'z \quad \text{s.t.} \quad Gz \le W + Sx \longrightarrow \text{continuous, piecewise affine}$$

$$V^*(x) = \tfrac{1}{2} x'Yx + \min_z \ \tfrac{1}{2} x'Hz + x'F'z \quad \text{s.t.} \quad Gz \le W + Sx \longrightarrow \text{convex, continuous, piecewise quadratic, (even } C^1\text{, if no degeneracy)}$$

(Bemporad, Morari, Dua, Pistikopoulos, 2002)

Corollary: The linear MPC control law is continuous and piecewise affine

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



Multiparametric QP solvers

- A variety of mpQP solvers is available

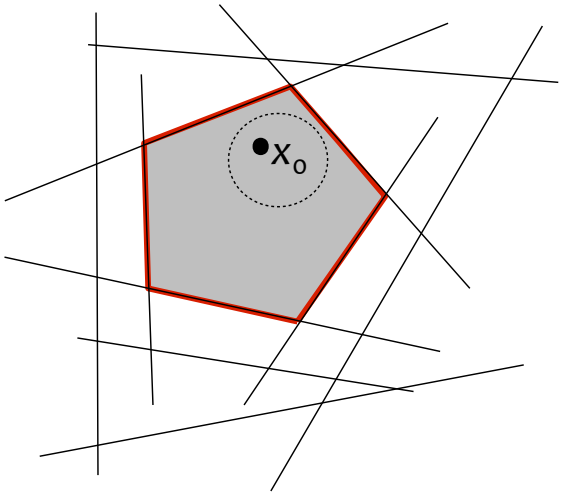
(Bemporad *et al.*, 2002) (Baotic, 2002)
(Tøndel, Johansen, Bemporad, 2003)
(Spjøtvold *et al.*, 2006)(Patrinos, Sarimveis, 2010)

- Most computations are spent in operations on polyhedra (=critical regions)

$$\begin{aligned} \hat{G}z^*(x) &\leq \hat{W} + \hat{S}x \\ \tilde{\lambda}^*(x) &\geq 0 \end{aligned}$$

← feasibility of primal solution
← feasibility of dual solution

- checking emptiness of polyhedra
- removal of redundant inequalities
- checking full-dimensionality of polyhedra



- All such operations are usually done via **linear programming (LP)**

A new mpQP algorithm based on NNLS

(Bemporad, 2014)

- New approach: use **Nonnegative Least Squares** (NNLS) for polyhedral computations

$$\begin{aligned} \min_v \quad & \|Av - b\|_2^2 \\ \text{s.t.} \quad & v \geq 0 \end{aligned}$$

efficiently solved e.g. via active set algorithm

(Lawson, Hanson, 1974)

- Partially Nonnegative Least Squares (PNNLS) problem

$$\begin{aligned} \min_{v,u} \quad & \|Av + Bu - c\|_2^2 \\ \text{s.t.} \quad & v \geq 0, \ \ u \text{ free} \end{aligned}$$

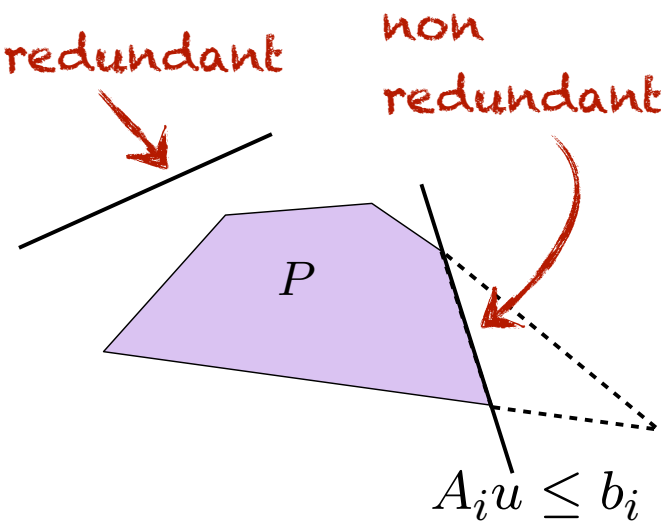
PNNLS = pseudo-inverse of B + solve NNLS

- Key result:

The polyhedron $P = \{u \in \mathbb{R}^n : \ Au \leq b\}$ *is nonempty iff the PNNLS problem*

$$\begin{aligned} (v^*, u^*) = \arg\min_{v,u} \quad & \|v + Au - b\|_2^2 \\ \text{s.t.} \quad & v \geq 0, u \text{ free} \end{aligned}$$

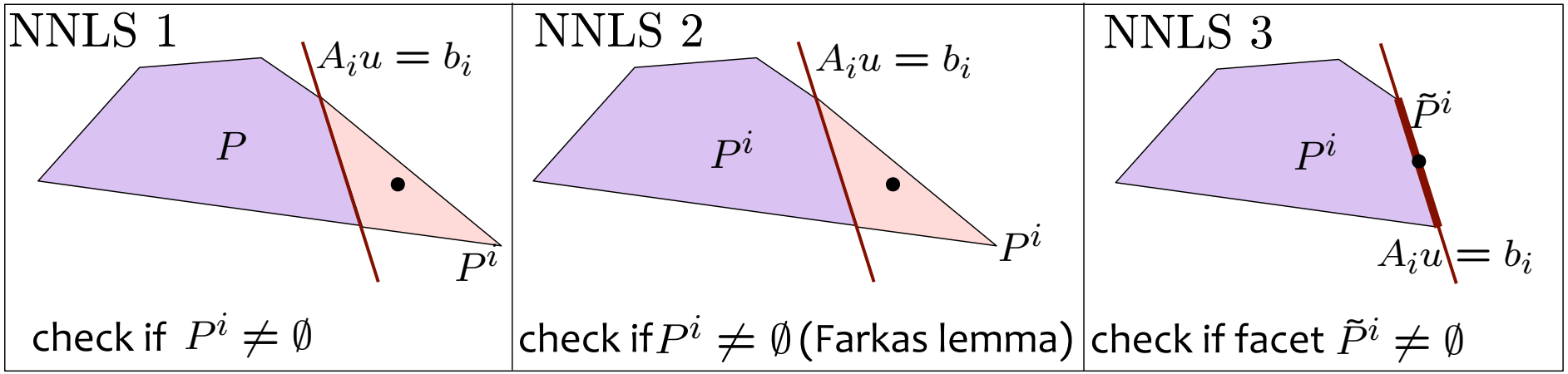
has zero residual $\|v^* + Au^* - b\|_2^2 = 0$



Polyhedral computations based on NNLS

(Bemporad, 2014)

- Three possible methods to check inequality redundancy via NNLS:



- Numerical results

m	NNLS 1	NNLS 2	NNLS 3	LP
2	0.0051	0.0035	0.0006	0.0046
4	0.0351	0.0301	0.0019	0.0103
6	0.1297	0.1233	0.0038	0.0193
8	0.4046	0.3976	0.0071	0.0340
10	1.0185	1.0082	0.0111	0.0554
12	2.0665	2.1030	0.0178	0.0955
14	4.5279	4.5546	0.0263	0.1426
16	7.9159	7.9133	0.0357	0.1959

NNLS = compiled Embedded MATLAB
LP = compiled C code (GLPK)

CPU time = seconds (this Mac)

random polyhedra of $\mathbb{R}^m$ with $10m$ inequalities

- Other polyhedral operations can be also tackled by NNLS
  (full-dimensionality check, Chebychev radius, union, projection)

A new mpQP algorithm based on NNLS

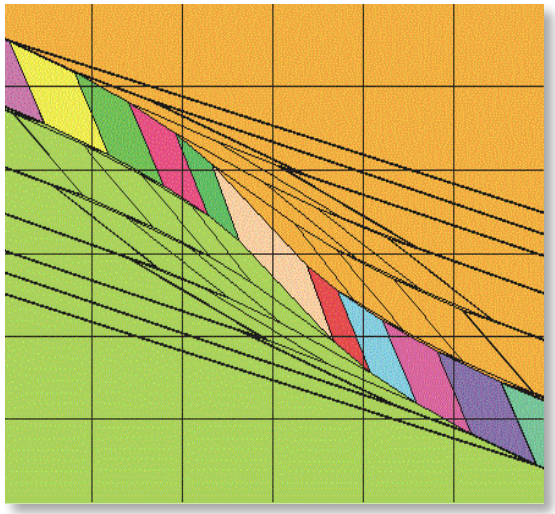
- New mpQP algorithm based on **NNLS** + **dual QP formulation** to compute active sets and deal with degeneracy (Bemporad, 2014)

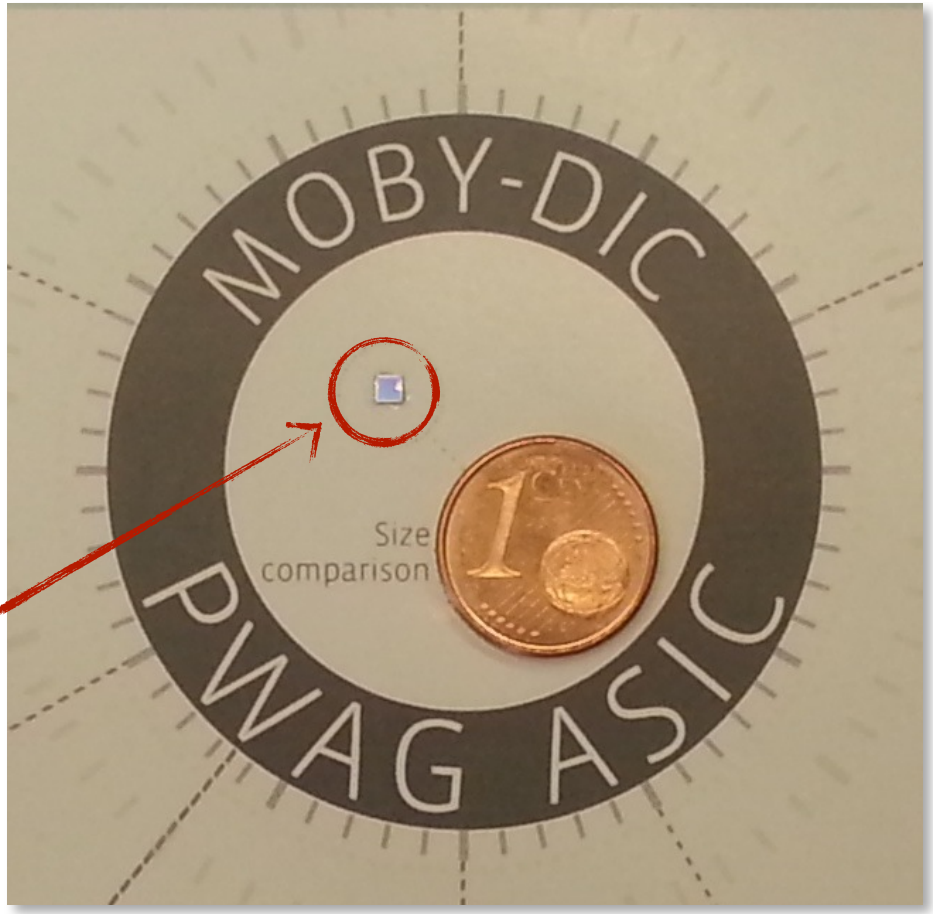
- Comparison with:
  - Hybrid Toolbox (Bemporad, 2003)
  - Multiparametric Toolbox 2.6 (with default opts) (Kvasnica, Grieder, Baotic, 2006)

q	m	Hybrid Tbx	MPT	NNLS
4	2	0.0174	0.0256	0.0026
4	3	0.0263	0.0356	0.0038
4	4	0.0432	0.0559	0.0061
4	5	0.0650	0.0850	0.0097
4	6	0.0827	0.1105	0.0126
8	2	0.0347	0.0396	0.0050
8	3	0.0583	0.0680	0.0092
8	4	0.0916	0.0999	0.0140
8	5	0.1869	0.2147	0.0322
8	6	0.3177	0.3611	0.0586
12	2	0.0398	0.0387	0.0054
12	3	0.1121	0.1158	0.0191
12	4	0.2067	0.2001	0.0352
12	5	0.6180	0.6428	0.1151
12	6	1.2453	1.3601	0.2426
20	2	0.1029	0.0763	0.0152
20	3	0.3698	0.2905	0.0588
20	4	0.9069	0.7100	0.1617
20	5	2.2978	1.9761	0.4395
20	6	6.1220	6.2518	1.2853

- Included in MPC Toolbox 5.0 (R2014b)

Hardware (ASIC) implementation of explicit MPC





Technology: 90 nm, 9 metal-layer from Taiwan Semiconductor Manufacturing Company

Chip size: 1860x1860 μm²

Memory sizes: 24 KB (tree memory) ; 30 KB (parameter memory)

Power supply: 2.5V (ring); 1.2V (core)

Maximum frequency: 107.5 MHz (with two inputs)

Power consumption: 38.08 mW@107.5MHz;

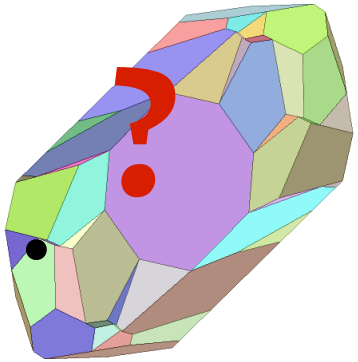
$$\frac{1}{107.5\ \text{MHz}} = 9.3\ \text{ns}$$

SEVENTH FRAMEWORK PROGRAMME

http://www.mobydic-project.eu/

Complexity of multiparametric solutions

- The number of regions depends (exponentially) on the number of possible **combinations of active constraints**
- Weak dependence on the number of states and references
- Explicit MPC gets less attractive when number of regions grows: too much **memory** required, too much **time** to locate state $x(t)$
- Fast and simple **on-line QP solvers** may be preferable



Linear time-varying MPC

- MPC can easily extend to **linear time-varying (LTV)** problems

$$\begin{cases} x_{k+1} = A_k(t,x(t))x_k + B_k(t,x(t))u_k + f_k(t,x(t)) \\ y_k = C_k(t,x(t))x_k + D_k(t,x(t))u_k + g_k(t,x(t)) \end{cases}$$

prediction model

$$E_k(t,x(t))x_k + F_k(t,x(t))u_k \le h_k(t,x(t))$$

constraints

$$k = 0, 1, \ldots, N-1$$

$$\min \sum_{k=0}^{N} \ell_k(y_k, u_k, r(t+k), t, x(t))$$

cost function

ℓ_k = quadratic function of y_k, u_k

$$\begin{array}{ll} \min\limits_{z} & \frac{1}{2} z' H(t) z + F(t)' z + \alpha(t) \\ \text{s.t.} & G(t) z \le W(t) \end{array}$$

LTV-MPC still leads to a convex QP

- **QP matrices must be constructed on line,** H and G are a function of t

Reqs for embedded optimization algorithms

- Distinguish between **off-line** and **on-line** computations
  - **off-line:** double precision (MATLAB/Python/Julia) on a PC (arbitrarily complex)
  - **on-line:** single-precision/fixed-point on a μ-processor (please simple!)
- Have degrees of freedom to trade off **throughput** vs. **memory**
- Typical problem size: **10÷50 variables, 10÷100 constraints**
- **Numerical robustness** (of on-line computations): must perform well in single precision (or even fixed-point) computations

Reqs for embedded optimization algorithms

- **Limited linear algebra** operations (in on-line computations) for code simplicity (example: at most matrix-vector products)

- **Dense problems**, no need for sparse linear algebra

- **Suboptimal solutions are ok.** Optimality is of less concern than feasibility (the prediction model is approximate, the cost function usually comes from trial & error tuning)

Reqs for embedded optimization algorithms

- Need to be able to estimate the **worst-case number of iterations**:
  - **Iterative methods,** for a given solution quality, are ok as long as the number of iterations can be well predicted (in theory and/or in experiments)
  - **Active-set methods** provide best solution quality (within machine precision)
- Insensitivity to **problem scaling** (or have good automatic scaling methods). Many QP problems from MPC are badly scaled (e.g.: soft constraints = high penalties)
- The simpler the code, the easier is to exploit super-optimized processor-specific functions (e.g.: scalar products)

Gradient projection method for QP (very simple!)

(Goldstein, 1964) (Levitin & Poljak, 1965)

- QP problem:

$$\begin{aligned} \min_z \quad & \frac{1}{2} z'Hz + x'Fz \\ \text{s.t.} \quad & Gz \le W + Sx \end{aligned}$$

$z =$ optimization vector

$x =$ parameter vector

- Apply gradient-projection to **dual QP**

$$\min_{y \ge 0} \quad \frac{1}{2} y'My + (Dx + W)'y$$

$$\begin{aligned} M &= GH^{-1}G' \\ D &= GH^{-1}F' + S \end{aligned}$$

prepared off-line

L = max eigenvalue of M, or $L = \sqrt{\sum_{i,j=1}^{m} |M_{i,j}|^2}$ (Frobenius norm)

- Iterate $y_{k+1} = \max\{(I - \frac{1}{L}M)y_k - \frac{1}{L}(Dx + W), 0\}$ from $y_0=0$

  until convergence to optimal y^* (guaranteed if QP is feasible)

- Set $z^* = -H^{-1}(G'y^* + F'x)$

Fast gradient projection for (dual) QP

(Nesterov, 1983)
(Patrinos, Bemporad, IEEE TAC, 2014)

- Apply fast gradient method to dual QP:

$$\begin{aligned} w_k &= y_k + \beta_k(y_k - y_{k-1}) \\ z_k &= -Kw_k - Jx \\ s_k &= \tfrac{1}{L}Gz_k - \tfrac{1}{L}(Sx + W) \\ y_{k+1} &= \max\{y_k + s_k, 0\} \end{aligned}$$

$$\begin{aligned} K &= H^{-1}G' \\ J &= H^{-1}F' \end{aligned} \qquad y_{-1} = y_0 = 0$$

$$\beta_k = \begin{cases} 0 & k = 0 \\ \frac{k-1}{k+2} & k > 0 \end{cases}$$

- Termination criterion #1: **primal feasibility**

feasibility tol

$$s_k^i \le \frac{1}{L}\epsilon_G, \quad \forall i = 1, \ldots, m$$

- Termination criterion #2: **primal optimality**

$$f(z_k) - f^* \le f(z_k) - \phi(w_k) = -w_k' s_k L \le \epsilon_V$$

dual function

optimality tol

$$-w_k' s_k \le \frac{1}{L}\epsilon_V$$

Fast gradient projection for (dual) QP

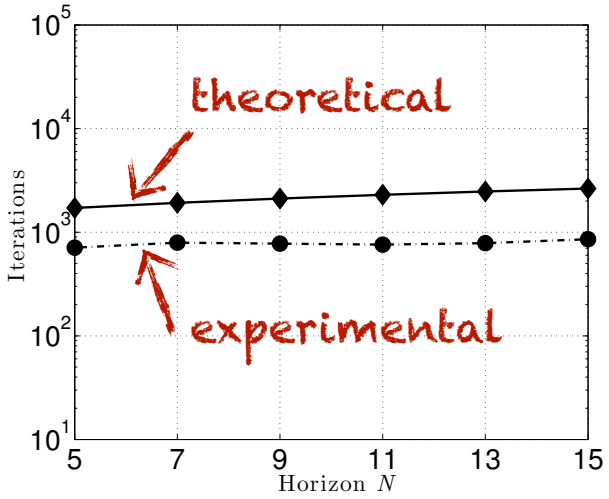
(Patrinos, Bemporad, IEEE TAC, 2014)

- Main on-line operations involve only **simple linear algebra**

```
while keepgoing && (i<maxiter),

    beta=(i-1)/(i-2).*(i>0);

    w=y+beta*(y-y0);
    z=-(iMG*w+iMc);
    s=GLz-bL;

    y0=y;

    % Check termination conditions
    if all(s<=epsGL),
        gapL=-w'*s;
        if gapL<=epsVL,
            return
        end
    end

    y=max(w+s,0);

    i=i+1;
end
```

- For MPC problems, using Riccati-like iterations complexity per iteration is $O(N)$
  $[N = \text{prediction horizon}]$

- Tight bounds on maximum number of iterations



- Can be very useful to warm-start other methods !

Primal vs. dual methods

- **Early stopping**: primal methods usually provide a feasible solution if stopped prematurely (e.g.: max time elapsed, task pre-emption), dual methods do not (infeasible & super-optimal primal solutions)

- Better **control of primal solution quality** with primal methods

- **Warm starting** easier in primal methods (shifted previous optimal solution $[u_1^*(x(t\text{-}1))\ \ldots\ u_{N\text{-}1}^*(x(t\text{-}1)),\ *]$ is often a good initial guess), more difficult with dual methods.

  E.g.: in LCP methods $w=Mz+d$, an easy initial guess is $z=0,\ w=d$, that satisfies the complimentarity constraints.

Primal vs. dual methods

- Dual methods can usually handle **more general linear constraints** ($Az \leq b$ vs. $l \leq z \leq u$)

- Often **box-constraints** $l \leq z \leq u$ are enough, so no need to go to dual QP. Example: **soft constraints** can be remapped into cost function:

$$Az \leq b \quad \longrightarrow \quad \min \ \ \rho \|Az + s - b\|^2, \ \ s \geq 0, \ \ \rho \gg 1$$

- Primal Hessian H always **>0**, dual Hessian $M = GH^{-1}G'$ usually only ≥ 0

- Dual Hessian $GH^{-1}G'$ may be **larger to store** than H and G

Primal vs. dual methods

- Sometimes **reconstructing primal solution** after solving dual problem is numerically troublesome (a small error in dual solution can propagate to a large error in primal solution)

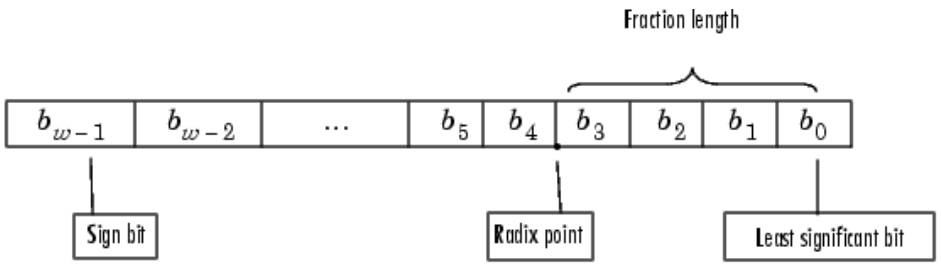
- Dual methods more difficult for **LTV** (need to compute $GH^{-1}G'$ on-line)

Whether to prefer primal or dual method depends on constraint type ($Ax \leq b$ vs. $l \leq x \leq u$) and LTI vs. LTV MPC control problem setup

Finite-precision arithmetics

How about numerical robustness ?

- **Fixed-point arithmetics** is very attractive for embedded control:
  - Computations are fast and cheap
  - Hardware support in all platforms



- Cons:
  - Accumulation of quantization errors
  - Limited range (numerical overflow)

Gradient projection in fixed-point arithmetics

(Patrinos, Guiggiani, Bemporad, 2013)

- Convergence to feasibility

$$\max_i g_i(z_k) \le \frac{2LD^2}{k+1} + L_v\epsilon_z^2 + 4D\epsilon_\xi$$

exponentially decreasing with p

- Convergence to optimality

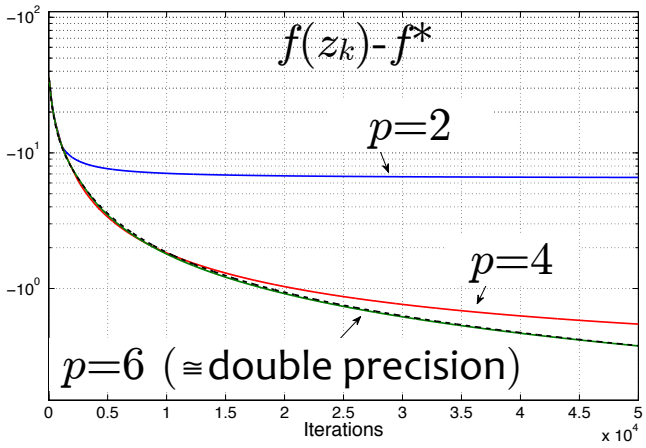
$$f(z_k) - f^\star \le \frac{L}{2(k+1)}(\|y^\star\|^2 + \|y_0\|^2) + \delta$$

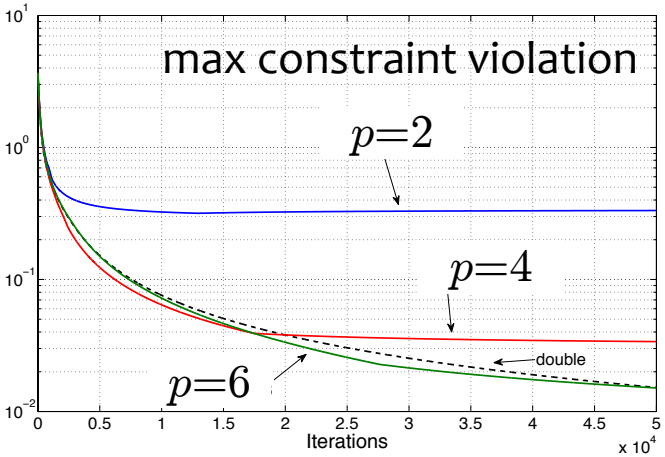
- Design guidelines (precision)

$$p \ge \log_2 \frac{m\sqrt{n}}{\sqrt{\frac{\varepsilon}{L_V} + \frac{n}{m}\left(\frac{2D}{L_V}\right)^2} - \sqrt{\frac{n}{m}}\frac{2D}{L_V}} - 1$$

fractional bits

desired solution "quality"





- Design guidelines for required #**integer bits** to avoid **overflow** also available

Hardware tests (floating vs fixed point)

(Patrinos, Guiggiani, Bemporad, 2013)

32-bit Atmel SAM3X8E **ARM Cortex-M3** processing unit:
84 MHz, 512 KB of flash memory and 100 KB of RAM.

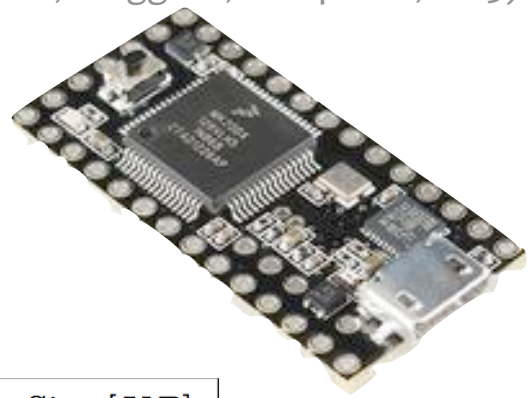


Table 1
Fixed-point hardware implementation

Size [variables/constraints]	Time $[ms]$	Time per iteration $[\mu s]$	Code Size $[KB]$
10/20	22.9	226	15
20/40	52.9	867	17
40/80	544.9	3382	27
60/120	1519.8	7561	43

fixed point

Table 2
Floating-point hardware implementation

Size [variables/constraints]	Time $[ms]$	Time per iteration $[\mu s]$	Code Size $[KB]$
10/20	88.6	974	16
20/40	220.1	3608	21
40/80	2240	13099	40
60/120	5816	30450	73

floating point

fixed-point about 4x faster than floating-point

Proximal Newton methods (fixed-point)

- Proximal Netwon method applied to **box-constrained QP**

(Patrinos, Guiggiani, Bemporad, 2014)

$$\min_{\underline{z} \le z \le \bar{z}} \quad \frac{1}{2} z'Qz + q'z$$

- Bound on **round-off error** accumulated at each iteration

$$\|z - \texttt{fi}(z)\| \le \left(1 + \kappa(Q)\|Q^{-1}\| \, (Nn_u)^{\frac{3}{2}}\right) 2^{-(p+1)}$$

condition number

prediction horizon

fractional bits

number of inputs

- Number of integer bits to avoid **overflow**

$$r \ge \log_2 \left(\max\left\{\hat{z}, \hat{d}\right\} + 1\right) + 1$$

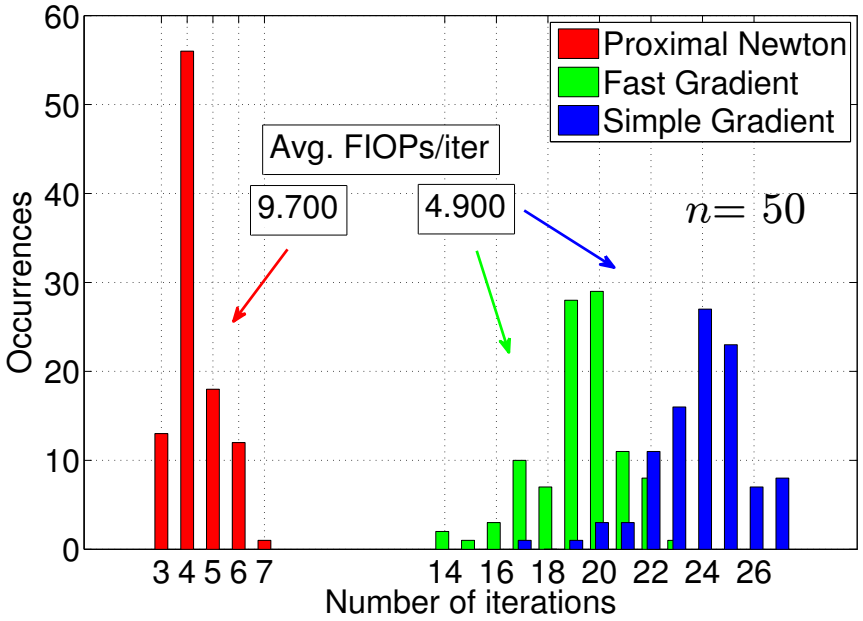
integer bits

$$\max\left\{\|\underline{z}\|_\infty, \|\bar{z}\|_\infty\right\} + \varepsilon_z \qquad \|Q^{-1}\|_\infty \left(\|Q\|_\infty \hat{z} + \|q\|_\infty\right) + \varepsilon_d$$

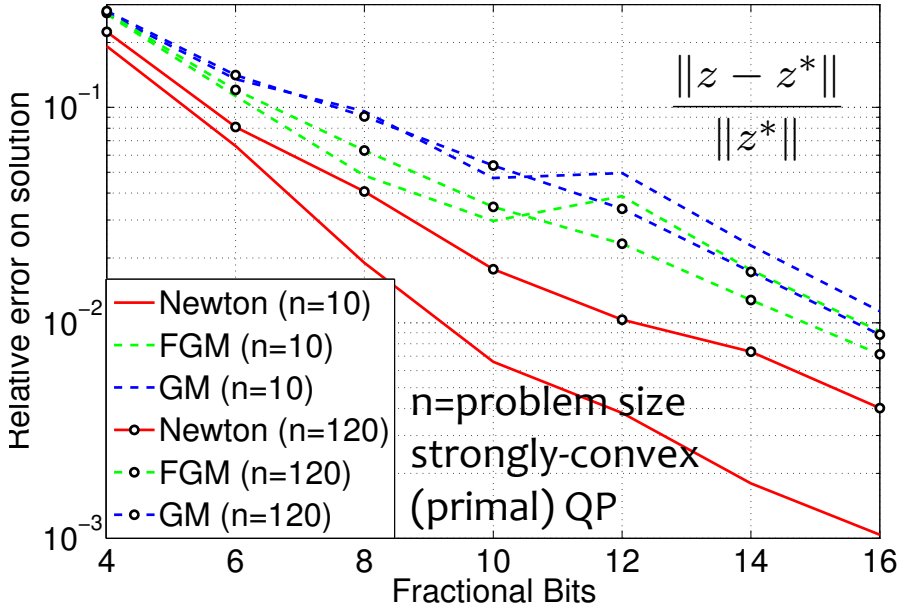
Proximal gradient methods (fixed-point)

(Patrinos, Guiggiani, Bemporad, 2014)

fixed-point computations



relative accuracy



Hardware implementation

random box-constrained QPs

fixed point: 8+8 bits
floating point: 32 bits

	fixed	floating	fixed	floating
Vars.	T_{fi} $[ms]$	T_{fl} $[ms]$	$Size_{fi}$ $[KB]$	$Size_{fl}$ $[KB]$
10	0.6	1.8	14.7	20.2
20	1.8	5	16	22
30	4.7	9.7	18	27
40	8.8	45.5	21	32.8
50	14.7	89.8	24.8	39.6
60	25.7	100.8	29.3	59.2
70	44.8	n/a	34.6	n/a
80	52.4	n/a	40.7	n/a

ARM-based Cortex-M3 (Atmel SAM3X8E)
84 MHz, 100 KB of RAM and 512 KB of flash memory

Anytime Embedded Optimization

What if the CPU time to solve QP exceeds sampling interval or optimization task is preempted ?

- Example: dual QP methods do not get a feasible solution until the end
- **Anytime optimization** idea:

  Given a feasible solution, keep improving its optimality as long as CPU is available, otherwise stop and apply best solution found so far

- Key ingredients needed:
  - Proper **solver** that recursively improves optimality while maintaining feasibility
  - Proper **MPC setup**: a feasible solution always immediately available, and feasibility implies stability

Feasibility-based methods

(Bemporad, Bernardini, Patrinos, 2014)

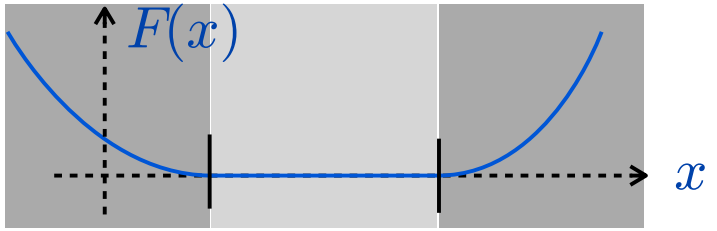
- Convex feasibility problem: find a vector x such that

$$f_i(x) \leq 0, \;\; i = 1, \ldots, m \qquad\qquad f_i : \mathbb{R}^n \mapsto \mathbb{R} \text{ convex}$$

- Key idea: solve unconstrained problem

$$\min \;\; F(x) = \frac{1}{2} \sum_{i=1}^{m} \max\{f_i(x), 0\}^2$$

$F : \mathbb{R}^n \mapsto \mathbb{R}$ convex
and continuously differentiable



- Use Newton-like method to minimize $F(x)$

$$\nabla F_I(x) = \textstyle\sum_{i \in I} f_i(x) \nabla f_i(x)$$

$$\nabla^2 F_I(x) = \sum_{i \in I} \nabla f_i(x) \nabla f_i(x)' + f_i(x) \nabla^2 f_i(x)$$

→ very simple to compute if f_i's are linear or quadratic

$$I = \{i = 1, \ldots, m \text{ such that } f_i(x) > 0\}$$

Feasibility-based methods

(Bemporad, Bernardini, Patrinos, 2014)

- Feasibility algorithm (regularized piecewise-smooth Newton method)

Input: $\sigma \in (0,1/2)$, $\zeta \in (0,1)$, $x^0 \in \mathbb{R}^n$, $k = 0$

if $F(x^k) = 0$ **then**
  empty←false, $x \leftarrow x^k$; **exit**
else if $\|\nabla F(x^k)\| = 0$ **then**
  empty=true; **exit**
end
Compute $d^k \in \mathbb{R}^n$ that solves

$$(\nabla^2 F_{I^k}(x^k) + \delta_k I)d = -\nabla F(x^k), \quad (14)$$

where $I_k = \{i \in [m] \mid f_i(x^k) \geq 0\}$, $\delta_k = \zeta\|\nabla F(x^k)\|$.
Compute $\tau_k = \max\{2^{-i} \mid i = 0, 1, 2, \ldots\}$ such that

$$F(x^k + \tau_k d^k) \leq F(x^k) + \sigma\tau_k \nabla F(x^k)' d^k. \quad (15)$$

$x^{k+1} \leftarrow x^k + \tau_k d^k$
$k \leftarrow k + 1$ and go to Step 1.

F(x)=0, feasible point found

stationary point with F(x)>0, problem infeasible

solve linear system (usually of small dimension)

line search (Armijo's rule)

Feasibility-based methods

(Bemporad, Bernardini, Patrinos, 2014)

- Convex optimization problem:

$$\begin{array}{ll}\min & f_0(x)\\ \text{s.t.} & f_i(x)\le 0,\ \ i=1,\dots,m\end{array}$$

- Solve feasibility problems recursively (bisection) by updating upper and lower bounds on optimal cost:

1. $\bar{f}_k = \dfrac{UB_{k-1}+LB_{k-1}}{2}$

bound on optimality

2. solve feasibility problem $\begin{array}{l} f_0(x)\le \bar{f}_k\\ f_i(x)\le 0,\ \ i=1,\dots,m\end{array}$

3. If feasible, set $\begin{array}{ll} UB_k &= \bar{f}_k\\ LB_k &= LB_{k-1}\end{array}$ otherwise set $\begin{array}{ll} LB_k &= \bar{f}_k\\ UB_k &= UB_{k-1}\end{array}$

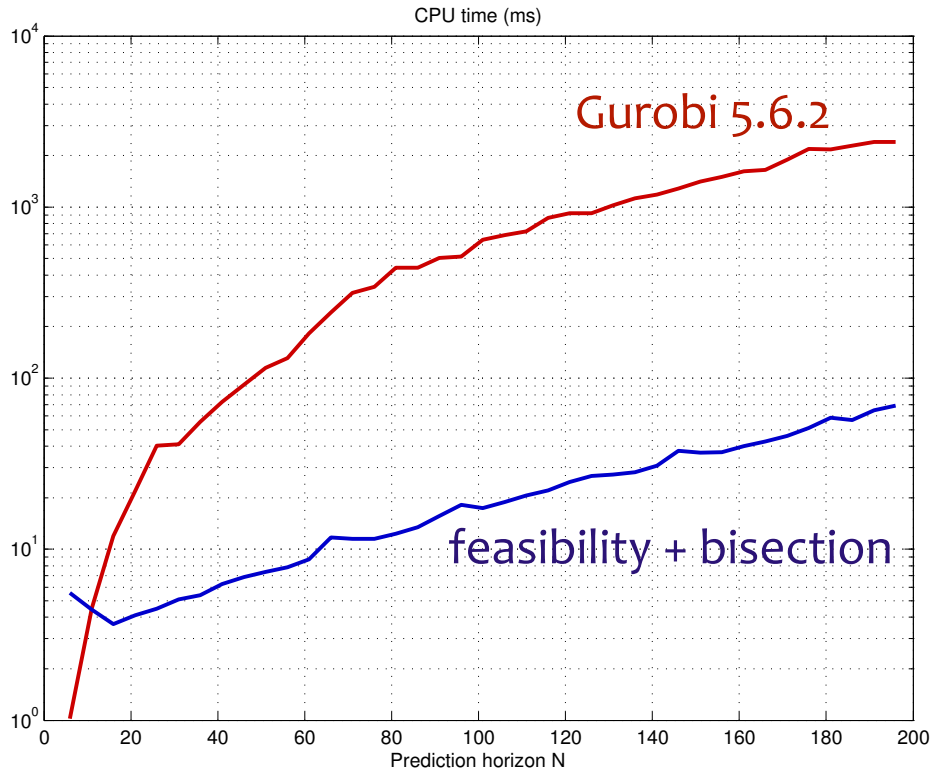
4. Repeat from step 1. until $UB_k - LB_k \le \epsilon$

- Better improvements of upper and lower bounds are possible

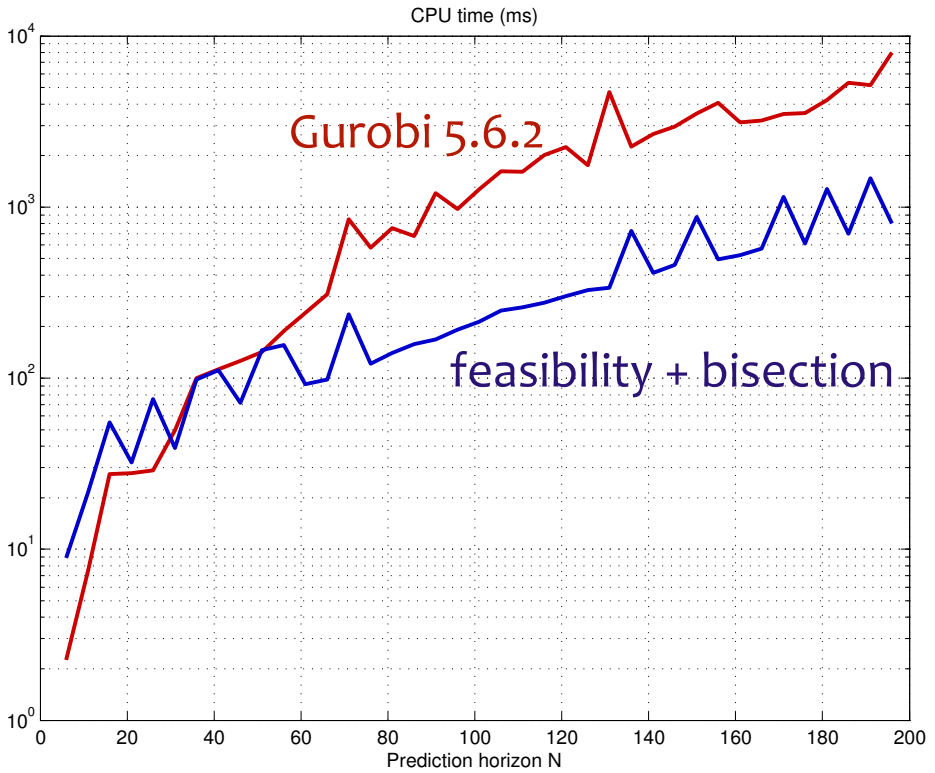
Feasibility-based methods

(Bemporad, Bernardini, Patrinos, 2014)

- Numerical results



QP problems
(arising from MPC)

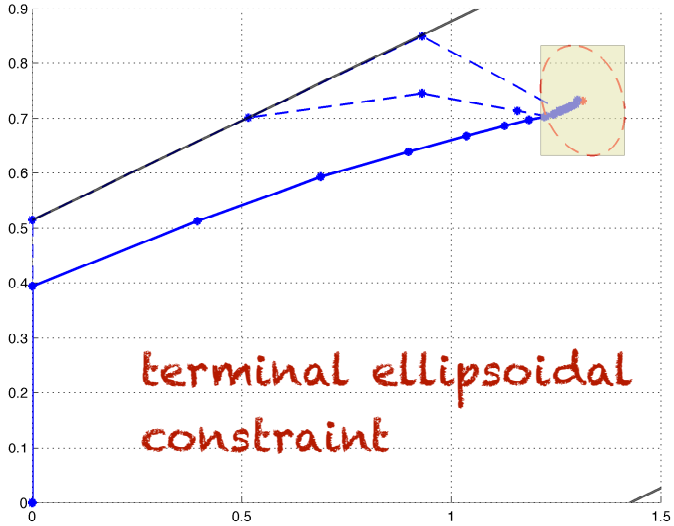


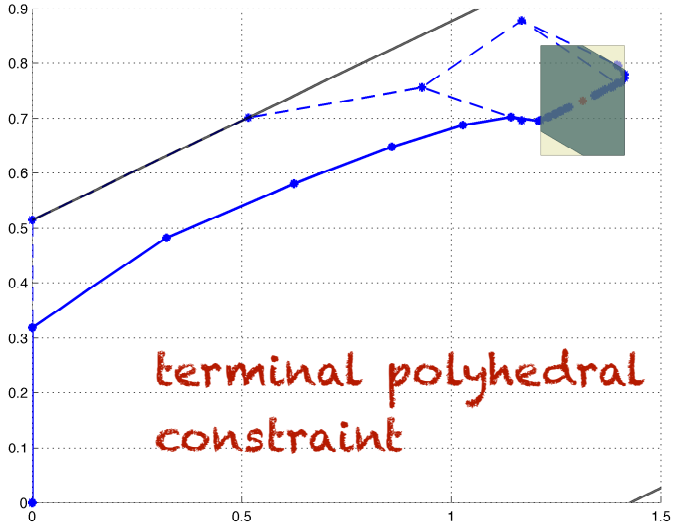
Quadratically Constrained QP problems
(arising from MPC)

Anytime embedded MPC

(Bemporad, Bernardini, Patrinos, 2014)

- **Key idea:** only keep bisecting if CPU resources are available, otherwise stop and apply current best feasible solution





max CPU time (ms)	Cumulated cost J
unbounded	8.7865
60	22.6572
40	26.4663
20	31.4528

max CPU time (ms)	Cumulated cost J
unbounded	9.0075
10	23.4491
5	26.4646
1	26.7551

Convergence guaranteed whatever CPU time is available at each step !

Scaling (or preconditioning)

- **Preconditioning** can improve convergence rate of iterative algorithms (in particular first-order methods are very sensitive to scaling)

primal QP

$$\begin{aligned} \min_z \quad & \frac{1}{2} z'Hz + f'z \\ \text{s.t.} \quad & Gz \le W \end{aligned} \qquad \begin{aligned} M &= GH^{-1}G' \\ d &= GH^{-1}f + W \end{aligned}$$

- **Dual scaling:**

$$\min_y \ \frac{1}{2} y'My + d'y \quad \text{s.t.} \ \ y \ge 0$$

scaling $y = Py_s$

$$P = \text{diag}\left(\frac{1}{\sqrt{M_{ii}}}\right)$$

$$\min_{y_s} \ \frac{1}{2} y_s'(PMP)y_s + d'Py_s \quad \text{s.t.} \ \ y_s \ge 0$$

scaled dual QP

- This is equivalent to scale constraints in primal problem: $\frac{1}{\sqrt{M_{ii}}} G_i z \le \frac{1}{\sqrt{M_{ii}}} W_i$

- Primal solution: $z^* = -H^{-1}(G'Py_s^* + f)$

Scaling example

- AFTI-16 linearized model (4th order, unstable)
  - 2 inputs (elevator & flap angles)
  - 2 outputs (attack & pitch angles)

- MPC problem:
  - prediction horizon=10, control horizon=2 (-> 4 free optimization vars)
  - feasibility threshold = optimality threshold = 0.01

only input constraints

AFTI	Max		Avg	
	NS	S	NS	S
GPAD	287	153	28	21
ADMM	1458	456	148	111
PQP	2396	2119	152	138
DRSA	441	278	57	32
FBN	6	7	3	2

input + output constraints

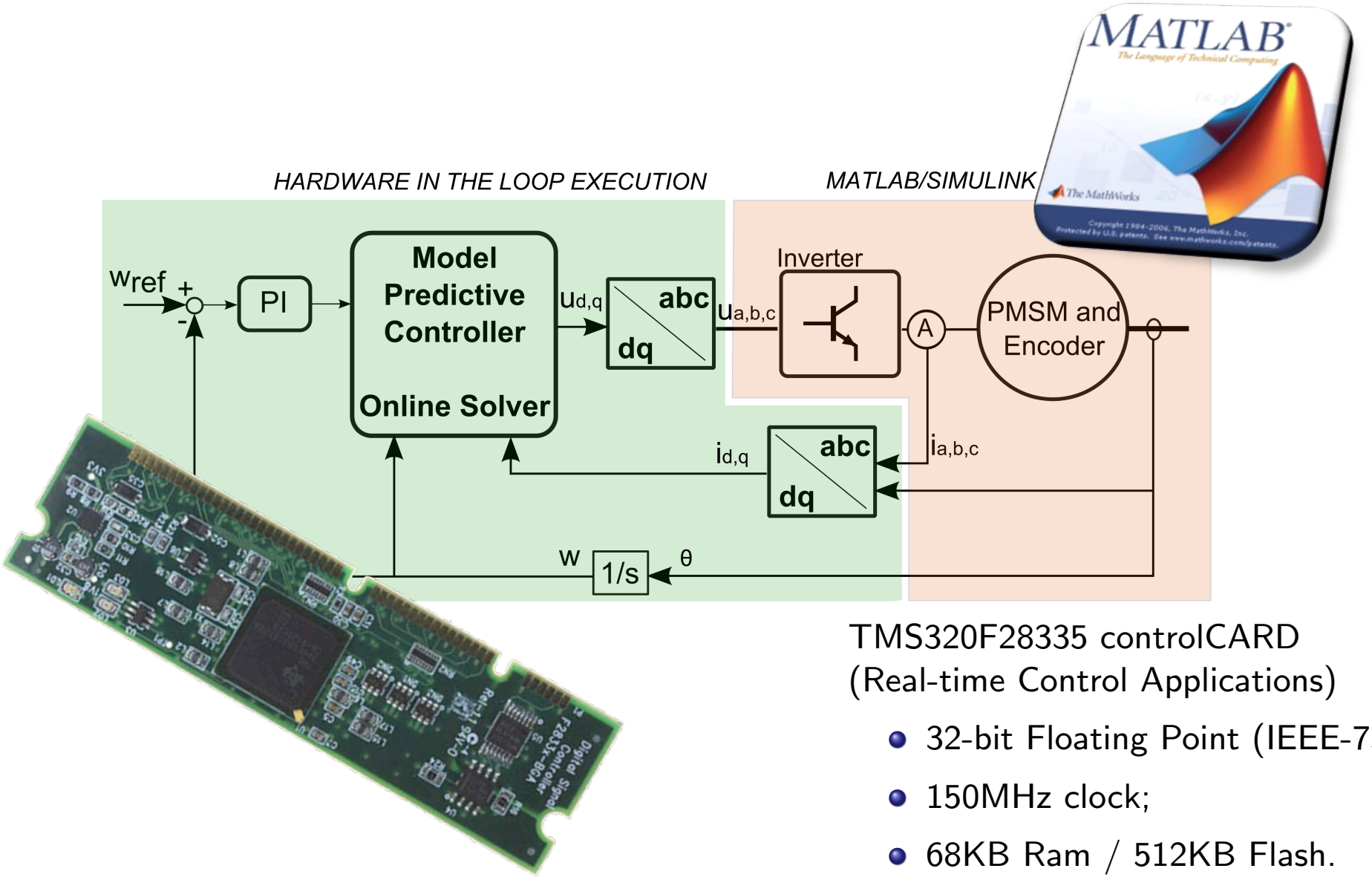
AFTI2	Max		Avg	
	NS	S	NS	S
GPAD	1605	498	668	92
ADMM	3518	1756	741	208
PQP	631075	49472	16293	3581
DRSA	3142	901	473	160
FBN	118	32	7	4

number of iterations (NS= Not Scaled, S=Scaled)

PQP = Projection-free parallel QP (Di Cairano, Brand, Bortoff, 2013)
DRSA = Accelerated Douglas-Rachford Splitting (Patrinos, Stella, Bemporad, 2014)

Hardware-In-the-Loop experiments (DSP)

- Control of a brushless DC motor



TMS320F28335 controlCARD
(Real-time Control Applications)

- 32-bit Floating Point (IEEE-754);
- 150MHz clock;
- 68KB Ram / 512KB Flash.

Hardware-In-the-Loop experiments (DSP)

- Linear MPC setup:

$$\begin{cases} x(k+1) &= Ax(k) + B_u u(k) + B_v(k) \\ y(k) &= Cx(k) \end{cases}$$

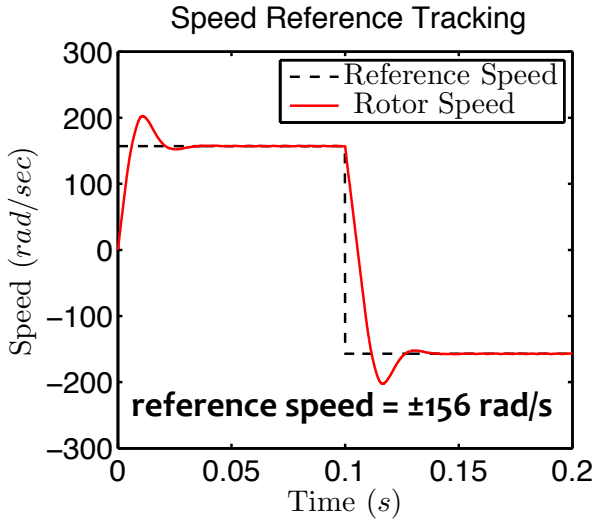
$$x = \begin{bmatrix} i_d \\ i_q \end{bmatrix} \qquad u = \begin{bmatrix} u_d \\ u_q \end{bmatrix}$$

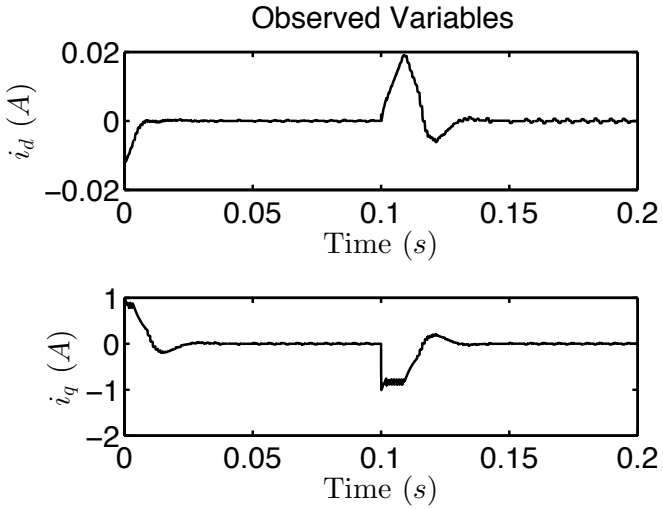
$$A = \begin{bmatrix} -\frac{R}{L} & \bar{\omega} \\ 0 & -\frac{R}{L} \end{bmatrix}, \; B_u = \begin{bmatrix} \frac{1}{L} & 0 \\ 0 & \frac{1}{L} \end{bmatrix}, \; B_v = \begin{bmatrix} 0 \\ -\omega^2 \frac{\lambda}{L} \end{bmatrix}, \; C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

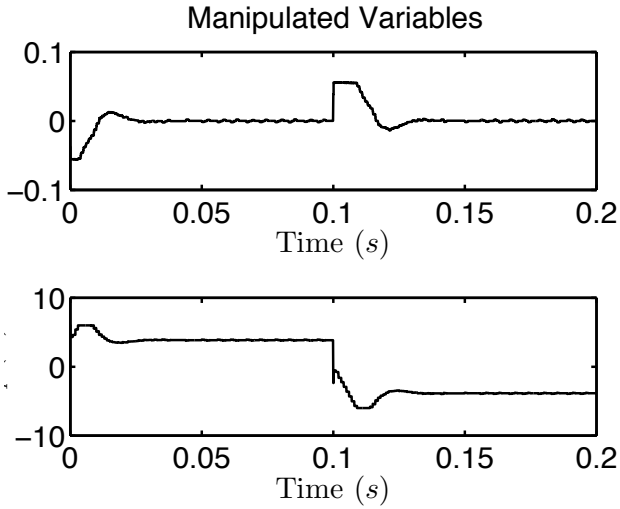
R = stator resistance [Ω]
L = stator inductance [H]
ω = rotor speed [rad/s]
λ = motor flux leakage [Wb]

prediction horizon = 4
control horizon = 2
box constraints on i_q and u_q

$$-\epsilon I_{\max} \le i_d \le \epsilon I_{\max}$$
$$V_{\max} \le u_q \le V_{\max}$$

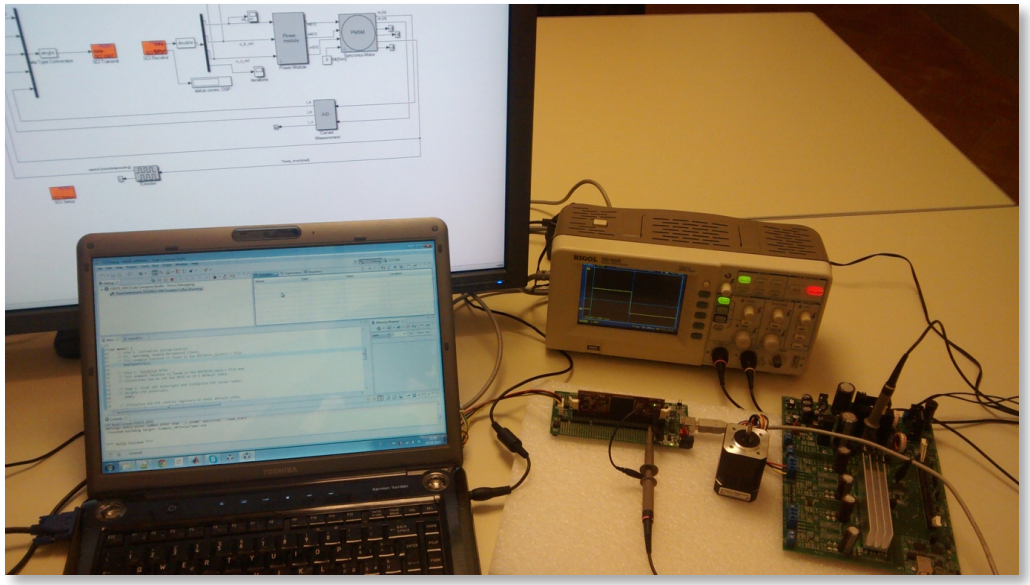






Hardware-In-the-Loop experiments (DSP)

- Experimental setup:
  - PC with MATLAB/Simulink
  - RS232 adapter
  - TMS320F28335 Experimenter Kit



var × constr.	GPAD	AS	ADMM	FBN
4 × 16	332 μs (18)	120 μs (3)	1.42 ms (62)	208 μs (2)
8 × 24	1.1 ms (22)	446 μs (5)	4 ms (77)	396 μs (2)
12 × 32	2.59 ms (27)	1.19 ms (7)	8.25 ms (82)	652 μs (2)

- Active set dominates on small problems (Goldfarb, Idnani, 1983)
- Forward-Backwards Newton (proximal method) better for larger problems

Conclusions

Which method to prefer for embedded MPC ?

	Explicit MPC	Implicit (=on-line QP)					
		active set	interior point	gradient projection	proximal Newton	ADMM	PQP
speed (small probs)	🙂	🙂	😐	😐	🙂	😐	😐
speed (large probs)	☹	☹	🙂	😐	🙂	😐	☹
worst-case estimates	🙂	☹	☹	😐	😐	☹	☹
numerical arithmetics	🙂	😐	😐	🙂	🙂	🙂	🙂
off-line computations	☹	🙂	🙂	🙂	🙂	🙂	🙂
solution quality (feas/opt)	🙂	🙂	😐	😐	😐	😐	😐
data memory	☹	🙂	🙂	🙂	🙂	🙂	🙂
control code	🙂	😐	😐	🙂	😐	🙂	🙂
best for problem size	small	medium	large	medium	medium	medium	small

very rough classification: →

```
small  ≤   6 vars,   15 constraints,    8 states/references
medium ≃  20 vars,   80 constraints,   50 states/references
large  ≥ 500 vars, 3000 constraints, 2000 states/references
```

Conclusions

- Embedded QP solvers for MPC must be **very simple** to code, **fast,** amenable for **low-precision** arithmetic, and with proved bounds on **real-time execution**

- A **"best QP algorithm" does not exist,** must have a library of methods, depending on memory/throughput/precision reqs.

- Research on QP solution methods started ~60 years ago …

(Beale, 1955)

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.

More research on QP is still needed to have an impact in real applications !