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

Today's Agenda

- General overview of the toolbox
- Functional step-by-step
- Using YALMIP inside of MPT

The MPT Framework

Model → Control → Analysis → Deployment

New powerful framework for control of hybrid systems

- Set of computationally challenging problems to be solved with new techniques, e.g. computational geometry
- Need for international development effort and tool repository to give educated user access to state-of-the-art techniques
- Multi-Parametric Toolbox with ETH leadership

MPT Contributors

- CDD LP solver
- Ellipsoidal Toolbox
- Hybrid Identification Toolbox
- HYSDEL
- Projection Algorithms
- SeDuMi SDP solver
- YALMIP

Possible Model Sources for MPT

Model → Control → Analysis → Deployment

Low-level formulations

- Linear state-space representation
- Piecewise Affine (PWA) models
- Mixed Logical Dynamical (MLD) form

Possible Model Sources for MPT

Model → Control → Analysis → Deployment

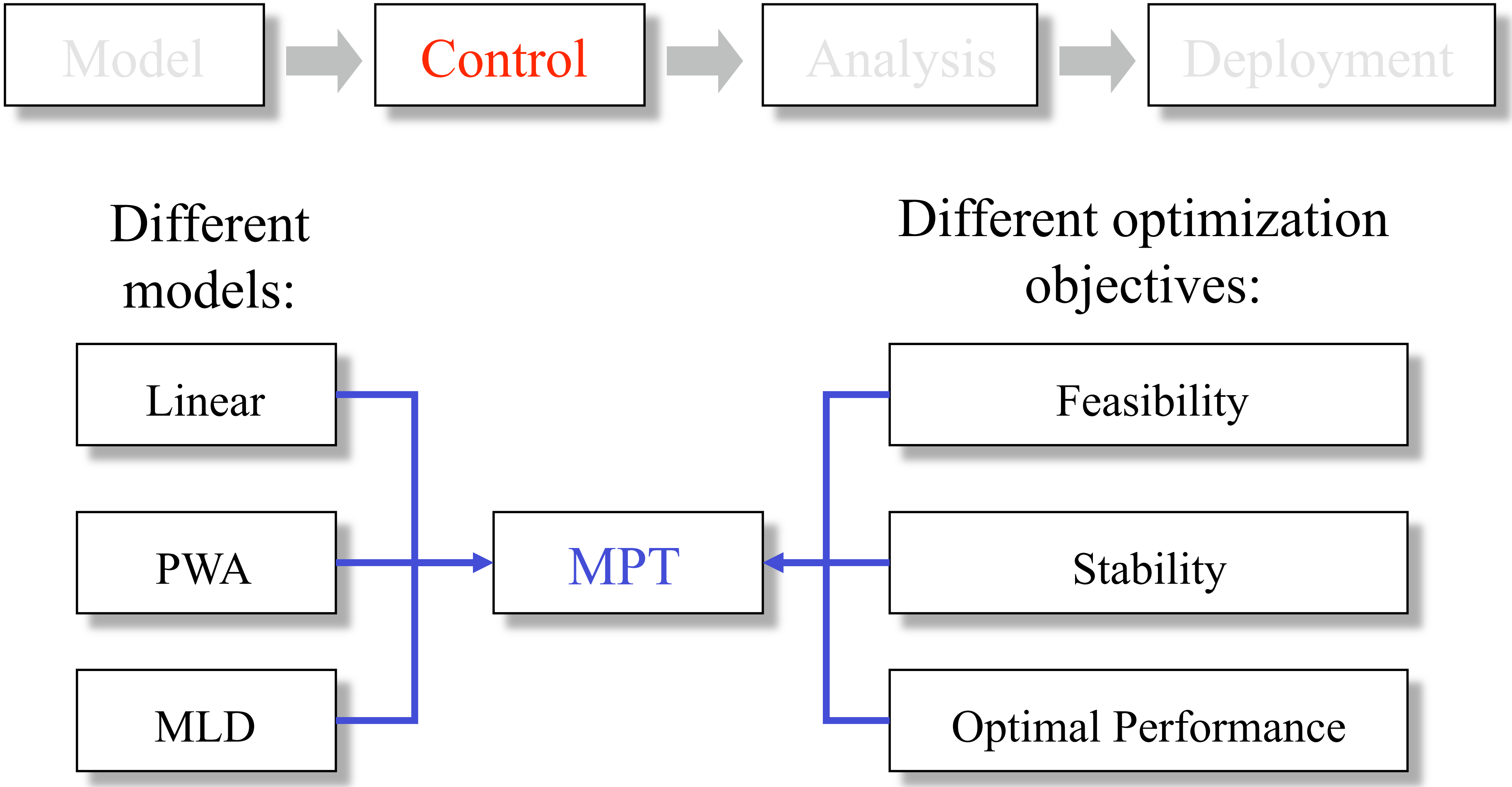
- Low-level formulations
  - Linear state-space representation
  - Piecewise Affine (PWA) models
  - Mixed Logical Dynamical (MLD) form
- High-level languages
  - HYSDEL

Possible Model Sources for MPT

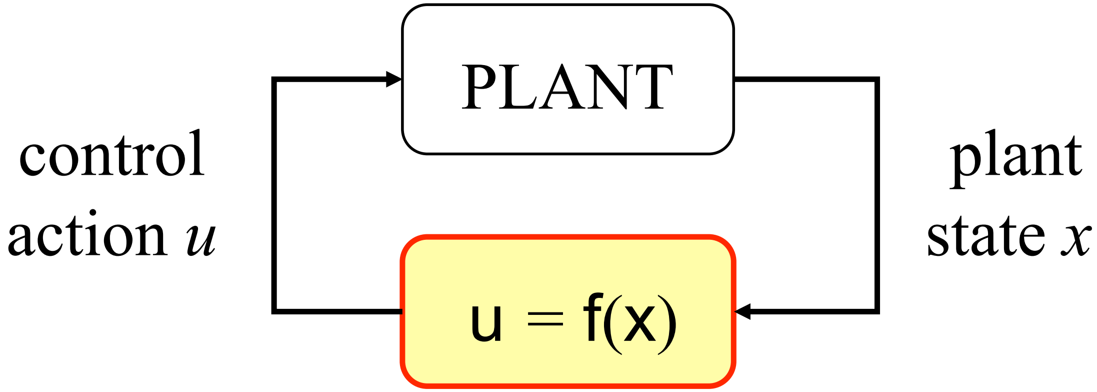
Model → Control → Analysis → Deployment

- Low-level formulations
  - Linear state-space representation
  - Piecewise Affine (PWA) models
  - Mixed Logical Dynamical (MLD) form
- High-level languages
  - HYSDEL
- Plant data
  - System identification

Control in MPT



Synthesis: Optimal Control

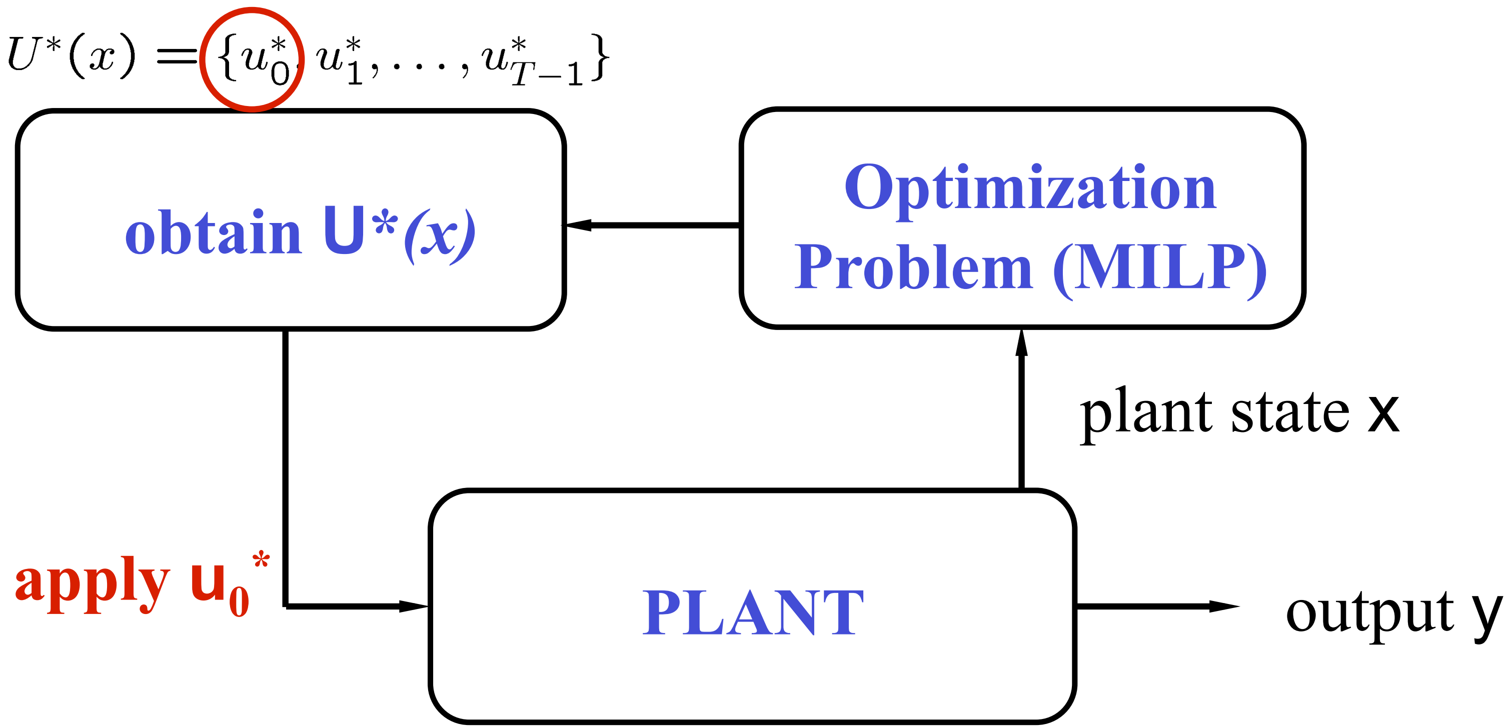


Given a performance index $$J_N = \sum_{k=0}^{N-1} \|Qx_k\|_p + \|Ru_k\|_p$$

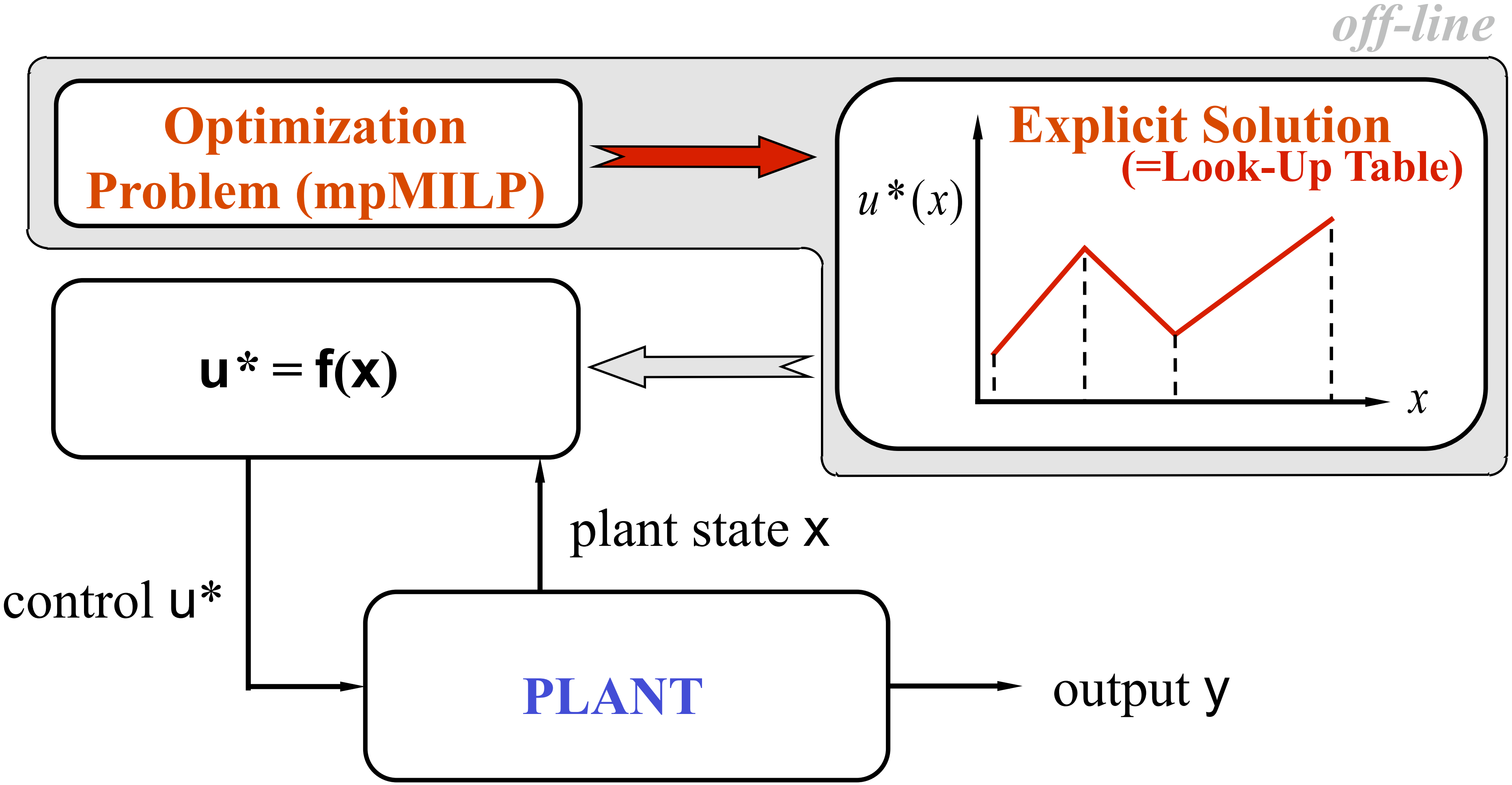
Obtain optimal feedback law $u^\star = f(x)$

$$\begin{aligned} U^*(x) \quad &= \quad \arg \min_{U=\{u_0,\dots,u_{N-1}\}} J_N, \\ \text{subj. to} \quad & \quad \text{Plant model} \\ & \quad \text{Constraints} \end{aligned}$$

Receding Horizon Control
On-Line Optimization



Receding Horizon Policy
Off-Line Optimization



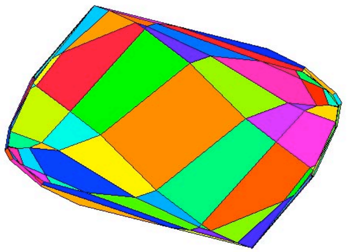
Analysis in MPT

Model → Control → **Analysis** → Deployment

- Simulations in Simulink
- Verification of safety and liveness properties
- Stability analysis via Lyapunov functions

Deployment of Explicit Control Laws – Code Generation

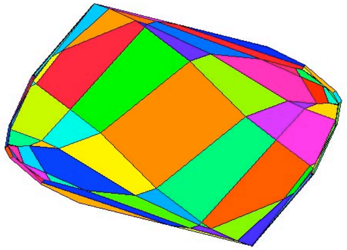
Model → Control → Analysis → **Deployment**



```
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;
    }
```

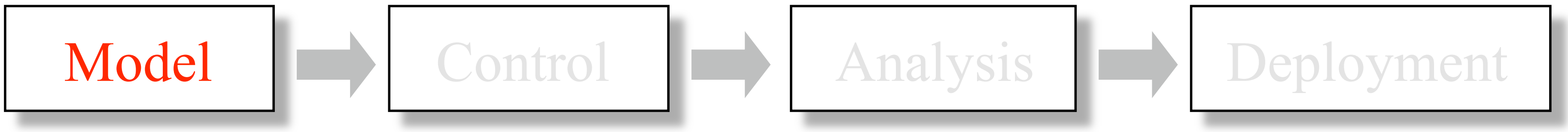
Deployment of Explicit Control Laws – Code Generation

Model → Control → Analysis → **Deployment**



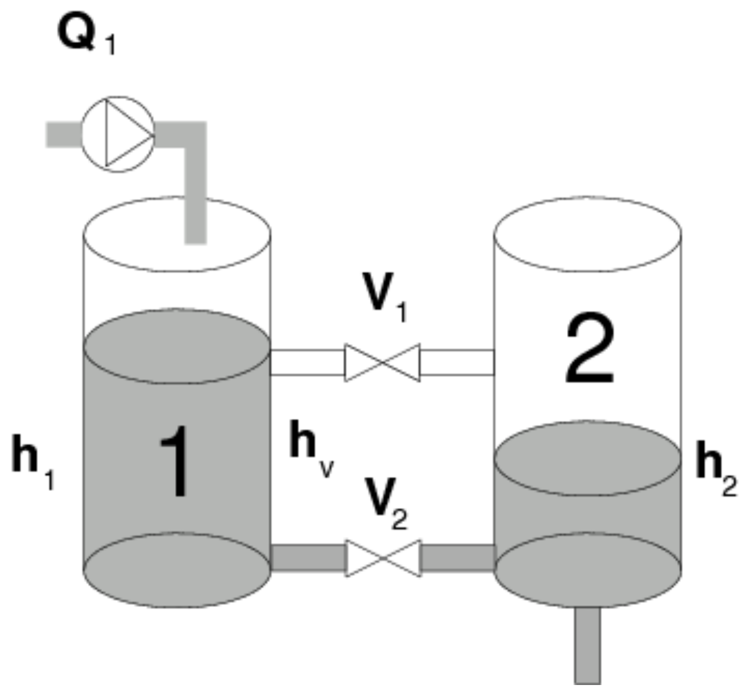
```
for (iN=0; iN<STOP_TIME; iN++)
{

    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;
    }
```

The MPT Framework



Motivating Example

- Two connected liquid tanks
- Hybrid system
- Inflow Q_1 operated continuously in range 0-1
- Valve V_2 operating discretely at values 0, 0.5, 1



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]
```

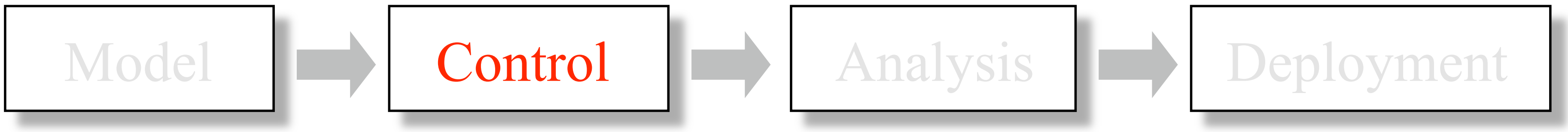
- Automatically creates a PWA model (can be disabled)
- Extracts constraints from the model

Modification of the System Structure

- Refine constraints
  - Constraints on state variables
  - Constraints on output variables
  - Constraints on manipulated variables
  - Rate constraints on manipulated variables
- Change nature of manipulated variables
  - Continuous variables
  - Boolean variables
  - Variables with values from finite alphabet

```
sysStruct.Uset{2} = [0 0.5 1]
```

The MPT Framework



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}`

Problem Formulation for the Two Tanks System

- Regulate level in 2nd tank to 0.2 m
- Use 1-norm formulation
- Prediction horizon 3

```
probStruct.N    = 3
probStruct.norm = 1
probStruct.Qy   = 100
probStruct.R    = 1e-4*eye(2)
probStruct.yref = 0.2
```

mpt_control

- Given: system and problem structures

- Compute the control law off-line:

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

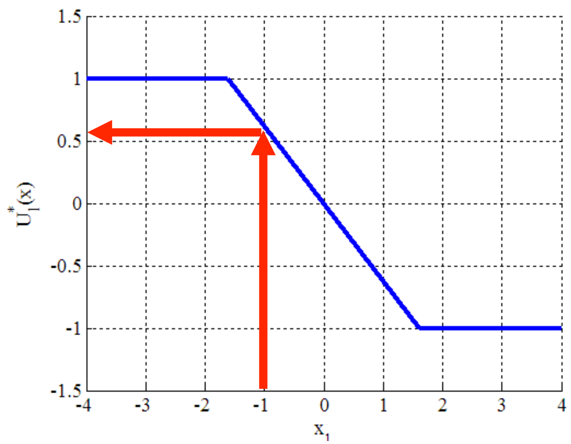
- Create an on-line controller:

```
ctrl=mpt_control(sysStruct,probStruct,‘online’)
```

Controllers are Functions

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

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



Example:

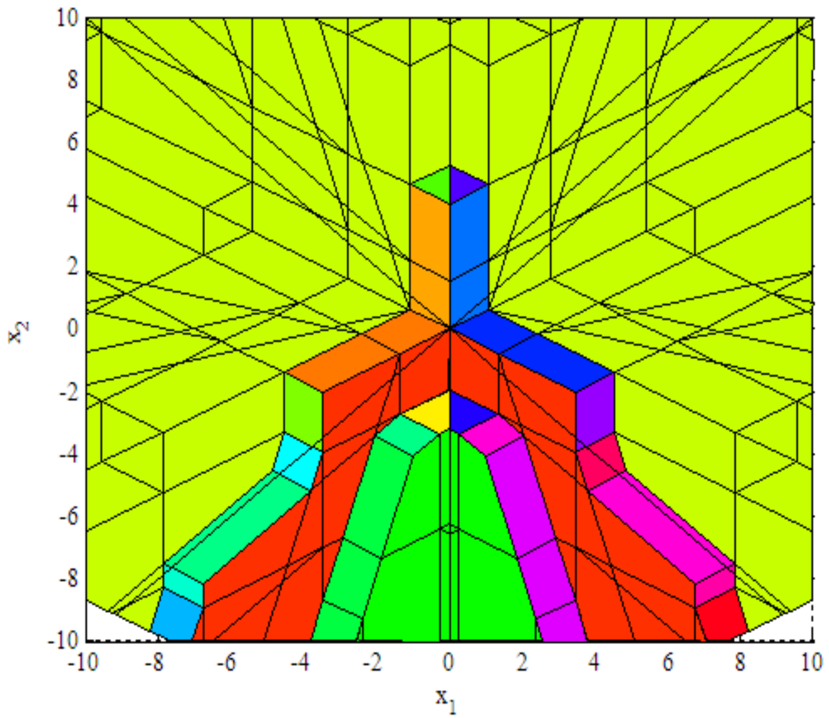
```
u = ctrl(-1)

u =

    0.6180
```

Post-processing – Complexity Reduction

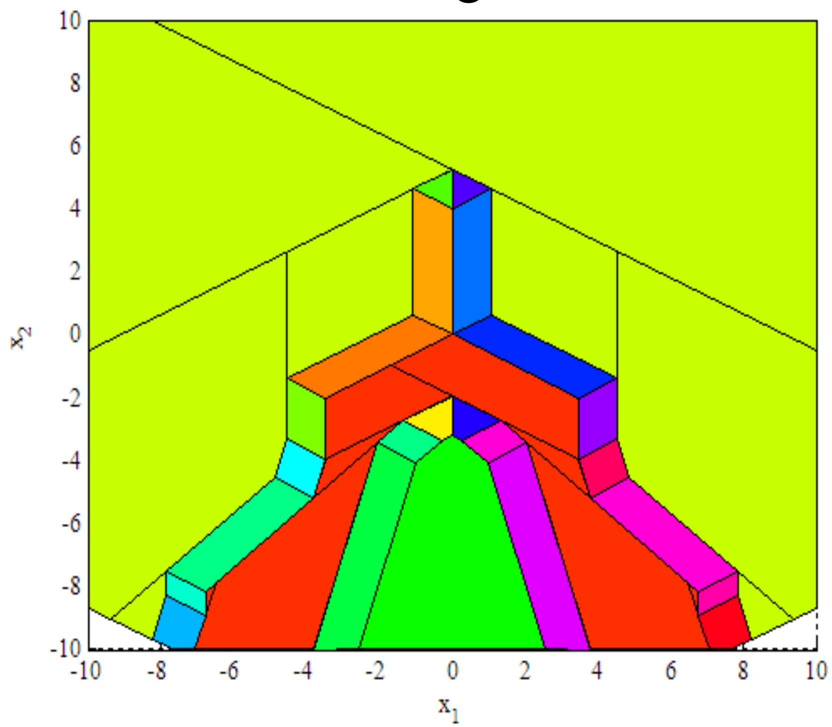
Reduce number of regions

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

252 regions



39 regions



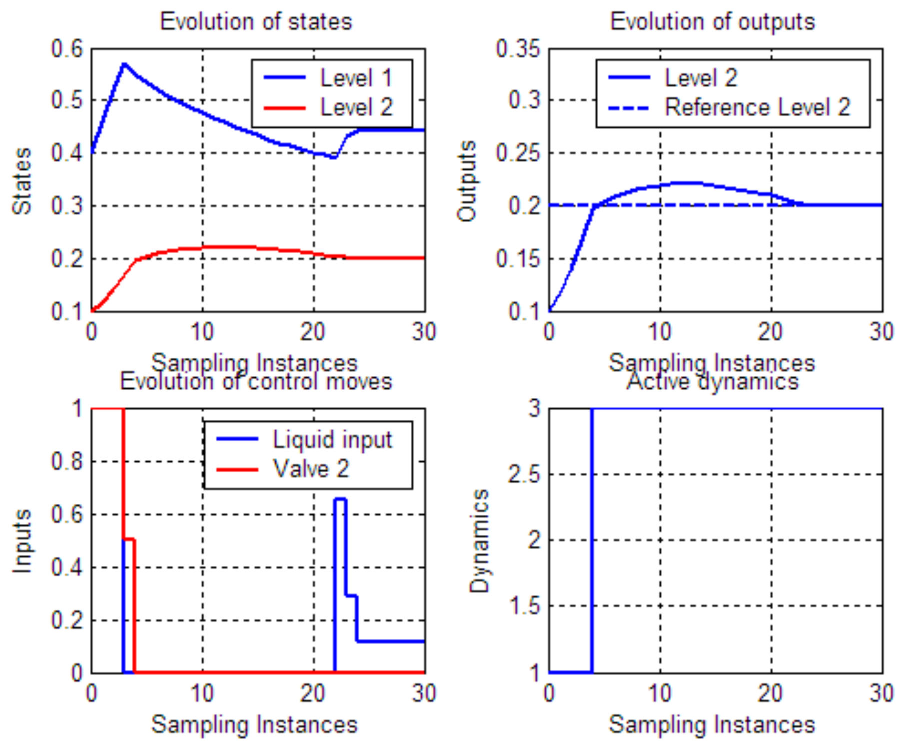
The MPT Framework

Model → Control → **Analysis** → Deployment

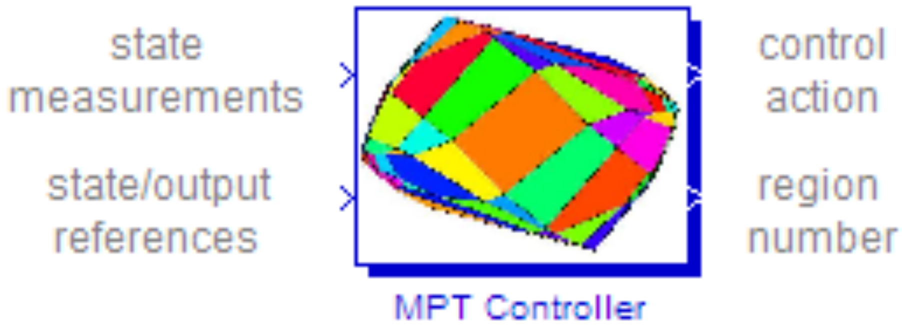
Simulations in Matlab

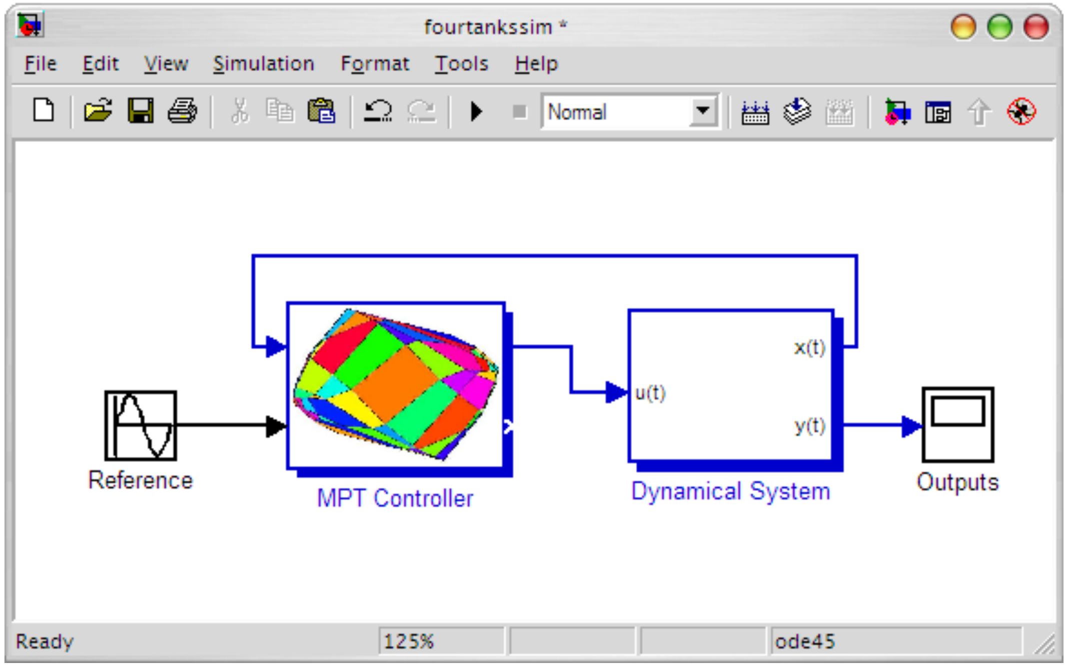
```
[X,U,Y] = sim(ctrl, x0, N)   simplot(ctrl, x0, N)
```

```
>> X=sim(ctrl, [0.4;0.1], 5)

X =

    0.4000    0.1000
    0.4605    0.1173
    0.5170    0.1396
    0.5696    0.1663
    0.5483    0.1969
    0.5347    0.2028
```

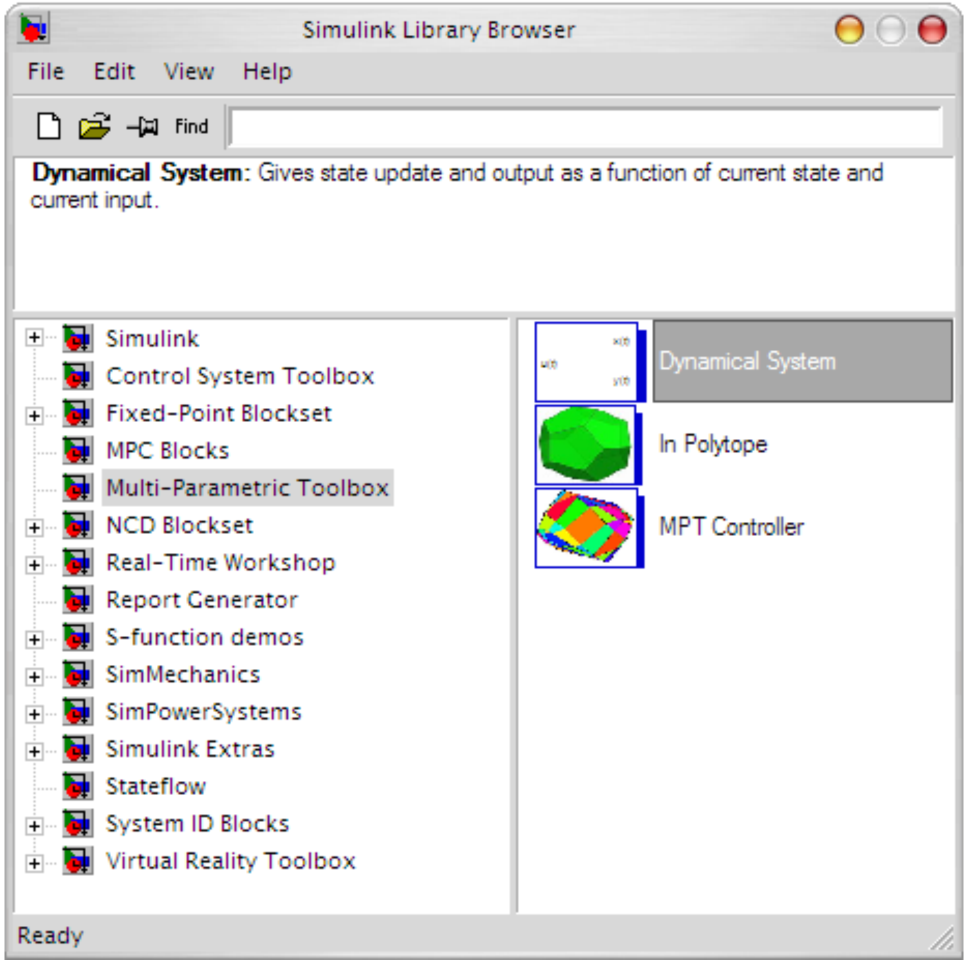


Simulations in Simulink





The Simulink Library



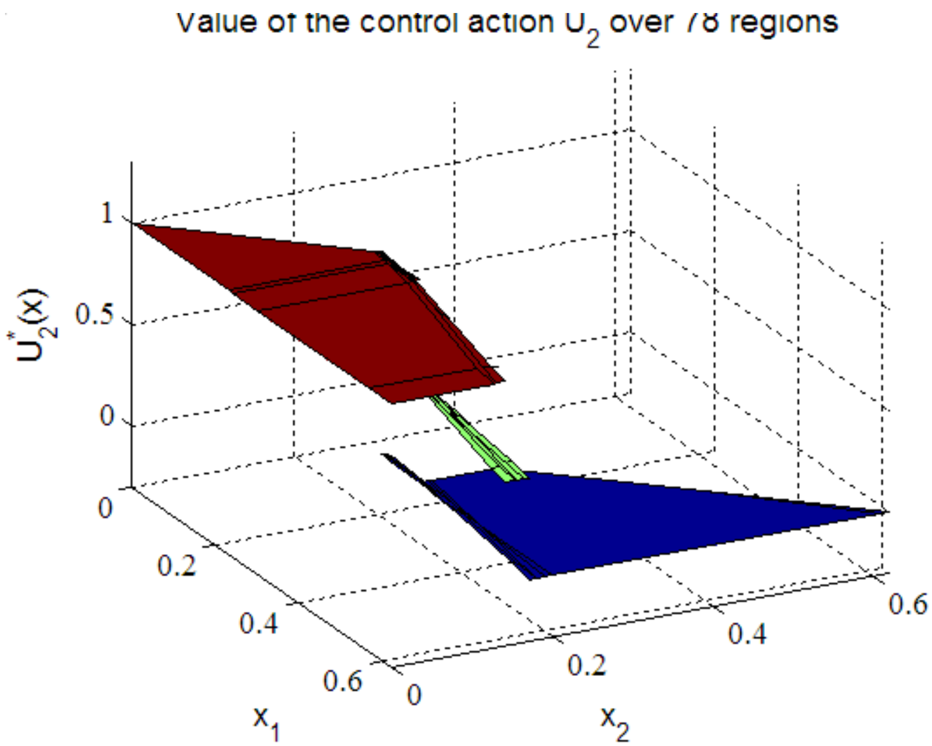
Visual Inspection

Visual inspection of controller regions:

```
plot(ctrl)
```

Visual inspection of controller actions:

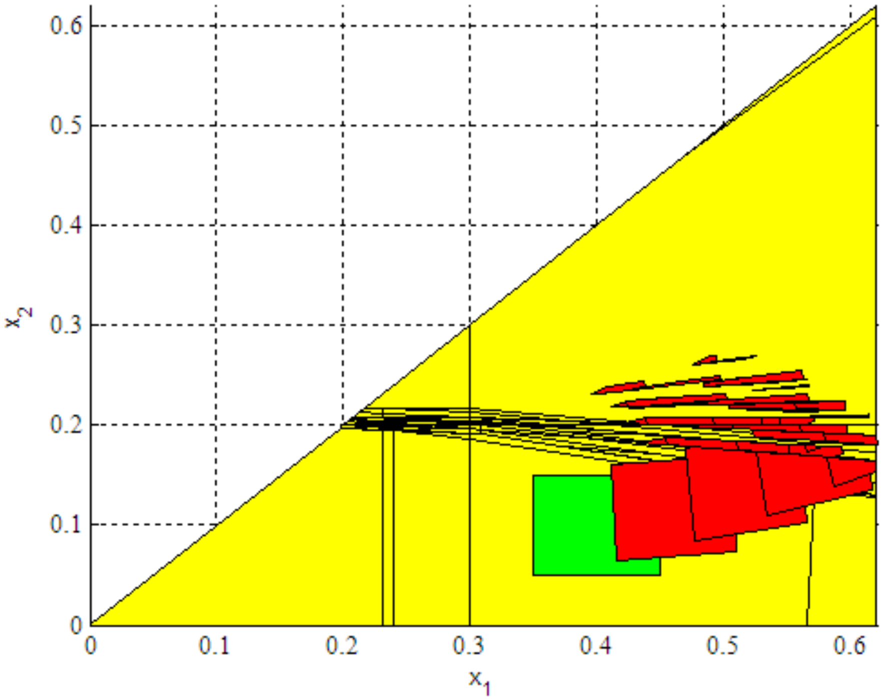
```
plotu(ctrl)
```



Reachability Analysis

Where will the controller drive system states from a given set of initial conditions?

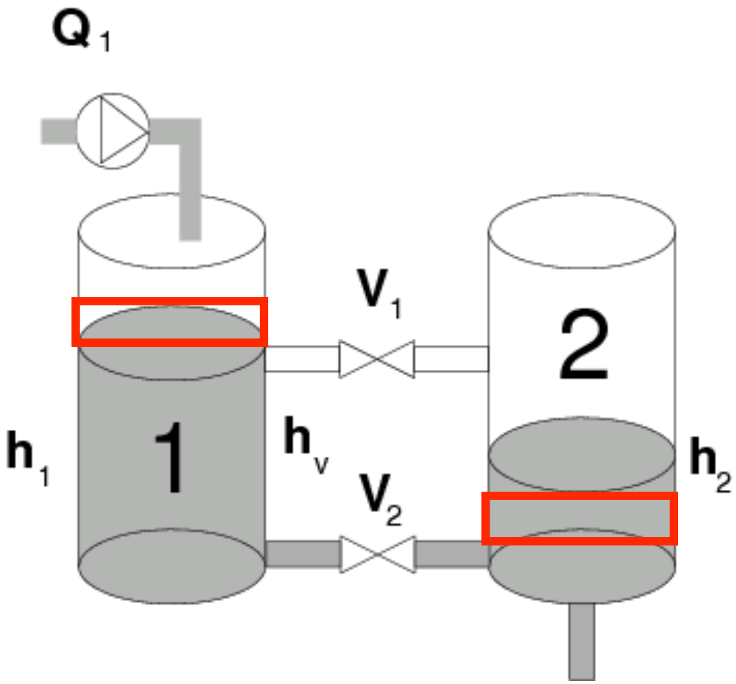
```
S=mpt_reachSets(ctrl, X0, N)
```

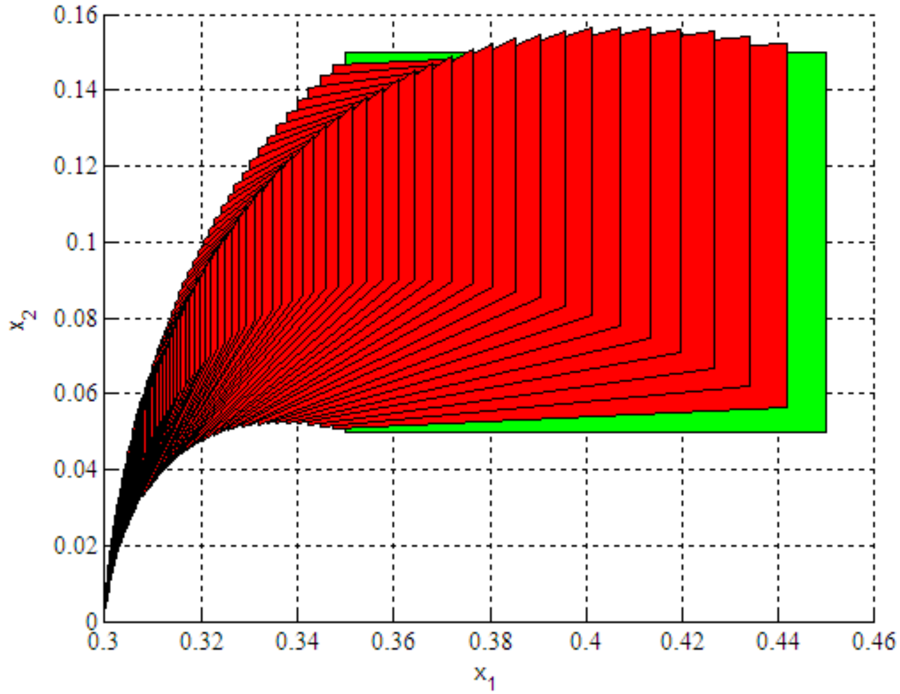


Reachability Analysis

How will the system evolve if valves are stuck?

```
Options.U = [0; 0]

S=mpt_reachSets(sysStruct, X0, N, Options)
```



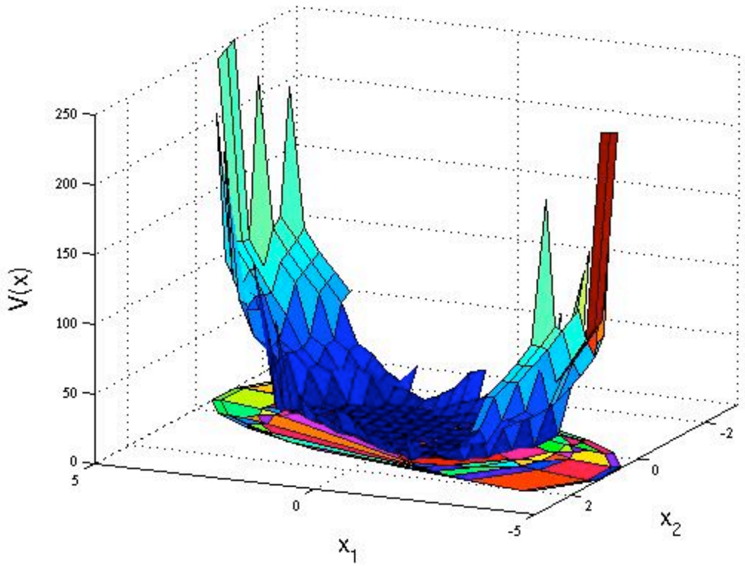


Lyapunov Analysis

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

Type of Lyapunov functions:

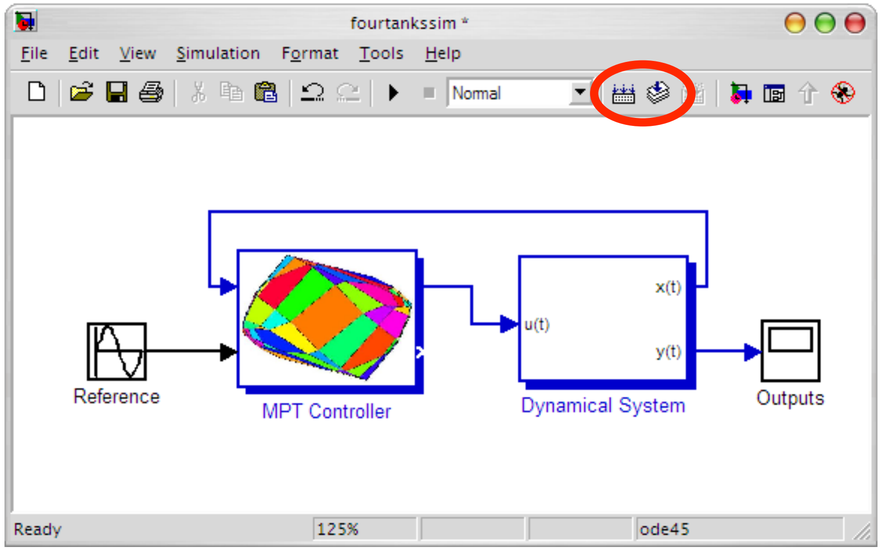
- Quadratic
- Sum of Squares
- Piecewise Affine
- Piecewise Quadratic
- Piecewise Polynomial



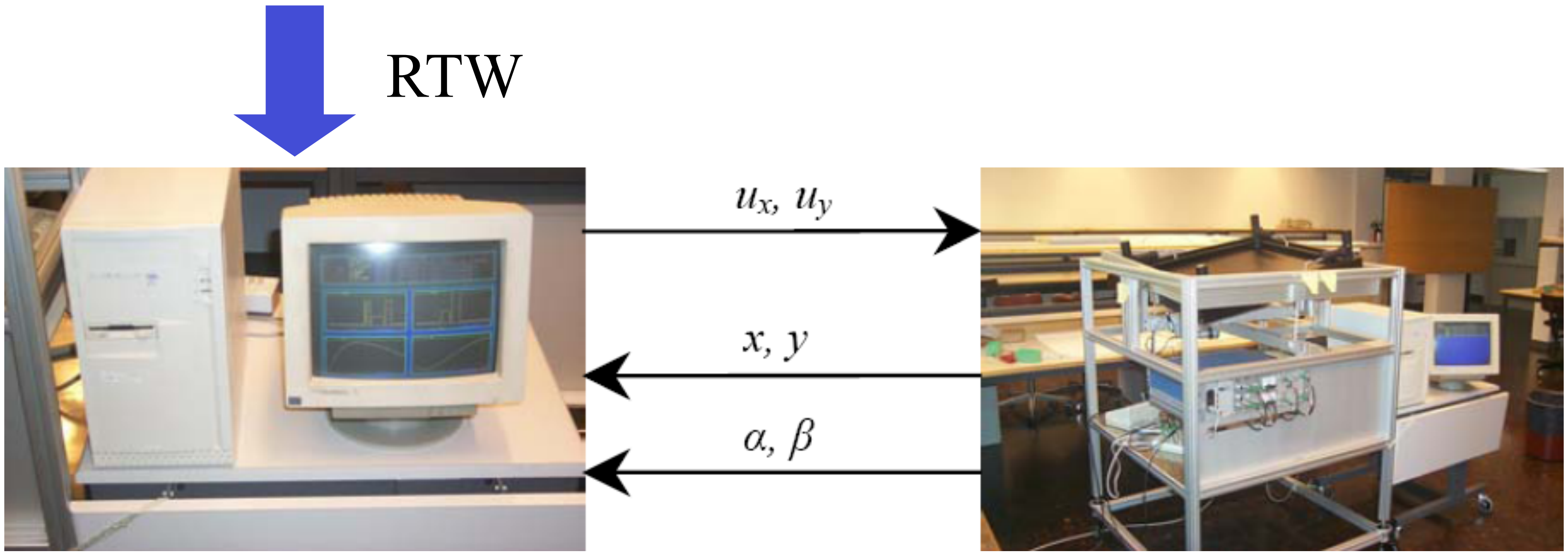
The MPT Framework

Model → Control → Analysis → Deployment

Real Time Workshop

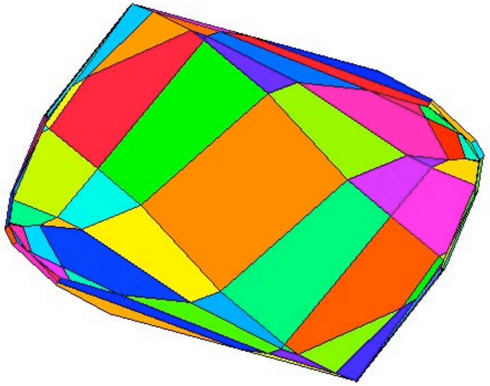


Just press a button…



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;
    }
```

Advanced Features

- Control of time-varying systems
- "Design your own MPC" function

Control of Time-Varying Systems

- Why: allow models with different sampling rates
- How: use one model for each prediction step

```
S1 = mpt_sys('two_tanks_Ts_10')
S2 = mpt_sys('two_tanks_Ts_20')
S3 = mpt_sys('two_tanks_Ts_100')
```

```
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!

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

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

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

- Different features can be combined together, e.g. move blocking, time-varying systems, soft constraints, …
- 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 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);
```

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})
```

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
```

Contraction Constraints

- Task: tell the controller to use at most 2 out of *n* available inputs at each time
- Implementation:

```
for k = 1:length(V.u)
  C = C + set(nnz(V.u{i}) <= 2)
end
```

MPT Characteristics

- Numerically refined problem formulation
- Builds on best available numerical packages, both free and commercial
- Extensible and continuously improving
- Released under an open-source GNU license

MPT in the World

20000+ downloads

Some Users and Areas of Applications

▪ Power electronics	*ETH/ABB*
▪ Autonomous driving	*DaimlerChrysler*
▪ Throttle control	*Uni Zagreb/FORD*
▪ Diesel engine control	*University of Cambridge*
▪ Robotic grasping	*Thales EE*
▪ Steel production	*Arcelor*
▪ Identification	N*ASA*

Acknowledgement & Download

Miroslav Barić	Frank J. Christophersen	Marco Lüthi
Alberto Bemporad	Eric Kerrigan	Saša V. Raković
Pratik Biswas	Adam Lagerberg	Raphael Suard
Francesco Borrelli	Arne Linder	Kari Unneland

Special thanks to:

Komei Fukuda	(CDD)	Jos F. Sturm	(SeDuMi)
Tobias Geyer	(Optimal Merging)	Johan Löfberg	(YALMIP)
Colin Jones	(ESP)	Fabio D. Torrisi	(HYSDEL)
Alex Kurzhanskiy	(Ellipsoidal Toolbox)	Gianni F.-Trecate	(HIT)

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