Multi-Parametric Toolbox (MPT)

Michal Kvasnica, Pascal Grieder, Mato Baotic, Miroslav Baric, Frank J. Christophersen, Manfred Morari

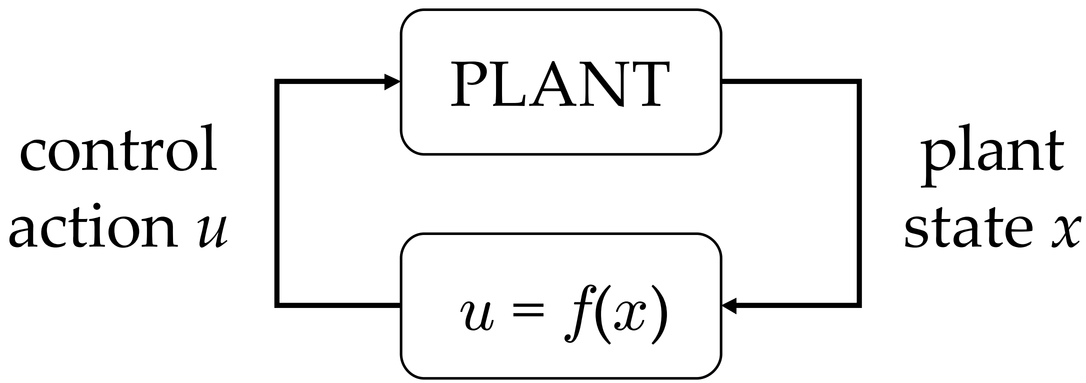
Automatic Control Laboratory, ETH Zürich

control.ee.ethz.ch

Agenda

- MPT from users' perspective
- Overview of basic concepts and functionality
- New features in MPT 2.6

Typical Problems in Control Theory



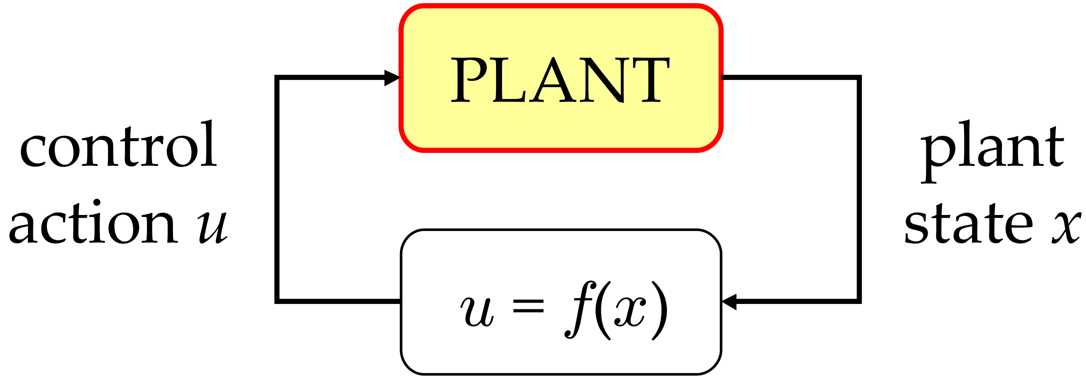
Modeling

Control

Analysis

Deployment

Typical Problems in Control Theory



Modeling

Control

Analysis

Deployment

mpt_sys and the System Structure

```
sysStruct = mpt_sys(source)
```

Possible sources:

- Control toolbox objects
- System identification toolbox objects
- MPC toolbox objects
- HYSDEL source file

Example – Conversion from HYSDEL

```
>> sysStruct = mpt_sys('two_tanks.hys')

Conversion from HYSDEL to PWA form finished (0.45 sec)

sysStruct =

         A: {[2x2 double]  [2x2 double]  [2x2 double]  [2x2 double]}
         B: {[2x2 double]  [2x2 double]  [2x2 double]  [2x2 double]}
         C: {[0 1]  [0 1]  [0 1]  [0 1]}
         D: {[0 0]  [0 0]  [0 0]  [0 0]}
      umax: [2x1 double]
      umin: [2x1 double]
      xmin: [2x1 double]
      xmax: [2x1 double]
```

- Automatically creates a PWA model (can be disabled)
- Extracts constraints from the model

Example – Conversion from the Control Toolbox

```
>> M = ss(A, B, C, D)
>> sysStruct = mpt_sys(M, Ts)

sysStruct =

     A: [2x2 double]
     B: [2x1 double]
     C: [2x2 double]
     D: [2x1 double]
    Ts: 1
```

- Performs discretization if needed
- Constraints have to be provided by the user

Possible Constraints

– Constraints on manipulated variables:

```
sysStruct.umin = umin
sysStruct.umax = umax
```

– Constraints on slew rate of manipulated variables:

```
sysStruct.dumin = dumin
sysStruct.dumax = dumax
```

– Constraints on system states:

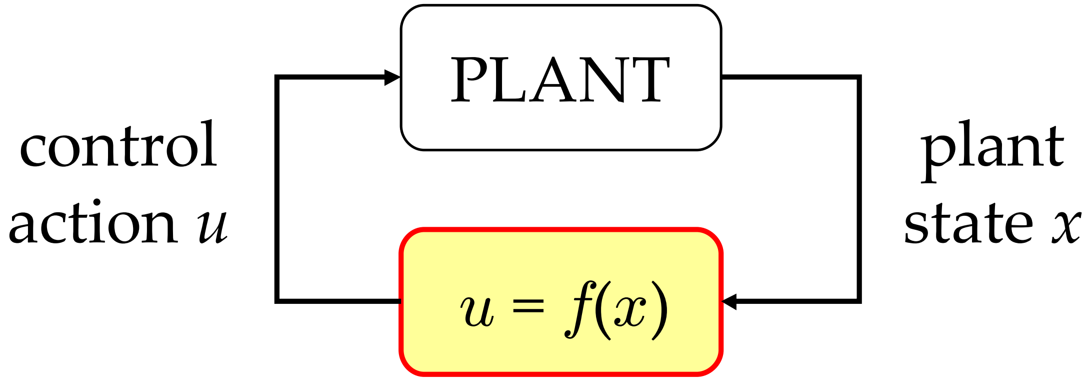
```
sysStruct.xmin = xmin
sysStruct.xmax = xmax
```

– Constraints on system inputs:

```
sysStruct.ymin = ymin
sysStruct.ymax = ymax
```

```
help mpt_sysStruct
```

Typical Problems in Control Theory



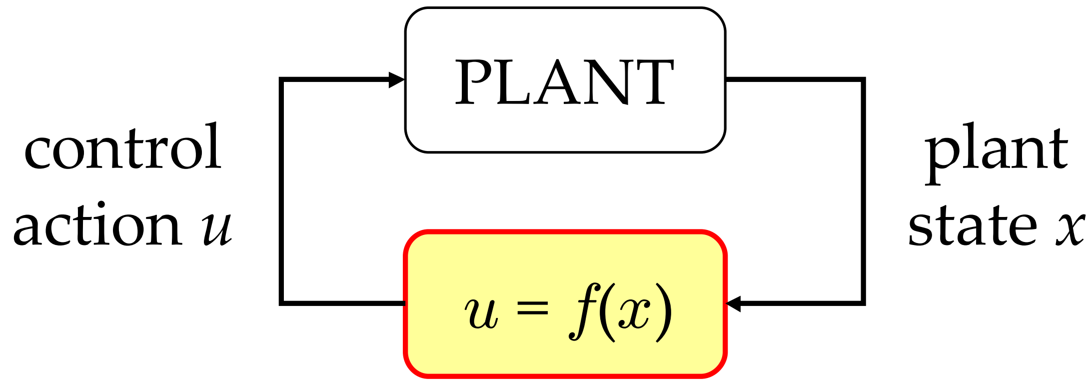
Modeling

Control

Analysis

Deployment

Synthesis: Optimal Control



Given a performance index $J_N = \sum_{k=0}^{N-1} \|Qx_k\|_p + \|Ru_k\|_p$

> Obtain optimal feedback law $u^\star = f(x)$

$$
\begin{aligned}
J_N^* \quad &= \quad \min_{u_0,\ldots,u_{N-1}} J_N, \\
\text{subj. to} \quad & \quad \text{System model} \\
& \quad \text{Constraints}
\end{aligned}
$$

Problem Structure probStruct

$$J_N = \sum_{k=0}^{N-1} \|Q(x_k - x_{ref})\|_p + \|R(u_k - u_{ref})\|_p$$

- Prediction horizon

```
probStruct.N = {N | Inf}
```

- Type of objective function

```
probStruct.norm = {1 | Inf | 2}
```

- Reference signals

```
probStruct.{xref|uref|yref|dref|zref}
```

- Penalties

```
probStruct.{Q | R | Qy | Qd | Qz | Rdu}
```

```
help mpt_probStruct
```

mpt_control

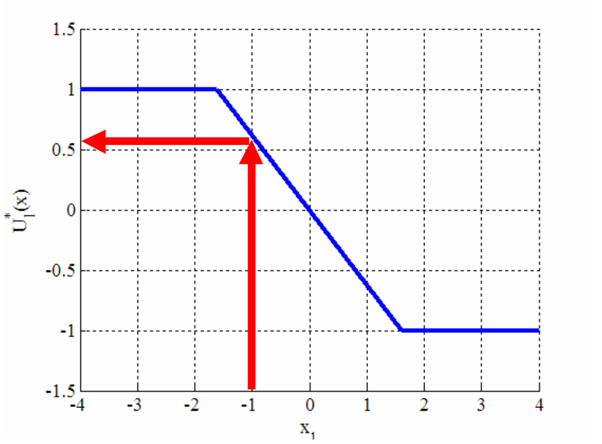
```
ctrl = mpt_control(sysStruct, probStruct, flag)
```

- Explicit controllers:
  ```
  ctrl=mpt_control(sysStruct,probStruct)
  ```

- On-line controllers:
  ```
  ctrl=mpt_control(sysStruct,probStruct,‘online’)
  ```

Controllers are Functions

To obtain the optimizer, simply evaluate the controller as a function:




```
u = ctrl(x0)
```

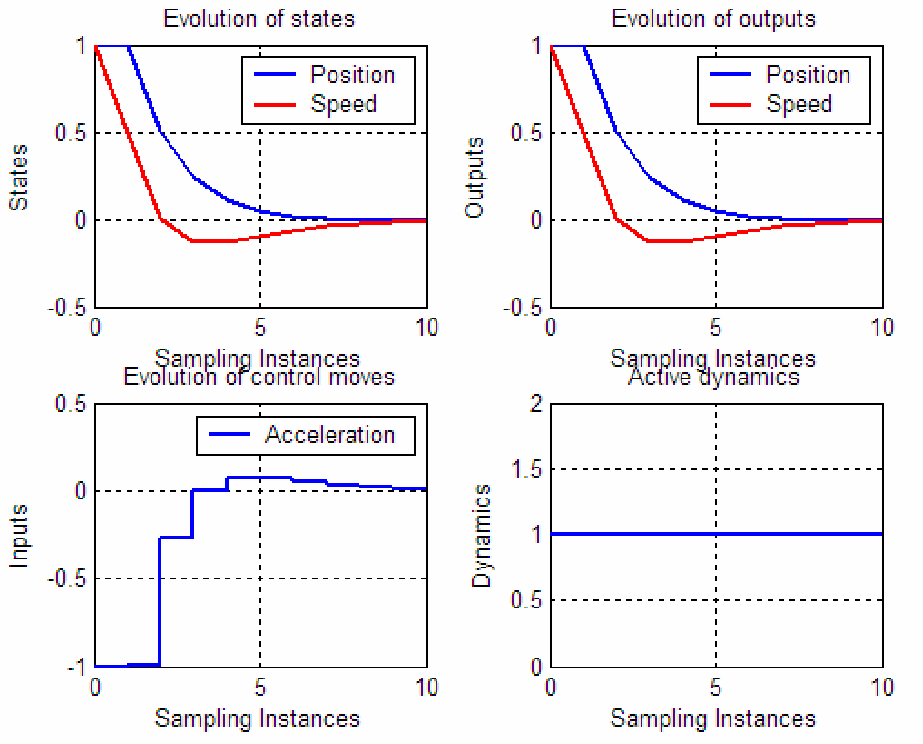
Example:

```
u = ctrl(-1)

u =

    0.6180
```

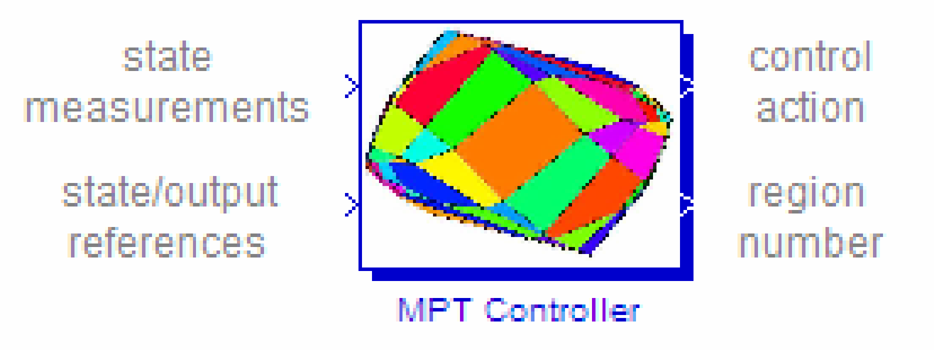
Simulations in Matlab

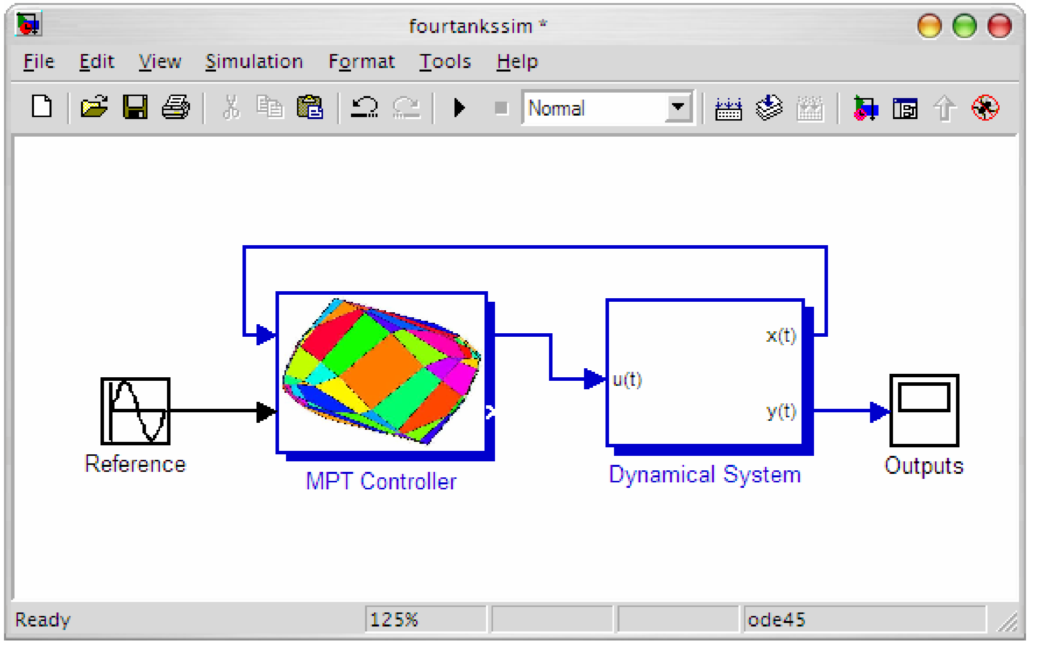
`[X,U,Y] = sim(ctrl, x0, N)`   `simplot(ctrl, x0, N)`

```
>> X=sim(ctrl, [1;1], 5)

X =

    1.0000    1.0000
    1.0000    0.5000
    0.5103    0.0051
    0.2454   -0.1299
    0.1100   -0.1326
    0.0449   -0.0989
```

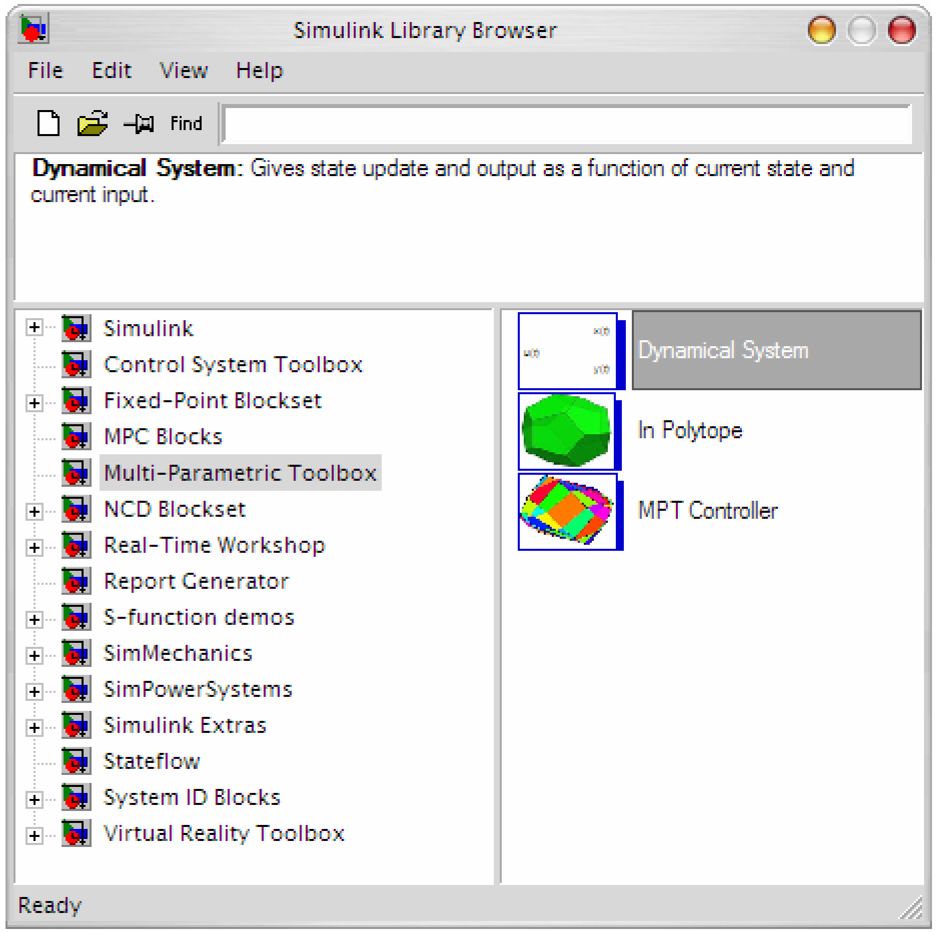


Simulations in Simulink





The Simulink Library



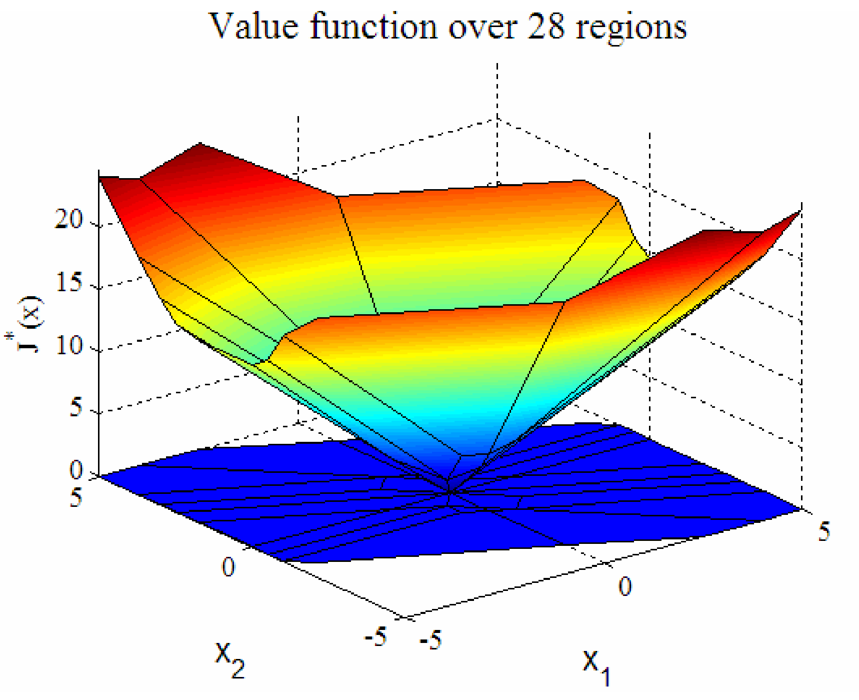
Inspection

- Direct access to internal fields, e.g.

```
Pn = ctrl.Pn
```

- Visual inspection:

```
plot(ctrl)
plotu(ctrl)
plotj(ctrl)
```

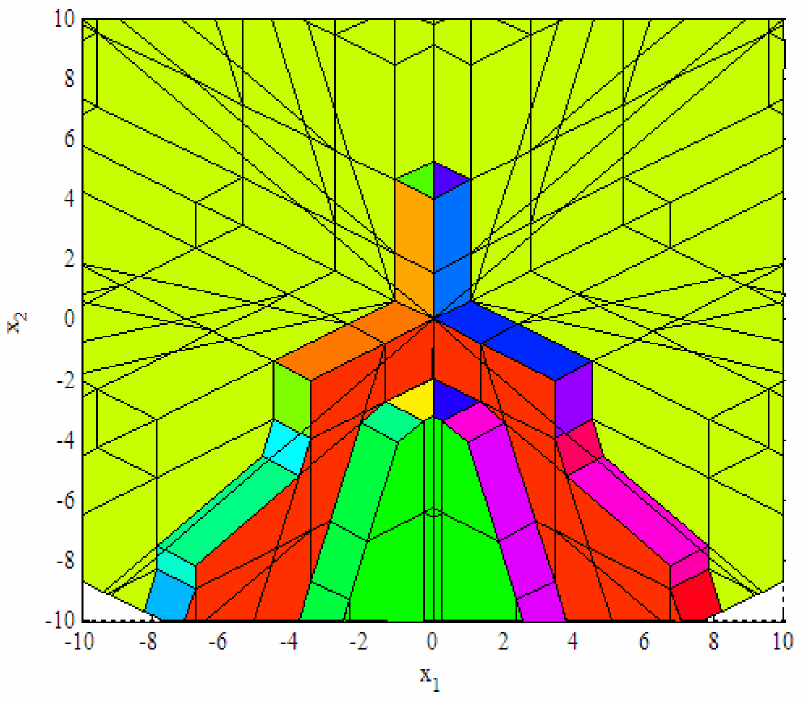


Post-processing

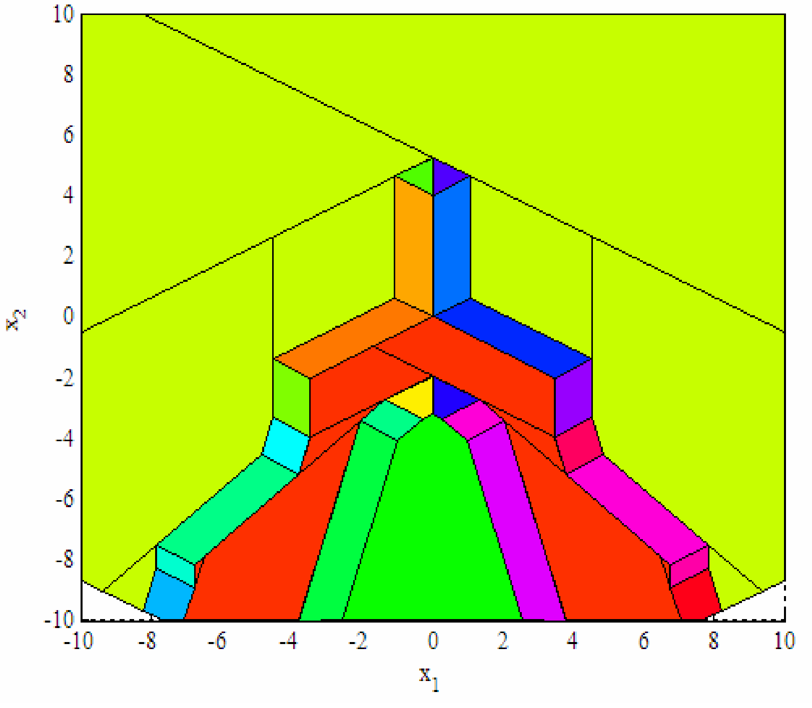
Reduce number of regions

```
ctrl = mpt_simplify(ctrl)
```

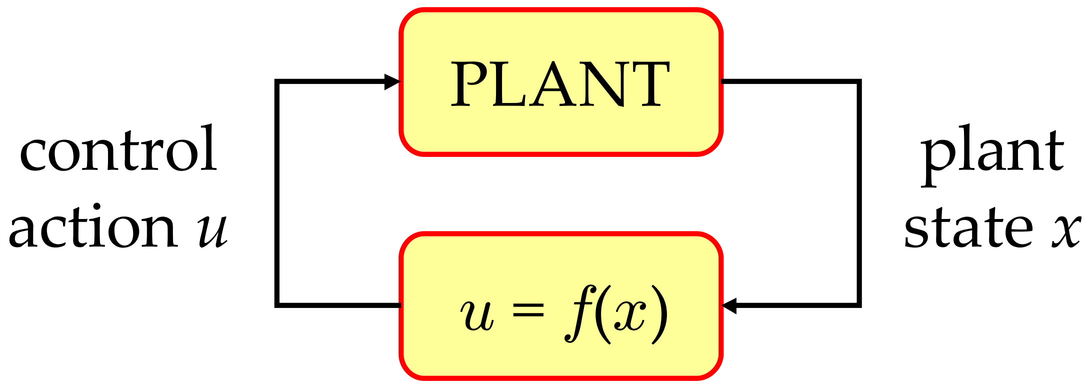
252 regions



39 regions



Typical Problems in Control Theory



Modeling

Control

Analysis

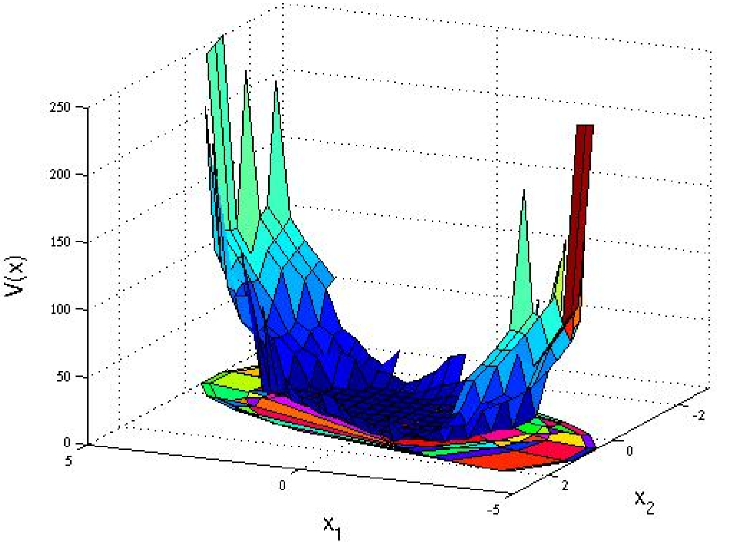
Deployment

Lyapunov Analysis

```
ctrl = mpt_lyapunov(ctrl, type)
```

Type of Lyapunov functions:

- Quadratic
- Sum of Squares
- Piecewise Affine
- Piecewise Quadratic
- Piecewise Polynomial



Reachability Analysis and Verification

- Computation of reachable sets

  `mpt_reachSets`

- Safety and liveness analysis

  `mpt_verify`

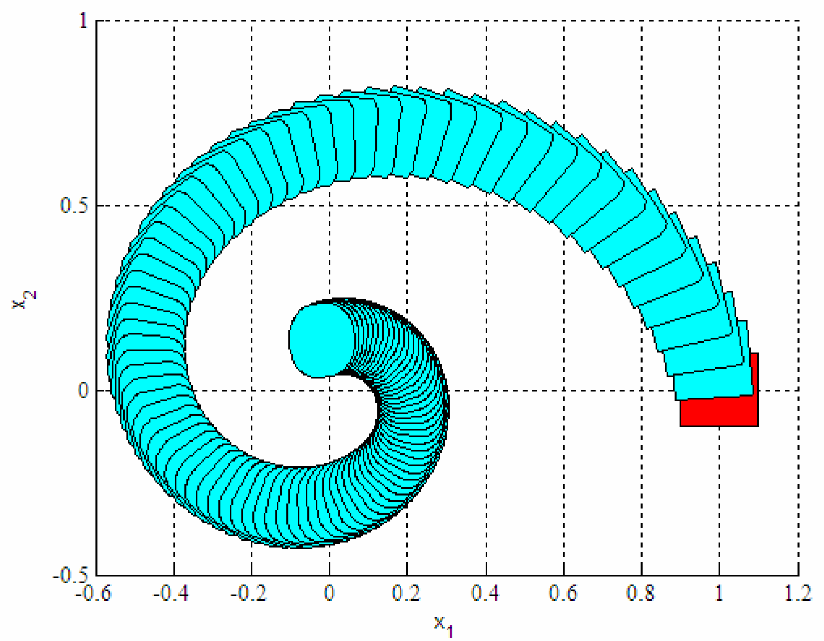
- Computation of invariant sets

  `mpt_invariantSet`

- Can analyze control laws or dynamical systems

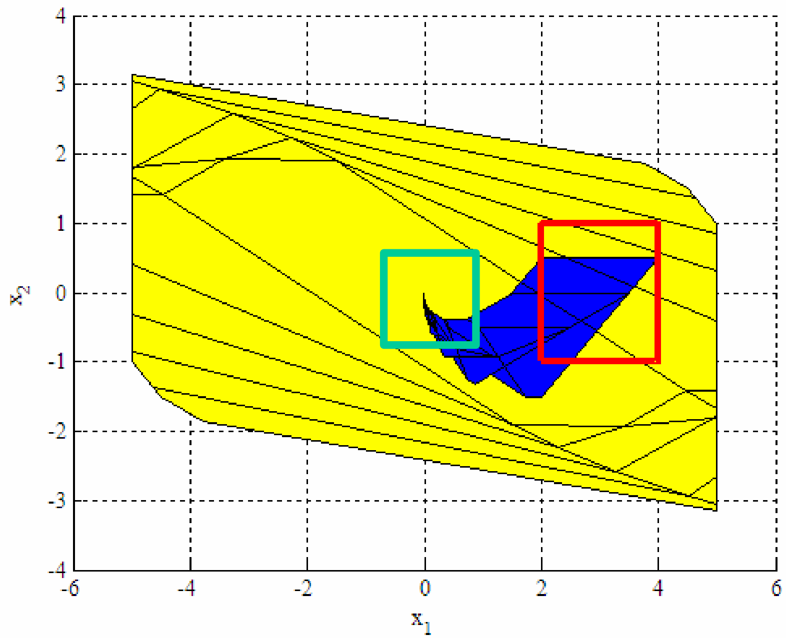
Computation of Reachable Sets – System Inputs from a Bounded Set

```
R = mpt_reachSets(sysStruct, X0, U0, N)
```

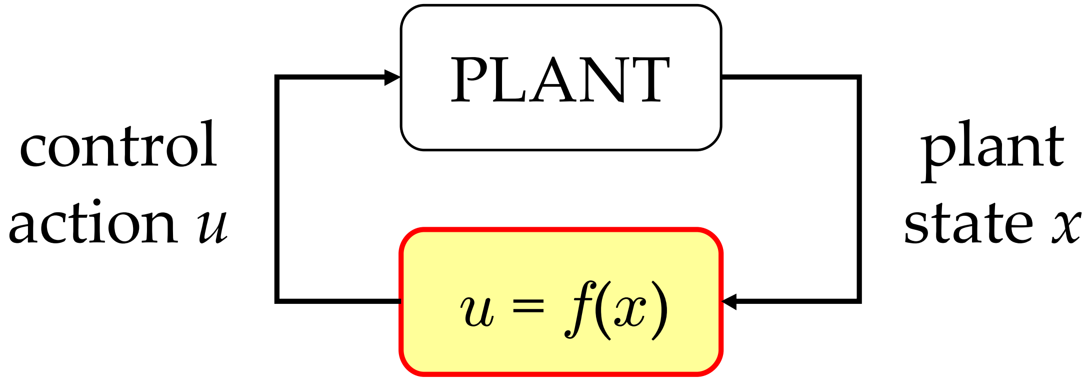


Verification –
System Inputs driven by a Controller

```
[canreach, Nf] = mpt_verify(ctrl, X0, Xf, N)
```



Typical Problems in Control Theory



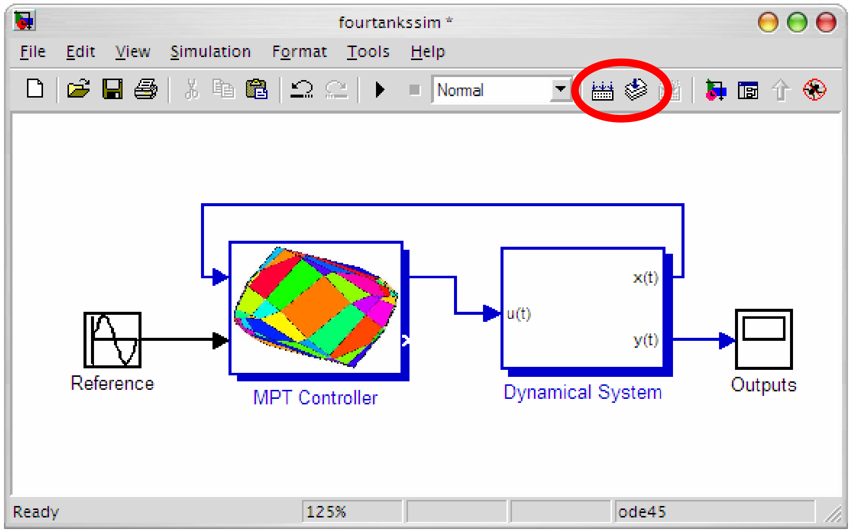
Modeling

Control

Analysis

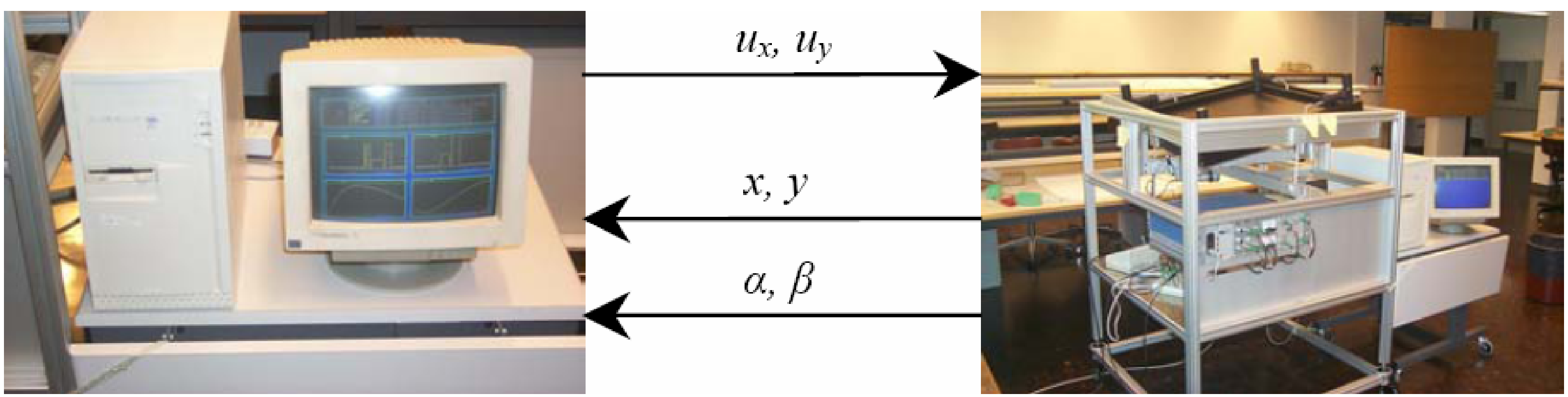
Deployment

Real Time Workshop



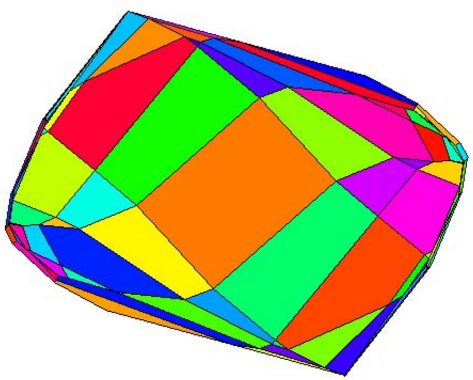
Just press a button...

RTW



Export to C code

```
mpt_exportc(ctrl, fname)
```



```c
for (iN=0; iN<STOP_TIME; iN++)
{
    printf("time: %d X = [%f %f]\n", iN, X0[0], X0[1]);

    /* tracking controllers require the state vector to be augmented.
     * if the controller is not for tracking, the function will just
     * copy X0 to X
     */
    mpt_augmentState(X, X0, Uprev, reference);

    /* obtain control action for a given state. */
    region = mpt_getInput(X, U);

    if (region < 1)
    {
        printf("No feasible control law found!\n");
        return 0;
    }
```

MPT + YALMIP = MPT 2.6

Summary of New Features

- Extended move blocking capabilities
- Control of time-varying systems
- Soft constraints
- MPC for nonlinear systems
- “Design your own MPC” function

Move Blocking

- Why: to decrease number of free control moves
- How: define a *control horizon* beyond which all control moves are assumed to be fixed
- Example: N=5, M=2

$$U = [\underbrace{u_0 \quad u_1}_{\text{Free moves}} \quad \underbrace{u_2 \quad u_3 \quad u_4}_{\text{Blocked moves}}]$$

$$u_1 = (u_2 = u_3 = u_4)$$

Move Blocking in MPT

```
probStruct.Nc = Nc
```

```
>> Double_Integrator; probStruct.N=5; probStruct.Nc=2;
>> ctrl = mpt_control(sysStruct, probStruct, 'online');
>> u = ctrl([1; 1], struct('openloop', 1))

u =

   -1.0000
   -0.4412
   -0.4412
   -0.4412
   -0.4412
```

Works for LTI, PWA, MLD

and nonlinear systems

Control of Time-Varying Systems

- Why: to allow more precise predictions
- How: use one model for each prediction
- Example:

$$
\begin{aligned}
x(1) &= A_1 x(0) + Bu(0) \\
x(2) &= A_2 x(1) + Bu(1) \\
x(3) &= A_3 x(2) + Bu(2)
\end{aligned}
$$

Where A_1, A_2, A_3 are time-varying matrices

Control of Time-Varying Systems in MPT

Define one sysStruct for each step of the prediction:

```
S1 = sysStruct; S1.A = A1;
S2 = sysStruct; S2.A = A2;
S3 = sysStruct; S3.A = A3;

model = { S1, S2, S3 };

probStruct.N = 3;

ctrl = mpt_control(model, probStruct)
```

Control of Time-Varying Systems in MPT

Anything can be time-varying, also constraints:

```
S1 = sysStruct; S1.ymax = ymax1; S1.ymin = S1.ymin1;
S2 = sysStruct; S2.ymax = ymax2; S2.ymin = S1.ymin2;
S3 = sysStruct; S3.umax = umax3; S3.umin = S3.umin3;

model = { S1, S2, S3 };

probStruct.N = 3;

ctrl = mpt_control(model, probStruct)
```

Control of Time-Varying Systems in MPT

One can also freely combine LTI/PWA/MLD models:

```
model = { MLDsysStruct, PWAsysStruct, ...
          LTIsysStruct, LTIsysStruct };

probStruct.N = length(model);

ctrl = mpt_control(model, probStruct)
```

However, dimensions must stay identical!

Soft Constraints

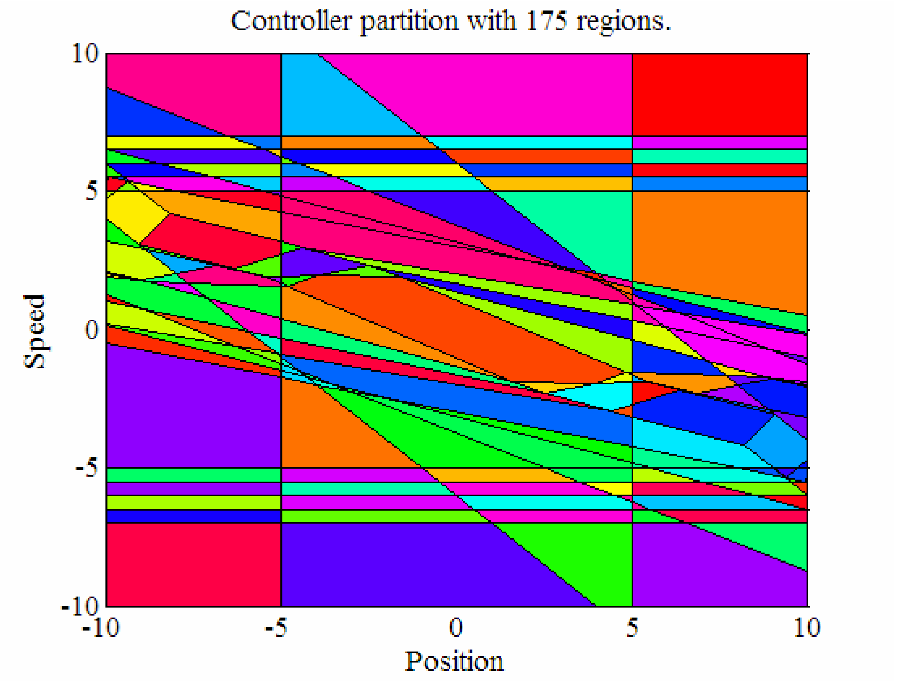
- Why: hard constraints can lead to infeasibility
- How: introduce a *slack* variable which quantifies by how much was a given constraint violated. Penalize the slack to heavily to stay within bounds if possible.
- Example:

$$x \leq x_{max} + s_x$$
$$s_x \geq 0$$

Soft Constraints in MPT

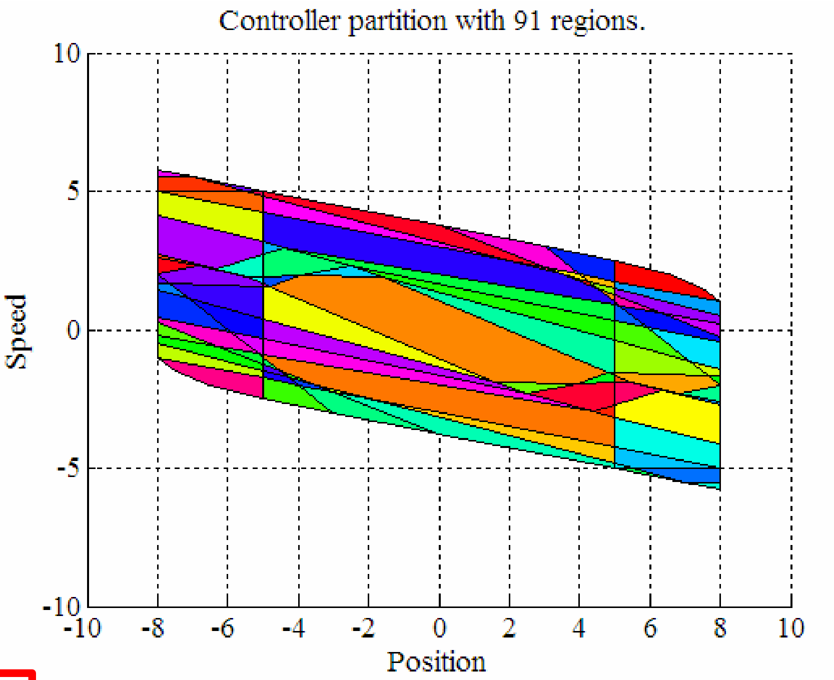
- State, input and output constraints can be softened
- To enable soft constraints, specify the corresponding penalty matrix:
  - `probStruct.Sx`
  - `probStruct.Sy`
  - `probStruct.Su`
- `probStruct.{Sx | Sy | Su}` can be combined

Soft Constraints in MPT




```
sysStruct.ymax = [5; 5]
sysStruct.ymin = [-5; -5]

probStruct.Sy = 1000

sysStruct.Pbnd = unitbox(2, 10)

ctrl = mpt_control(sysStruct, probStruct)
plot(ctrl)
```

Soft Constraints in MPT




```
sysStruct.ymax = [5; 5]
sysStruct.ymin = [-5; -5]

probStruct.Sy = 1000
probStruct.symax = [3; 3]

ctrl = mpt_control(sysStruct, probStruct)
plot(ctrl)
```

Soft Constraints in MPT

`probStruct.{sxmax | symax | sumax}` can also be used to denote only a subset of constraints to be soft, e.g.

```
probStruct.sxmax = [6.5; 0]
```

Here, constraint on the 2nd state is still hard.

MPC for Nonlinear Systems

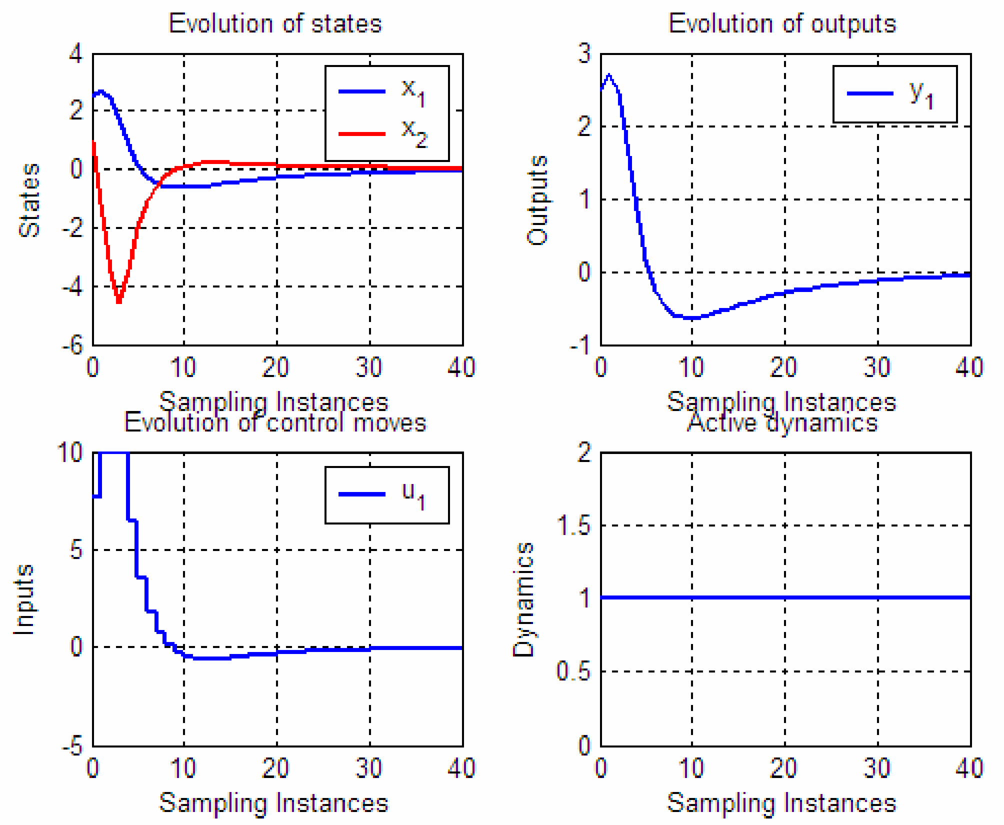
- Why: to make a basis for comparison of PWA approximations
- How: formulate (easy) and solve (extremely difficult) a nonlinear problem
- Only polynomial type of nonlinearities allowed
- Support for piecewise nonlinear systems (even harder to solve)
- Available solvers:
  - **fmincon** (Matlab)
  - **PENBMI** (commercial)
  - YALMIP's **bmibnb** (bilinear branch & bound)

MPC for Nonlinear Systems in MPT

```
sysStruct = mpt_sys(@duffing_oscillator)

probStruct.norm = 2;
probStruct.Q = [1 0; 0 1];
probStruct.R = 0.1;
probStruct.N = 3;

ctrl = mpt_control(sysStruct, probStruct, 'online')
```

MPC for Nonlinear Systems in MPT

Duffing oscillator example (*Fotiou et al., 2006*)



Computation time below 2 secs per sample

Design Your Own MPC Problem

- Why: to allow (almost) arbitrary MPC problem formulations
- How: generate a skeleton of an MPC problem and allow users to add/remove constraints and/or create a new objective function
- Goal: make the whole procedure entirely general, easy to use and fit the results into our framework

$$
\begin{aligned}
J_N^* \;=\; & \min_{u_0,\ldots,u_{N-1}} J_N, \longleftarrow \texttt{probStruct} + \text{user} \\
\text{subj. to} \quad & \left.\begin{array}{l}\text{System dynamics} \\ \text{Constraints}\end{array}\right\} \longleftarrow \texttt{sysStruct} + \text{user}
\end{aligned}
$$

3 Phases of mpt_ownmpc

1. Design phase

```
[C, O, V] = mpt_ownmpc(sysStruct, probStruct, flag)
```

2. Modification phase

   Modify the constraints "C" and/or the objective "O"

3. Computation phase

```
ctrl = mpt_ownmpc(sysStruct, probStruct, C, O, V)
```

Design Phase

- All previously mentioned features can be combined together, e.g. move blocking, time-varying systems, soft constraints, nonlinear models
- Generates a skeleton of the MPC problem based on `sysStruct/probStruct`
- Returned variables are YALMIP objects

Design Phase

```
[C,O,V] = mpt_ownmpc(sysStruct,probStruct)
>> V
       x: {[1x1 sdpvar]  [1x1 sdpvar]  [1x1 sdpvar]}
       u: {[1x1 sdpvar]  [1x1 sdpvar]}
       y: {[1x1 sdpvar]  [1x1 sdpvar]}
>> C
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
| ID|    Constraint|                         Type|                Tag|
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
| #1| Numeric value|       Element-wise 2x1|    umin < u_1 < umax|
| #2| Numeric value|       Element-wise 2x1|    ymin < y_1 < ymax|
| #3| Numeric value| Equality constraint 1x1| x_2 == A*x_1 + B*u_1|
| #4| Numeric value| Equality constraint 1x1| y_1 == C*x_1 + D*u_1|
| #5| Numeric value|       Element-wise 2x1|          x_2 in Tset|
| #6| Numeric value|       Element-wise 2x1|          x_0 in Pbnd|
| #7| Numeric value|       Element-wise 2x1|    umin < u_0 < umax|
| #8| Numeric value|       Element-wise 2x1|    ymin < y_0 < ymax|
| #9| Numeric value| Equality constraint 1x1| x_1 == A*x_0 + B*u_0|
|#10| Numeric value| Equality constraint 1x1| y_0 == C*x_0 + D*u_0|
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Polytopic Constraints on all Predicted States

- Task: add polytopic constraints $Hx_k \leq K$
- Implementation:

```
[C, O, V] = mpt_ownmpc(sysStruct, probStruct);
x = V.x;

for k = 1:length(x)
    C = C + set(H * x{k} <= K);
end

ctrl = mpt_ownmpc(sysStruct, probStruct, C, O, V);
```

Polytopic Constraints on a Subset of Predicted States

- Task: add polytopic constraints $Hx_k \leq K$ on x_0, x_2 and x_3
- Implementation:

```
for k = [1 3 4],
    % x{1} corresponds to x(0)
    % x{2} corresponds to x(1)
    % x{3} corresponds to x(2)
    % x{4} corresponds to x(3)

    C = C + set(H * x{k} <= K);
end
```

Complex Move Blocking

- Task: add complex move-blocking type of constraints:
    1. $u_0 = u_1$
    2. $(u_1 - u_2) = (u_2 - u_3)$
    3. $u_3 = K\,x_2$
- Implementation:

```
% u_0 == u_1
C = C + set( V.u{1} == V.u{2} )

% (u_1-u_2) == (u_2-u_3)
C = C + set( (V.u{2} - V.u{3} ) == ( V.u{3} - V.u{4} ) )

% u_3 == K*x_2
C = C + set( V.u{4} == K * V.x{3})
```

Using sum() and abs() Operators

- Task: bound the overall energy consumption during the whole prediction horizon
- Implementation:

```
C = C + set( sum( abs([V.u{:}] ) ) <= bound )
```

Contraction Constraints

- Task: force state x_{k+1} to be closer (in a 1-norm sense) to the origin that x_k has been
- Implementation:

```
for k = 1:length(V.x)-1
  C = C + set(norm(V.x{k+1}, 1) <= norm(V.x{k}, 1));
end
```

Logic Constraints

- Task: $x_0 \leq 0 \Rightarrow u_0 \geq 1$
- Implementation:

```
C = C + set( implies(V.x{1} <= 0, V.u{1} >= 1))
```

- However: doesn't say anything about u_0 if x_0 is > 0

Logic Constraints

- Task: $x_0 \leq 0 \iff u_0 \geq 1$
- Implementation:

```
C = C + set( iff(V.x{1} <= 0, V.u{1} >= 1))
```

Nonlinear Constraints

- Task: add a ball constraint $x_N^2 \leq 1$
- Implementation:

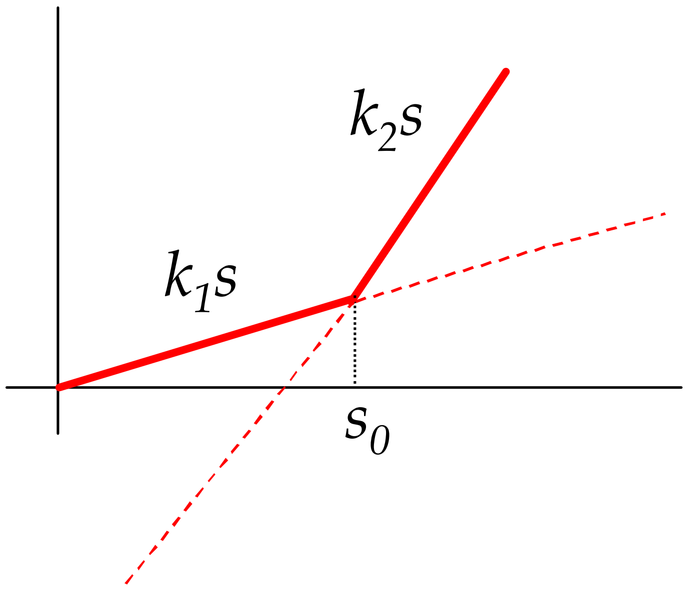
```
C = C + set( V.x{end}'*V.x{end} <= 1 )
```

- However: the problem will become nonlinear with all the negative implications

Custom Objectives

- Task: use piecewise affine penalties on slacks

```
probStruct.sxmax = Inf
probStruct.Sx = 0
[C, O, V] = mpt_ownmpc(sysStruct, probStruct)

V =

       x: {[1x1 sdpvar]   [1x1 sdpvar]   [1x1 sdpvar]}
       u: {[1x1 sdpvar]   [1x1 sdpvar]}
       y: {[1x1 sdpvar]   [1x1 sdpvar]}
      sx: {[1x1 sdpvar]   [1x1 sdpvar]   [1x1 sdpvar]}
    type: 'explicit'
```




```
for k = 1:length(V.sx),
  s = V.sx{k};
  O = O + max([k1*s, k2*s - s0*(k2 - k1)])
end
```

We can use $max(k_1 s, k_2 s)$

Custom Objectives

- Task: use time varying reference signals
- By default MPT assume the reference is fixed for all predictions steps
- But the user can change it!

```
O = norm(V.x{1} - rt1, 1) + ...
    norm(V.x{2} - rt2, 1) + ...
    norm(V.x{3} - rt3, 1) + ...
    V.u{1}'*V.u{1} + V.u{2}'*V.u{2}
```

"Unpack and Use" Toolbox

- Works on Windows, Linux, Solaris, Macs
- HYSDEL
- Hybrid Identification Toolbox
- YALMIP
- Ellipsoidal Toolbox
- ESP
- Interfaces to many solvers

http://control.ee.ethz.ch/~mpt/