HYBRID TOOLBOX

FOR REAL-TIME APPLICATIONS

User's Guide

Alberto Bemporad

bemporad@ing.unitn.it

July 14, 2010

This document describes a Matlab/Simulink Toolbox for the synthesis of hybrid controllers for real-time applications.

The author gratefully acknowledges the Scientific Research Laboratory of Ford Motor Company (Dearborn, MI) for continuous support, and in particular Davor Hrovat, Ilya V. Kolmanovsky, and Stefano Di Cairano. Fabio D. Torrisi is also acknowledged for having developed the HYSDEL compiler, Domenico Mignone for having maintained MIQP.M for a long time, Nicolò Giorgetti for having developed Matlab interfaces to several solvers and for continuously testing the toolbox and discussing new features, and Alessandro Alessio, Davide Barcelli, Daniele Bernardini, Filippo Brogi, Giulio Ripaccioli, Sergio Trimboli, for inspiring discussions, suggestions, and for reporting bugs. The European Commission is also acknowledged for supporting many theoretical developments through EU project "CC" (IST-2001-33520). Finally, I thank all the coauthors of my scientific contributions at the basis of the Hybrid Toolbox, discovering theoretical properties and designing algorithms with them was really exciting.

Contents

Chapter 1

Installation

To install the toolbox, you should follow the following procedure:

1. Extract the zip file to a directory (e.g. `C:\MatlabR2007b\toolbox\hybrid`)
2. Run Matlab
3. Add the directory `C:\MatlabR2007b\toolbox\hybrid` and `C:\MatlabR2007b\toolbox\hybrid\utils` to the MATLABPATH.
   (Go to the menu "FILE/SET PATH" from the Matlab Command Window, and then "PATH/ADD TO PATH" from the Path Browser window)

Please read the following carefully when installing the toolbox:

- Errors have been reported when the working directory or the toolbox directory contain spaces, you should avoid paths containing spaces.
- Make sure that the `\utils` directory and working directories have write permissions and that the `.h` and `.dll` files are not read-only.

This manual is available in PDF in the `/manual` subdirectory. Some demos are in the `/demos` directory.

You should have a C-compiler compatible with Matlab (e.g. Microsoft Visual C++ 6.0™, or the free compiler and mex interface MINGW/GNUMEX) to be able to generate mex files.

The toolbox contains LP, QP, and MILP solvers that can be redistributed and/or modified under the terms of the GNU General Public License as published by the Free Software Foundation. The QP solver of the Model Predictive Control Toolbox (The Mathworks, Inc.), the LP/QP solvers of the NAG Foundation Toolbox, the LP/QP solvers of the Optimization Toolbox (The Mathworks, Inc.), the LP/QP/MILP/MIQP solvers of Cplex (Ilog, Inc.), and the LP/QP/MILP/MIQP solvers of Xpress-MP (Dash Optimization) are also supported.

The Model Predictive Control Toolbox for Matlab must be installed for using certain features of the Hybrid Toolbox.

If you plan to use Cplex (Ilog, Inc.) with the Hybrid Toolbox and the mex interface CPLEMEX, you should copy `..\ILOG\CPLEX91\BIN\X86_WIN32\CPLEX91.DLL`

in a visible path (e.g.: `C:\WINDOWS\SYSTEM32`). The precompiled mex file `CPLEXMEX.DLL` was compiled for CPLEX 9.1. If you have an earlier version of Cplex (e.g.: Cplex 9.0) try renaming `CPLEX90.DLL` to `CPLEX91.DLL`.

Since version 1.2.1 of the Hybrid Toolbox, the CPLEX interface CPLEXINT by Mato Baotic is also supported, and successfully tested with CPLEX up to version 11.2.

Chapter 2

Description of the Toolbox

The Hybrid Toolbox offers the following features:

- Design, simulation, and control of hybrid dynamical systems.
- Design of constrained optimal controllers in explicit piecewise affine form for linear and hybrid dynamical systems.
- A Simulink library for simulation and control of linear constrained systems and hybrid systems.
- A multiparametric Quadratic Programming (mpQP), a multiparametric Linear Programming (mpLP) solver, and a multiparametric hybrid optimal control solver for obtaining explicit piecewise affine controllers.
- C-code generation for real-time prototyping.
- Manipulation and visualization of polyhedral objects and polyhedral partitions.
- Demos

Chapter 3

Explicit Constrained Control of Linear Systems

Receding horizon optimal control RHC (also known as Model Predictive Control, MPC) has become the accepted standard for complex constrained multivariable control problems in the process industries. Here at each sampling time, starting at the current state, an open-loop optimal control problem is solved over a finite horizon. At the next time step the computation is repeated starting from the new state and over a shifted horizon, leading to a moving horizon policy. The solution relies on a linear dynamic model, respects all input and output constraints, and optimizes a quadratic performance index. Thus, as much as a quadratic performance index together with various constraints can be used to express true performance objectives, the performance of RHC is excellent. Over the last decade a solid theoretical foundation for RHC has emerged so that in real-life large-scale MIMO applications controllers with non-conservative stability guarantees can be designed routinely and with ease [21].

The big drawback of RHC is the on-line computational effort which may limit its applicability to relatively slow and/or small problems.

In [12], Bemporad et al. have shown how to move the computations necessary for the implementation of RHC off-line while preserving all its other characteristics. This should largely increase the range of applicability of RHC to problems where anti-windup schemes and other ad hoc techniques dominated up to now. Such an explicit form of the controller provides also additional insight for better understanding the control policy of RHC.

Linear RHC is based on the solution of a quadratic program (QP) which needs to be solved to determine the optimal control action. As the coefficients of the linear term in the cost function and the right hand side of the constraints depend linearly on the current state, the quadratic program can be viewed as a *multi-parametric quadratic program* (mp-QP). In [12], the authors analyze the properties of mp-QP, showing that the optimal solution is a piecewise affine function of the vector of parameters. As a consequence, the MPC controller is a piecewise affine control law which not only ensures feasibility and stability, but is also optimal with respect to LQR performance. An algorithm based on a geometric approach

for solving mpQP problems, and therefore obtain explicit RHC controllers, was proposed in [12]. More recently, in [24] the authors proposed a faster algorithm based on an active-set approach, which is implemented in the function `mpqp` described in Appendix D.13.

3.1 Example

Consider the double integrator [12]

$$y(t) = \frac{1}{s^2}u(t),$$

and its equivalent discrete-time state-space representation

$$\left\{ \begin{array}{rcl} x(t+1) & = & \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \\ y(t) & = & \begin{bmatrix} 1 & 0 \end{bmatrix} x(t) \end{array} \right. \tag{3.1}$$

obtained by setting $\ddot{y}(t) \approx \frac{\dot{y}(t+T)-\dot{y}(t)}{T}$, $\dot{y}(t) \approx \frac{y(t+T)-y(t)}{T}$, $T = 1$ s.

We want to regulate the system to the origin while minimizing the quadratic performance measure

$$\sum_{t=0}^{\infty} x'(t)Qx(t) + Ru^2(t)$$

subject to the input constraint $-1 \leq u(t) \leq 1$. This task is addressed by using RHC with prediction horizon $p = 2$, number of free moves $m = 2$, and a terminal weight P solving the Riccati equation with weight matrices $Q = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right]$, $R = 0.01$.

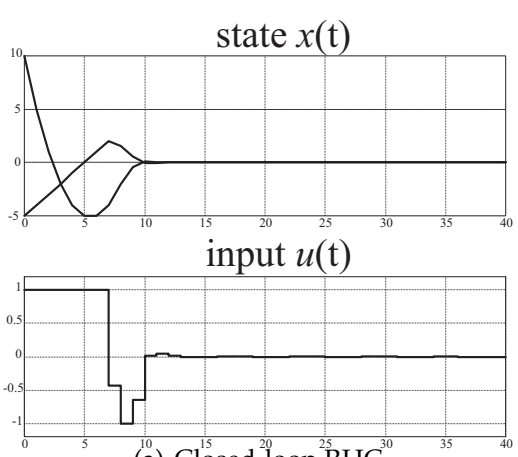
In a neighborhood around the origin, the RHC controller coincides with the constrained linear quadratic regulator $K = [-0.81662 \ -1.7499]$.

The multiparametric Quadratic Programming problem associated with the RHC law is

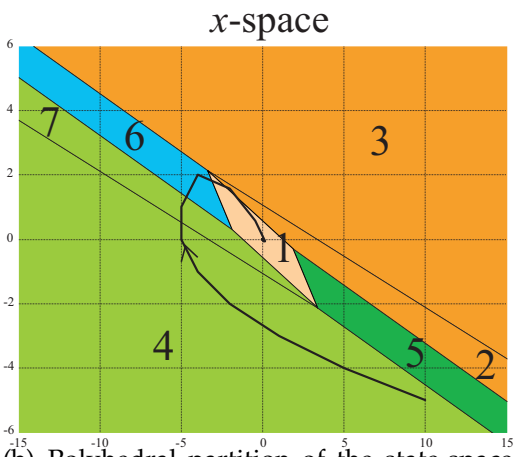
$$\begin{array}{ll} \min\limits_{z} & \frac{1}{2}z'Hz + x'(t)F'z + \frac{1}{2}x'(t)'Hx(t) \\ \text{s.t.} & Gz \leq W + Sx(t) \end{array} \tag{3.2}$$

where $z = [u(t) \ u(t+1)]'$ is the optimization vector, and

$$\begin{array}{rcl} H & = & \begin{bmatrix} 6.0916 & 2.6241 \\ 2.6241 & 1.4996 \end{bmatrix}, \ F = \begin{bmatrix} 3.3675 & 1.2246 \\ 9.3591 & 3.8487 \end{bmatrix}, \ Y = \begin{bmatrix} 3.1429 & 6.5104 \\ 6.5104 & 15.8694 \end{bmatrix} \\ G & = & \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix}, \ W = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \ S = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \end{array}$$



(a) Closed-loop RHC



(b) Polyhedral partition of the state-space and closed-loop MPC trajectories

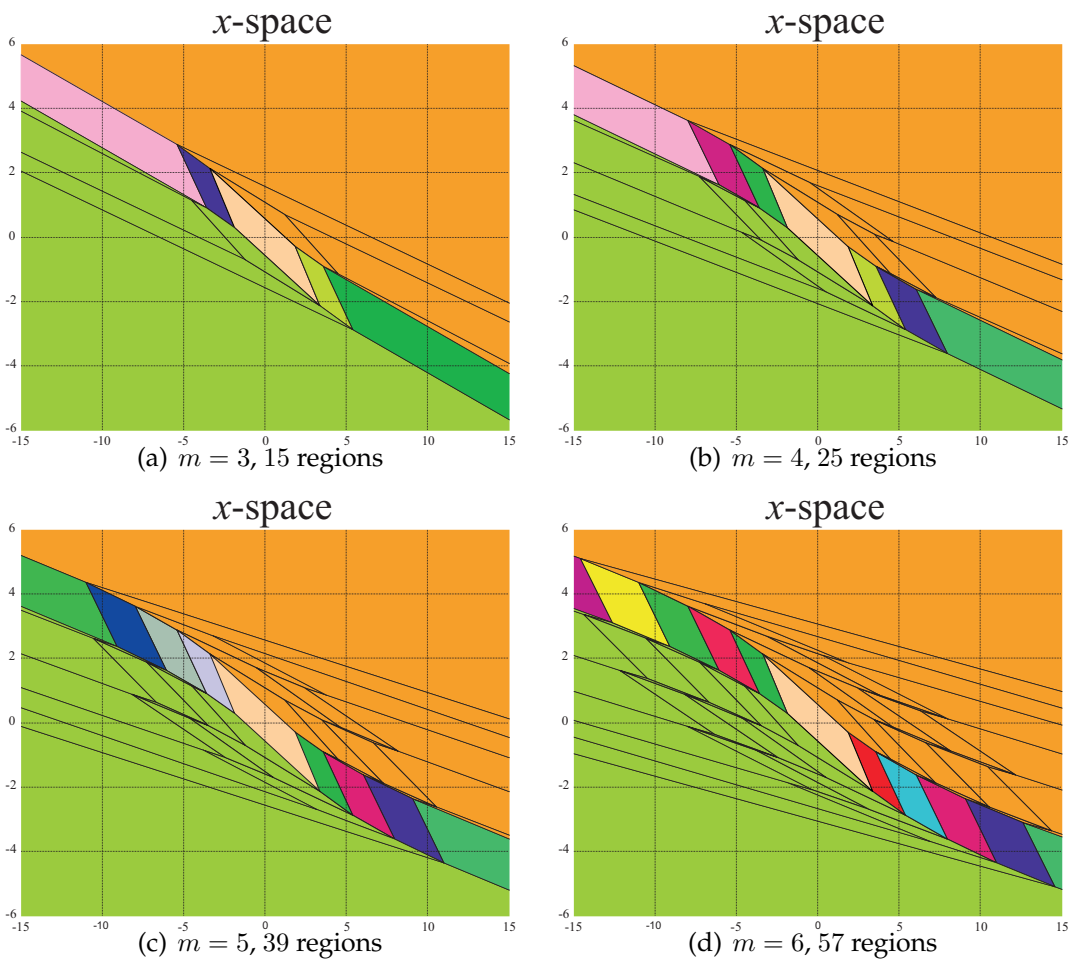
Figure 3.1: Double integrator example (number of free input moves $m = 2$)

and the solution is

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

The corresponding polyhedral partition of the state-space is depicted in Fig. 3.1(b). The same example was solved by increasing the number of degrees of freedom m. The corresponding partitions are reported in Fig. 3.2.

The above example can be reproduced as follows (see the Matlab demo `doubleint.m` in the `demos` folder):



(a) $m = 3$, 15 regions (b) $m = 4$, 25 regions

(c) $m = 5$, 39 regions (d) $m = 6$, 57 regions

Figure 3.2: Partition of the state space for the constrained receding horizon optimal controller: m=number of control degrees of freedom

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

% Define the constrained optimal controller (implicit)
clear limits interval weights
limits.umin=-1;
limits.umax=1;

interval.N=2;

weights.R=.1;
weights.Q=[1 0;0 0];
weights.P='lqr'; % P=solution of Riccati Equation

% Optimal controller based on on-line optimization (implicit)
Cimp=lincon(model,'reg',weights,interval,limits);

% Get the PWA representation of the controller (explicit)
```

```
% Define range of states (=parameters)
range=struct('xmin',-15,'xmax',15);

% Compute explicit version of the controller
Cexp=expcon(Cimp,range);
plot(Cexp)
axis([-15 15 -6 6]);

% Closed-loop simulation
x0=[10,-.3]';
Tstop=40; %Simulation time
[X,U,T,Y,I]=sim(Cexp,model,[],x0,Tstop);
hold on
plot(X(:,1),X(:,2),X(:,1),X(:,2),'d');
```

3.2 Constrained Optimal Control Formulation

The receding horizon control problem is defined either a *regulation* or as a *tracking* problem.

3.2.1 Constrained Regulation to the Origin

The regulator is based on the performance index

$$\begin{array}{ll} \min & x'(t+T|t)Px(t+N|t) + \sum_{k=0}^{N-1} x'(t+k|t)Qx(t+k|t) + u'(t+k)Ru(t+k) + \rho\varepsilon^2 \\ \text{s.t.} & y_{\min} - \varepsilon \le y(t+k|t) \le y_{\max} + \varepsilon,\ k = 1, ..., N_{cy} \\ & u_{\min} \le u(t+k) \le u_{\max},\ k = 0, ..., N_{cu} \\ & u(t+k) = Kx(t+k|t),\ k \ge N_u \\ & x(t+k+1|t) = Ax(t+k|t) + Bu(t+k) \\ & y(t+k|t) = Cx(t+k|t) + Du(t+k) \end{array} \tag{3.3}$$

where A, B, C, D are the state-space matrices of the LTI state-space model, Q, R are weight matrices, and ρ is a penalty on constraint violation. P is a weight on the terminal state, and K is a state-feedback gain. P, K may have different values:

- P, K solve the Riccati equation

$$K = -(R + B'PB)^{-1}B'PA \tag{3.4a}$$

$$P = (A + BK)'P(A + BK) + K'RK + Q. \tag{3.4b}$$

  This choice of P, K correspond to setting $N = +\infty$ in (3.6), and (for $\rho = +\infty$) make (3.6) equivalent to the Linear Quadratic Regulation problem

$$\min \sum_{k=0}^{\infty} x'(t+k)Qx(t+k) + u'(t+k)Ru(t+k)$$

  under the constraints $y_{\min} \le y(t+k) \le y_{\max}$, $u_{\min} \le u(t+k) \le u_{\max}$[1].

[1] The equivalence only holds in a region Ω around the origin. The larger m, N_{cy}, N_{cu}, the larger Ω. See [12,17] for further details

- P solves the Lyapunov equation

$$P = A'PA + Q, \;\; K = 0. \tag{3.5}$$

  This choice of P, K correspond to setting $N = +\infty$ under the assumption $u(t+k) = 0$ for $k \geq N_u$.

- Matrices P, K are provided by the user.

3.2.2 Constrained Tracking

The tracking controller is based on the performance index

$$\begin{array}{ll} \min & \sum_{k=0}^{N-1} [y'(t+k|t) - r(t)] S [y(t+k|t) - r(t)] + \Delta u'(t+k) T \Delta u(t+k) + \rho \varepsilon^2 \\ \text{s.t.} & y_{\min} - \varepsilon \leq y(t+k|t) \leq y_{\max} + \varepsilon, \; k = 1, ..., N_{cy} \\ & u_{\min} \leq u(t+k) \leq u_{\max}, \; k = 0, ..., N_{cu} \\ & \Delta u_{\min} \leq \Delta u(t+k) \leq \Delta u_{\max}, \; k = 0, ..., N_{cu} \\ & u(t+k) = 0, \; k \geq N_u \\ & x(t+k+1|t) = Ax(t+k|t) + B[u(t+k-1|t) + \Delta u(t+k)] \\ & y(t+k|t) = Cx(t+k|t) + D[u(t+k-1|t) + \Delta u(t+k)] \end{array} \tag{3.6}$$

where $\Delta u(t) = u(t) - u(t-1)$ represents the input increment.

A much more versatile formulation and handling of controllers based on on-line quadratic optimization can be found in the Model Predictive Control Toolbox for Matlab.

3.2.3 Computation of the Explicit Representation

The constrained finite time optimal control problems described above can be converted into the multiparametric Quadratic Program (mpQP)

$$\begin{array}{ll} \min & \frac{1}{2} z' Q z + \theta' C' z \\ \text{s.t.} & Gz \leq W + S\theta \end{array} \tag{3.7}$$

where $z = [u'(0), \ldots, u'(m-1)]'$ is the vector to be optimized, and $\theta = \begin{bmatrix} x(t) \\ r(t) \end{bmatrix}$ is the vector of parameters. When output constraints are softened as $y_{\min} - \epsilon \leq y(t+k|t) \leq y_{\max} + \epsilon$, $k = 1, ..., N_{cy}$, and the term $\rho\epsilon^2$ is added in the cost function to penalize constraint violation, $z = [u'(0), \ldots, u'(m-1), \epsilon]'$.

The vector of parameters θ ranges within the box $\theta_{\min} \leq \theta \leq \theta_{\max}$. Function `mpqp` implements the active-set method proposed in [24] to solve problem (3.7).

The output argument of the multiparametric quadratic solver is a partition of neighboring convex polytopes $H_i\theta \leq K_i$ in the θ-space and the optimizer $z(\theta) = F_i\theta + G_i$ in each region #i. See Appendix C for details on how polyhedral partitions are stored.

Explicit controllers can be obtained through the Hybrid Toolbox by conversion of MPC objects defined using the Model Predictive Control Toolbox for Matlab.

3.2.4 Output-Feedback Controllers

An output feedback constrained optimal controller is obtained by computing the control law as a function of an estimate of the state vector, obtained via a state observer, see Section 6.2.1. See also the demo `dcmotor` in the `demos/linear` directory for an example.

Chapter 4

Hybrid Dynamical Systems

The mathematical model of a system is traditionally associated with differential or difference equations, typically derived from physical laws governing the dynamics of the system under consideration. Consequently, most of the control theory and tools have been developed for such systems, in particular for systems whose evolution is described by smooth linear or nonlinear state transition functions. On the other hand, in many applications the system to be controlled is also constituted by parts described by *logic*, such as for instance on/off switches or valves, gears or speed selectors, and evolutions dependent on if-then-else rules. Often, the control of these systems is left to schemes based on heuristic rules inferred from practical plant operation.

Recent technological innovations have caused a considerable interest in the study of dynamical processes of a heterogeneous continuous and discrete nature, denoted as *hybrid systems*. The peculiarity of hybrid systems is the interaction between continuous-time dynamics (governed by differential or difference equations), and discrete dynamics and logic rules (described by temporal logic, finite state machines, if-then-else conditions, discrete events, etc.) and discrete components (on/off switches, selectors, digital circuitry, software code, etc.).

Hybrid systems switch among many operating modes, where each mode is governed by its own characteristic dynamical laws. Mode transitions are triggered by variables crossing specific thresholds (state events), by the elapse of certain time periods (time events), or by external inputs (input events) [2]. A typical example of hybrid systems are embedded systems, constituted by dynamical components governed by logical/discrete decision components. Complex systems organized in hierachical way, where for instance discrete planning algorithms at the higher level interact with continuous control algorithms and processes at the lower level, are another example of hybrid systems.

As an example of hybrid control problem consider the design of a cruise control system that commands the gear shift, the engine torque, and the braking force in order to track a desired vehicle speed while minimizing fuel consumption and emissions. Designing a control law that optimally selects both the discrete inputs (gears) and continuous inputs (torque and brakes) requires a hybrid model that includes the continuous dynamics of the power train, the discrete logic of the

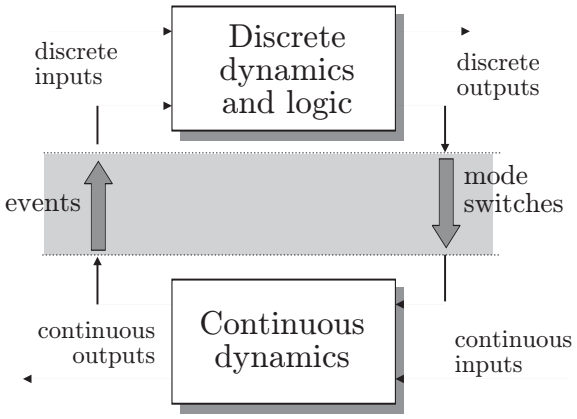


Figure 4.1: Hybrid systems. Logic-based discrete dynamics and continuous dynamics interact through events and mode switches

gearbox, and consumption/emission maps [4].

We focus here on discrete-time hybrid systems, denoted as *discrete hybrid automata* (DHA) [27], whose continuous dynamics is described by linear difference equations and whose discrete dynamics is described by finite state machines, both synchronized by the same clock.

A particular case of DHA is the popular class of *piecewise affine* (PWA) systems first introduced by Sontag [23]. Essentially, PWA are switched affine systems whose mode only depends on the current location of the state vector. More precisely, the state space is partitioned into polyhedral regions, as depicted in Figure 4.2, and each region is associated with a different affine state-update equation (more generally, the partition is defined in the combined space of state and input vectors). We will actually show that DHA and PWA systems are equivalent model classes, and hence, in particular, that generic DHA systems can be converted to equivalent PWA systems.

4.1 HYSDEL Models

We designed a modeling language to describe DHA models, called HYbrid System DEscription Language (HYSDEL). In this section we will detail the language capabilities and we will show how DHA systems can be modeled within HYSDEL. The HYSDEL description is an abstract modeling step. The associated HYSDEL compiler then translates the description into several computational models, in particular into MLD and PWA form.

A HYSDEL list is composed of two parts: The first one, called INTERFACE, contains the declaration for all the variables and parameters, so that it is possible to make the proper type checks. The second part, IMPLEMENTATION, is composed of specialized sections where the relations among the variables are described, these sections are described next.

AD SECTION The HYSDEL section AD allows one to define Boolean variables

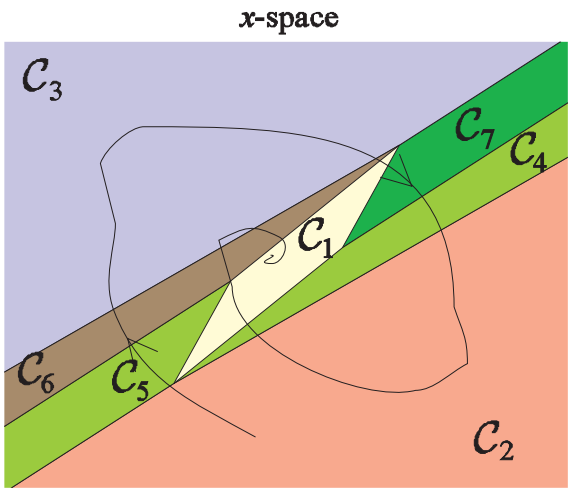


Figure 4.2: Piecewise affine systems. Mode switches are triggered by threshold events

from continuous ones, such as "yes/no" events representing that certain continuous variables have passed some given thresholds. HYSDEL does not provide explicit access to the time instance, however this limitation can be easily overcome by adding a continuous state variable t such that $t' = t + T_s$, where T_s is the sampling time. Examples include level indicator variables, operational alarms, etc.

Example: In a water tank with inflow Q, the sensor δ provides the signal 1 if and only if the liquid level $h \geq h_{max}$.

LOGIC SECTION The section LOGIC allows one to specify arbitrary functions of Boolean variables: In particular the mode selector is a Boolean function and therefore it can be modeled in this section.

DA SECTION The HYSDEL section DA defines continuous variables according to if-then-else conditions on Boolean variables. This section is particularly useful to model parts of the switched affine system (SAS) dynamics.

Example: The command signal u to an electric motor is passed through a nonlinear amplifier, which has two modes of operation: high gain and low gain. In "low gain" the amplifier completely rejects the noise d, while in "high gain" it only attenuates it. The Boolean input $l = 1$ selects the low gain mode. The actual signal u_{comp} applied to the motor is $u_{\text{comp}} = u$ if $l = 1$, otherwise $u_{\text{comp}} = 2.3u + .4d$.

CONTINUOUS SECTION The CONTINUOUS section describes the linear dynamics, expressed as difference equations.

LINEAR SECTION HYSDEL allows also to define a continuous variable as an affine function of continuous variables in the LINEAR section. This section, together with the CONTINUOUS and AD sections allows more flexibility when modeling the SAS. This extra flexibility allows algebraic loops that may render unde-

fined the trajectories of the model. The HYSDEL compiler integrates a semantic checker that is able to detect and report such abnormal situations.

AUTOMATA SECTION The AUTOMATA section specifies the state transition equations of the finite state machine (FSM) as Boolean functions $x_b'(k) = f_{\rm B}(x_b(k), u_b(k), \delta_e(k))$.

OUTPUT SECTION The OUTPUT section allows specifying static linear and logic relations for the output vector $y = \left[\begin{smallmatrix} y_r \\ y_b \end{smallmatrix}\right]$.

Finally HYSDEL allows one more section:

MUST SECTION The MUST section specifies constraints on continuous and Boolean variables, i.e., linear constraints and Boolean formulas, and therefore it allows restricting the sets of feasible real and binary states, inputs, and outputs (more generally, the MUST section allows also mixed constraints on states, inputs, and outputs).

A more detailed description of the syntax of HYSDEL is available in the HYSDEL user's guide contained in the `doc/` directory.

4.1.1 Mixed Logical Dynamical (MLD) Systems

In [10] a class of hybrid systems has been introduced in which logic, dynamics and constraints are integrated, of the form

$$x(k+1) = Ax(k) + B_1 u(k) + B_2 \delta(k) + B_3 z(k) \tag{4.1a}$$

$$y(k) = Cx(k) + D_1 u(k) + D_2 \delta(k) + D_3 z(k) \tag{4.1b}$$

$$E_2 \delta(k) + E_3 z(k) \leq E_1 u(k) + E_4 x(k) + E_5, \tag{4.1c}$$

where $x(k) = \left[\begin{smallmatrix} x_c(k) \\ x_\ell(k) \end{smallmatrix}\right]$ is the state vector, $x_c(k) \in \mathbb{R}^{n_c}$ and $x_\ell(k) \in \{0,1\}^{n_\ell}$, $y(k) = \left[\begin{smallmatrix} y_c(k) \\ y_\ell(k) \end{smallmatrix}\right] \in \mathbb{R}^{p_c} \times \{0,1\}^{p_\ell}$ is the output vector, $u(k) = \left[\begin{smallmatrix} u_c(k) \\ u_\ell(k) \end{smallmatrix}\right] \in \mathbb{R}^{m_c} \times \{0,1\}^{m_\ell}$ is the input vector, $z(k) \in \mathbb{R}^{r_c}$ and $\delta(k) \in \{0,1\}^{r_\ell}$ are auxiliary variables, A, B_i, C, D_i and E_i denote real constant matrices, E_5 is a real vector, $n_c > 0$, and p_c, m_c, r_c, n_ℓ, p_ℓ, m_ℓ, $r_\ell \geq 0$. Without loss of generality, we assumed that the continuous components of a mixed-integer vector are always the first. Inequalities (4.1c) must be interpreted componentwise. Systems that can be described by model (4.1) are called Mixed Logical Dynamical (MLD) systems. Contrarily to [10], we allow here that the input vector $u(k)$ and state vector $x(k)$ may have unbounded components.

The MLD system (4.1) is called *completely well-posed* if $\delta(k)$ and $z(k)$ are uniquely defined by (4.1c) in their domain, once $x(k)$ and $u(k)$ are assigned [10]. From (4.1a)–(4.1b) this implies that also $x(k+1)$, $y(k)$ are uniquely defined functions of $x(k)$, $u(k)$.

The MLD formalism allows specifying the evolution of continuous variables through linear dynamic equations, of discrete variables through propositional logic statements and automata, and the mutual interaction between the two. The key idea of the approach consists of embedding the logic part in the state equations by transforming Boolean variables into 0-1 integers, and by expressing the

relations as mixed-integer linear inequalities (see [10, 27] and references therein). MLD systems are therefore capable of modeling a broad class of systems, in particular those systems that can be modeled through the hybrid system description language HYSDEL [27].

4.1.2 Example

Consider the following example of hybrid system [10] (demo `bm99sim.m`):

$$\begin{cases} x(k+1) &= 0.8 \begin{bmatrix} \cos\alpha(k) & -\sin\alpha(k) \\ \sin\alpha(k) & \cos\alpha(k) \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(k) \\ y(k) &= \begin{bmatrix} 0 & 1 \end{bmatrix} x(k) \\ \alpha(k) &= \begin{cases} \frac{\pi}{3} & \text{if } \begin{bmatrix} 1 & 0 \end{bmatrix} x(k) \geq 0 \\ -\frac{\pi}{3} & \text{if } \begin{bmatrix} 1 & 0 \end{bmatrix} x(k) < 0 \end{cases} \end{cases} \tag{4.2}$$

Assuming that $[-10, 10] \times [-10, 10]$ is the set of states $x(k)$ of interest, and that $[-1.1, 1.1]$ is the set of inputs $u(k)$ of interest, using HYSDEL we describe (4.2) as

```
/* 2x2 PWA system */

SYSTEM pwa {

INTERFACE {
    STATE { REAL x1 [-10,10];
            REAL x2 [-10,10];
        }
    INPUT { REAL u [-1.1,1.1];
        }
    OUTPUT{ REAL y;
        }
    PARAMETER {
        REAL alpha = 1.0472; /* 60 deg in radiants */
        REAL C = cos(alpha);
        REAL S = sin(alpha);
            }
    }

IMPLEMENTATION {
        AUX { REAL z1,z2;
              BOOL sign; }
        AD  { sign = x1<=0; }

        DA  { z1 = {IF sign THEN 0.8*(C*x1+S*x2)
                    ELSE 0.8*(C*x1-S*x2) };
              z2 = {IF sign THEN 0.8*(-S*x1+C*x2)
                    ELSE 0.8*(S*x1+C*x2) };  }

        CONTINUOUS { x1 = z1;
                     x2 = z2+u; }

        OUTPUT { y = x2;  }
      }
}
```

(see `bm99.hys`). The equivalent MLD system has the form and obtain the

equivalent MLD form

$$
\begin{aligned}
&x(k+1) = \left[\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right] z(k) + \left[\begin{smallmatrix}0\\ 1\end{smallmatrix}\right] u(k)\\
&y(k) = \left[\begin{smallmatrix}0 & 1\end{smallmatrix}\right] x(k)\\
&\begin{bmatrix} -10\\ 10\\ 21.8564\\ 21.8564\\ -21.8564\\ -21.8564\\ 21.8564\\ 21.8564\\ -21.8564\\ -21.8564 \end{bmatrix} \delta(k) + \begin{bmatrix} 0 & 0\\ 0 & 0\\ -1 & 0\\ 1 & 0\\ -1 & 0\\ 1 & 0\\ 0 & -1\\ 0 & 1\\ 0 & -1\\ 0 & 1 \end{bmatrix} z(k) \leq \begin{bmatrix} 1 & 0\\ -1 & 0\\ -0.4 & -0.6928\\ 0.4 & 0.6928\\ -0.4 & 0.6928\\ 0.4 & -0.6928\\ 0.6928 & -0.4\\ -0.6928 & 0.4\\ -0.6928 & -0.4\\ 0.6928 & 0.4 \end{bmatrix} x(k) + \begin{bmatrix} 0\\ 10\\ 21.8564\\ 21.8564\\ 0\\ 0\\ 21.8564\\ 21.8564\\ 0\\ 0 \end{bmatrix}.
\end{aligned}
\tag{4.3}
$$

Now go to the directory `demos/hybrid/` and type:

```
S=mld('bm99')
```

The corresponding output is

```
S is an MLD hybrid model generated from the HYSDEL file <bm99.hys>

  2 states    (2 continuous, 0 binary)
  1 inputs    (1 continuous, 0 binary)
  1 outputs   (1 continuous, 0 binary)

  2 continuous auxiliary variables
  1 binary auxiliary variables
 10 mixed-integer linear inequalities

sampling time: 1

Type S.rowinfo for information about dynamics and constraints.
Type S.symtable for information about variables.
Type "struct(S)" for extra details.
```

The MLD system equations are stored in the MLD object `S` (try typing `S.A`, `S.B1`, `S.E2`, `S.E3`, `S.E4`, or `S.E5`).

4.1.3 Piecewise Affine Systems

Piecewise affine (PWA) systems are described by

$$
\begin{aligned}
x(k+1) &= A_i x(k) + B_i u(k) + f_i \\
y(k) &= C_i x(k) + D_i u(k) + g_i
\end{aligned}
\quad \text{for} \begin{bmatrix} x(k) \\ u(k) \end{bmatrix} \in \Omega_i, \tag{4.4}
$$

where $u(k) \in \mathbb{R}^m$, $x(k) \in \mathbb{R}^n$ and $y(k) \in \mathbb{R}^p$ denote the input, state and output, respectively, at time k, $\Omega_i \triangleq \{\left[\begin{smallmatrix} x \\ u \end{smallmatrix}\right] : \ H_{ix}x + H_{iu}u \leq K_i\}$, $i = 1, \ldots, s$, are convex (possibly unbounded) polyhedra in the input+state space. A_i, B_i, C_i, D_i, H_{ix} and H_{iu} are real matrices of appropriate dimensions and f_i and g_i are real vectors for all $i = 1, \ldots, s$.

PWA systems have been studied by several authors (see [10, 16, 18, 23] and references therein) as they form the "simplest" extension of linear systems that

can still model nonlinear and non-smooth processes with arbitrary accuracy and are capable of handling hybrid phenomena, such as linear-threshold events and mode switching.

A PWA system of the form (4.4) is called *well-posed*, if (4.4) is uniquely solvable in $x(k+1)$ and $y(k)$, once $x(k)$ and $u(k)$ are specified. A necessary and sufficient condition for the PWA system (4.4) to be well-posed over $\Omega \triangleq \cup_{i=1}^{s} \Omega_i$ is therefore that $x(k+1)$, $y(k)$ are single-valued PWA functions of $x(k)$, $u(k)$. Therefore, typically the sets Ω_i have mutually disjoint interiors, and are often defined as the partition of a convex polyhedral set Ω. In case of discontinuities of the PWA functions over overlapping boundaries of the regions Ω_i, one may ensure well-posedness by writing some of the inequalities in the form $(H_{ix})^j x + (H_{iu})^j u < K_i^j$, where ^{j} denotes the j-th row. Although this would be important from a system theoretical point of view, it is not of practical interest from a numerical point of view, as "$<$" cannot be represented in numerical algorithms working in finite precision.

4.2 Equivalence of DHA, MLD, and PWA Models

In [3] it is shown that MLD systems can be transformed into equivalent PWA systems. Equivalence means that for the same initial conditions and input sequences the trajectories of the system are identical. Once the HYSDEL model of a system is available, the equivalent MLD and PWA models can be generated in Matlab.

4.2.1 Example

Consider again the example in demo `bm99sim.m`. To convert the MLD system to PWA form, type `P=pwa(S)`. The corresponding output is

```
P is a PWA hybrid model defined over 2 polyhedral regions and with
  2 state(s)    (2 continuous, 0 binary)
  1 input(s)    (1 continuous, 0 binary)
  1 output(s)  (1 continuous, 0 binary)

P was generated from the MLD system 'S' (HYSDEL model <bm99.hys>, sampling time = 1).
Type "struct(P)" for full information.
```

The corresponding equivalent PWA object is now contained in the workspace variable `P`, and was computed using the algorithm of [3]. `P` contains all the information about the different dynamical affine submodels and the polyhedral cells where they are defined. For instance, the second dynamics (A_2, B_2, f_2, C_2, D_2, g_2) and corresponding cell $\Omega_2 = \{\left[\begin{smallmatrix} x \\ u \end{smallmatrix}\right] : H_{2x}x + H_{2u}u \leq K_2\}$ can be retrieved as follows:

```
A2=P.A{2};
B2=P.B{2};
f2=P.f{2};
```

```
C2=P.C{2};
D2=P.D{2};
g2=P.g{2};
H2x=P.Hx{2};
H2u=P.Hu{2};
K2=P.K{2};
```

You can plot the section of the cell in the x-space $\{x : H_{2x}x \leq K_2\}$ by typing

```
polyplot(H2x,K2)
```

(see also `plot` and `plotsection` for plotting 2-D sections of PWA system partitions).

4.3 Control of Hybrid Systems

Controlling a system means to choose the command input signals so that the output signals tracks some desired reference trajectories.

In [6, 10] the authors showed how mixed-integer programming (MIP) can be efficiently used to determine optimal control sequences. It was also shown that when optimal control is implemented in a receding horizon fashion by repeatedly solving MIPs on-line, this leads to an asymptotically stabilizing control law. For those cases where on-line optimization is not viable, [6, 7] proposed multiparametric programming as an effective means for synthesizing piecewise affine optimal controllers, that solve in state-feedback form the finite-time hybrid optimal control problem with criteria based on linear (1-norm, ∞-norm) and quadratic (squared Euclidean norm) performance objectives. Such a control design flow for hybrid systems was applied to several industrial case studies, in particular to automotive problems where the simplicity of the control law is essential for its applicability [4, 9, 20].

Receding horizon ideas can be applied to control hybrid models. Assume we want the output $y(t)$ to track a reference signal y_r, and let x_r, u_r, z_r be a desired references for the state, input, and auxiliary variables. Let t be the current time,

and $x(t)$ the current state. Consider the following optimal control problem

$$\min_{\{u,\delta,z\}_0^{N-1}} J(\{u,\delta,z\}_0^{N-1}, x(t)) \triangleq \|Q_{xN}(x(N|t)-x_r)\|_p + \sum_{k=1}^{N-1} \|Q_x(x(k)-x_r)\|_p +$$

$$+ \sum_{k=0}^{N-1} \|Q_u(u(k)-u_r)\|_p + \|Q_z(z(k|t)-z_r)\|_p + \|Q_y(y(k|t)-y_r)\|_p \tag{4.5a}$$

$$\text{s.t.} \begin{cases} x(0|t) &= x(t) \\ x(k+1|t) &= Ax(k|t) + B_1u(k) + B_2\delta(k|t) + B_3z(k|t) \\ y(k|t) &= Cx(k|t) + D_1u(k) + D_2\delta(k|t) + D_3z(k|t) \\ E_2\delta(k|t) &+ E_3z(k|t) \le E_1u(k) + E_4x(k|t) + E_5 \\ u_{\min} &\le u(t+k) \le u_{\max},\ k=0,1,\ldots,N-1 \\ x_{\min} &\le x(t+k|t) \le x_{\max},\ k=1,\ldots,N \\ y_{\min} &\le y(t+k) \le y_{\max},\ k=0,\ldots,N-1 \\ S_x x(N|t) &\le T_x \end{cases} \tag{4.5b}$$

where N is the optimal control interval, $x(k|t)$ is the state predicted at time $t+k$ resulting from the input $u(t+k)$ to (4.1) starting from $x(0|t)=x(t)$, $u_{\min}$, $u_{\max}$, $y_{\min}$, $y_{\max}$, and $x_{\min}$, $x_{\max}$ are hard bounds on the inputs, outputs, and states, respectively, and $\{x:\ S_x x \le T_x\}$ is a final target polyhedral subset of the state-space $\mathbb{R}^n$. In (4.5a), $\|Qx\|_p = x'Qx$ for $p=2$ and $\|Qx\|_p = \|Qx\|_\infty$ for $p=\infty$.

Assume for the moment that the optimal solution $\{u_t^*(0), \ldots, u_t^*(N-1), \delta_t^*(0), \ldots, \delta_t^*(N-1), z_t^*(0), \ldots, z_t^*(N-1)\}$ exists. According to the *receding horizon* philosophy, set

$$u(t) = u_t^*(0), \tag{4.6}$$

disregard the subsequent optimal inputs $u_t^*(1), \ldots, u_t^*(N-1)$, and repeat the whole optimization procedure at time $t+1$. The control law (4.5)–(4.6) provides an extension of MPC to hybrid models, and relies upon the solution of the mixed-integer program (4.5). The exact formulation of the mixed-integer linear program (MILP) corresponding to problem (4.5) is reported in Appendix A for the case $p=\infty$.

Problem (4.5) is formulated for *hard* constraints. However, in practice it is often useful to treat constraints as *soft* by adding a scalar panic variable $\rho \ge 0$:

$$u_{\min} - \mathbf{1_m}\rho \le u(t+k) \le u_{\max} + \mathbf{1_m}\rho,\ k=0,1,\ldots,N-1 \tag{4.7a}$$

$$x_{\min} - \mathbf{1_n}\rho \le x(t+k|t) \le x_{\max} + \mathbf{1_n}\rho,\ k=1,\ldots,N \tag{4.7b}$$

$$y_{\min} - \mathbf{1_p}\rho \le y(t+k) \le y_{\max} + \mathbf{1_p}\rho,\ k=0,\ldots,N-1 \tag{4.7c}$$

$$S_x x(N|t) \le T_x + \mathbf{1_q}\rho \tag{4.7d}$$

where 1_h is a column vector of ones of length h, and by adding the term $\|Q_\rho \rho\|_p$ in cost function (4.5a), which penalizes constraint violations.

By default $p=\infty$, $Q_x = I$, $Q_u = 0.1I$, $Q_y = I$, $Q_z = 0$, $Q_{xN} = Q_x$, $Q_\rho = \infty$ (hard constraints), $u_{\max} = -u_{\min} = \infty$, $x_{\max} = -x_{\min} = \infty$, $y_{\max} = -y_{\min} = \infty$, and $\{x:\ S_x x \le T_x\} = \mathbb{R}^n$.

4.3.1 Example

Consider again the example in demo `bm99sim.m`. We want to obtain a controller based on the optimal control problem

$$\min_{u(0),u(1),\delta(0|t),\delta(1|t),z(0|t),z(1|t)} \sum_{k=0}^{2} \|y(k|t) - r_y(t))\|_\infty \tag{4.8a}$$

$$\text{s.t.} \begin{cases} x(t|t) = x(t) \\ \text{MLD model (4.3)} \\ -1 \le u(t), u(t-1) \le 10 \\ -10 \le x(1|t), x(2|t) \le 10. \end{cases} \tag{4.8b}$$

We define a controller object `C`:

```
clear Q refs limits
refs.y=1; % output references (no state, input, zeta references)
Q.y=1;
Q.rho=Inf;  % Hard constraints
Q.norm=Inf; % Infinity norm
N=2;

limits.umin=-1;
limits.umax=1;
limits.xmin=[-10;-10];
limits.xmax=[10;10];

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

The corresponding output is

```
C is a hybrid controller based on MLD model S <bm99.hys>

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

 10 optimization variable(s) (8 continuous, 2 binary)
 36 mixed-integer linear inequalities
sampling time = 1, MILP solver = glpk

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

The MILP problem solving (4.8) has the form

$$\begin{aligned} \min \quad & f'q \\ \text{s.t.} \quad & Aq \le b + C_x x(t) + C_r^y r_y(t) \end{aligned} \tag{4.9}$$

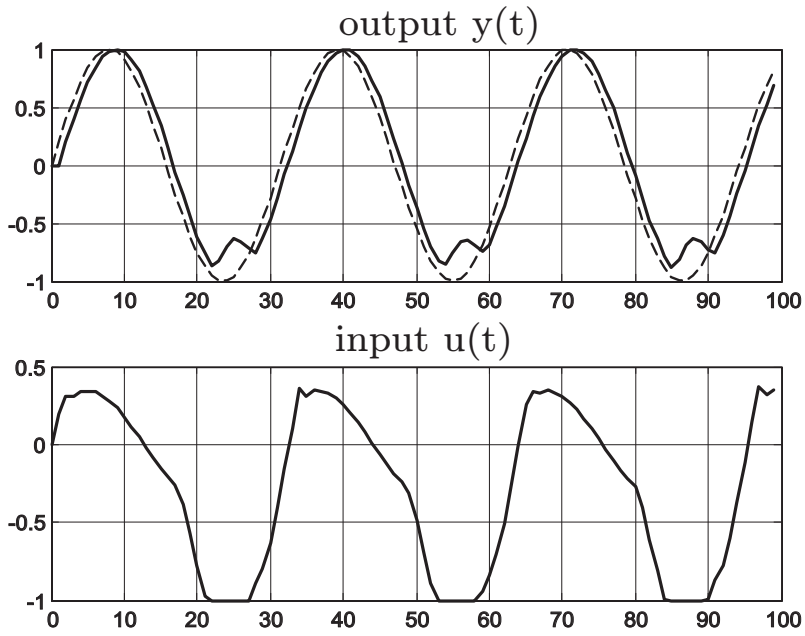


Figure 4.3: Closed-loop system constituted by the hybrid system (4.3) and the controller (4.8).

where $q_c = [u(0), u(1), \delta(0|t), \delta(1|t), z(0|t), z(1|t)]$ are the continuous and binary optimization variables. The matrices of (4.9) are stored in the HYBCON object contained in the workspace variable `C`, in particular f=`C.f`, A=`C.A`, b=`C.b`, C_x=`C.Cx`, C_r^y=`C.Cr.y`.

We want to simulate now the closed-loop system constituted by the hybrid system (4.3) and the controller (4.8). To this end, typing

```
Tstop=100;
x0=[0;0];
r.y=sin((0:99)'/5);
sim(C,S,r,x0,Tstop);
```

produces the plot shown in Figure 4.3.

To switch to the squared 2-norm (while maintaining the same controller weights, constraints, horizons, and referenced variables) type:

```
Q.norm=2; % 2-norm
C=hybcon(S,Q,N,limits,refs);
```

Terminal state constraints of the form $S_x x(N|t) \leq T_x$ can be enforced by defining `limits.Sx`=S_x, `limits.Tx`=T_x. For controllers that must be processed by multiparametric solvers, the set $\{x : S_x x \leq T_x\}$ should be full-dimensional in order to prevent numerical problems.

4.4 Explicit Form of the Optimal Hybrid Controller

There is an alternative to on-line mixed integer optimization for implementing controllers for hybrid systems.

By generalizing the result of [12] for linear systems to hybrid systems and handling the state vector $x(t)$, which appears in the the linear part of the objective function and of the rhs of the constraints, as a vector of parameters, in [5,6] it was shown how to transform the hybrid control law (4.5) into the piecewise affine form

$$\begin{array}{rl} u(t) = & F_i \begin{bmatrix} x(t) \\ r(t) \end{bmatrix} + g_i, \text{ for} \\ & \begin{bmatrix} x(t) \\ r(t) \end{bmatrix} \in \Omega_i \triangleq \{ \begin{bmatrix} x \\ r \end{bmatrix} : \; H_i \begin{bmatrix} x \\ r \end{bmatrix} \leq S_i \}, \; i = 1, \ldots, s \end{array} \tag{4.10}$$

where $\cup_{i=1}^{s} \Omega_i$ is the set of states+references for which a feasible solution to (4.5) exists.

There are several approaches to compute the explicit representation (4.10). Here, we use a combination of reachability analysis, multiparametric linear programming, and computational geometry for comparison of convex piecewise affine functions (see function `pwaopt.m`). In case of infinity norms, the explicit hybrid optimal controller is returned as a piecewise affine function. In case of squared Euclidean norms, the controller is currently returned as a collection of (possibly overlapping and redundant) piecewise affine maps (techniques for simplifying the collection are currently under development).

4.4.1 Example

Consider again the example in demo `bm99sim.m`. We want to obtain the explicit form of controller (4.8). To this end, we define a range of parameters where we want to solve the problem, namely the set of states $x(t)$ whose components are between -10 and 10, and of references $r_y(t)$ between -1 and 1:

```
range.xmin=[-10;-10];
range.xmax=[10;10];
range.refymin=-1;
range.refymax=1;
E=expcon(C,range,options)
```

The corresponding output is

```
E is an explicit controller (based on controller C)
  3 parameter(s)
  1 input(s)
  5 partition(s)
sampling time = 1

The controller is for hybrid systems (tracking)
This is an output-feedback controller.
```

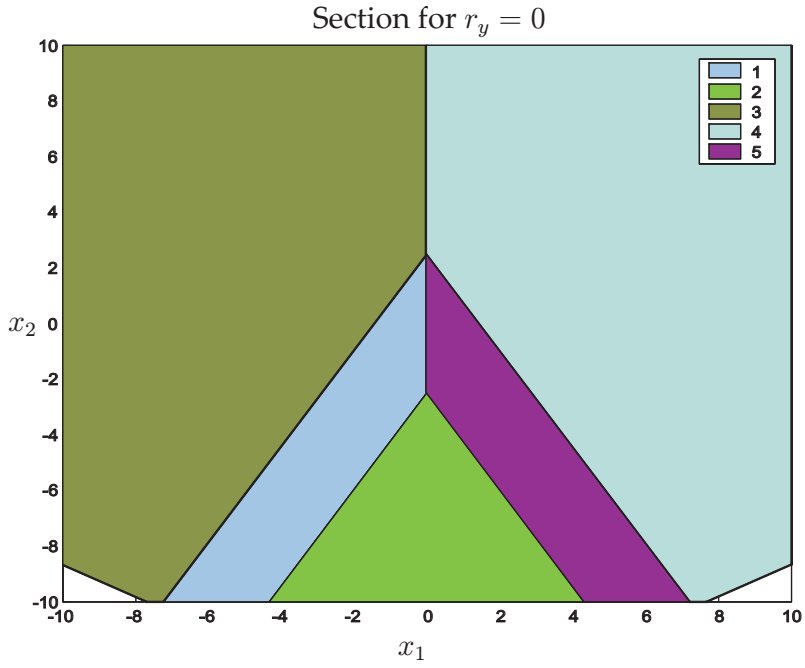


Figure 4.4: Partition associated with the explicit hybrid controller equivalent to (4.8).

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

The partition consist of 5 polyhedral regions in $\mathbb{R}^3$. A section for $r_y = 0$ can be plotted using the command `plotsection(E,3,0)`, and is depicted if Figure 4.4.

The piecewise affine controller (4.10) and the MPC controller (4.5) *are equal*, in the sense that they produce the same control action, and therefore share the same stabilizing and optimality properties. The difference is only in the implementation: for the form (4.10), on-line computation reduces to a function evaluation, instead of a mixed-integer linear program.

The explicit representation of the MPC controller discussed above is significant for several reasons. First of all, it gives some insight into the mathematical structure of the controller which is otherwise hidden behind the optimization formalism. Furthermore, it offers an alternative route to an efficient controller implementation, opening up the route to use MPC in "fast" and "cheap" systems where the on-line solution of a mixed-integer program is prohibitive.

4.5 Reachability Analysis and Verification of Safety Properties

Although simulation allows to probe a model for a certain initial condition and input excitation, any analysis based on simulation is likely to miss the subtle phenomena that a model may generate, especially in the case of hybrid models. Reachability analysis (also referred to as "safety analysis" or "formal verifi-

cation"), aims at detecting if a hybrid model will eventually reach unsafe state configurations or satisfy a temporal logic formula [1] for *all* possible initial conditions and input excitations within a prescribed set. Reachability analysis relies on a reach set computation algorithm, which is strongly related to the mathematical model of the system.

Timed automata and hybrid automata have proved to be a successful modeling framework for formal verification and have been widely used in the literature. The starting point for both models is a finite state machine equipped with continuous dynamics. In the theory of *timed automata*, the dynamic part is the continuous-time flow $\dot{x} = 1$. Efficient computational tools complete the theory of timed automata and allow one to perform verification and scheduling of such models. Timed automata were extended to *linear hybrid automata* [1], where the dynamics is modeled by the differential inclusion $a \leq \dot{x} \leq b$. Specific tools allow one to verify such models against safety and liveness requirements. Linear hybrid automata were further extended to *hybrid automata* where the continuous dynamics is governed by differential equations. Tools exist to model and analyze those systems, either directly or by approximating the model with timed automata or linear hybrid automata (see e.g. the survey paper [22]).

For MLD and PWA systems formulated in discrete time, several approaches have been proposed, we refer the reader to [11, 13, 14, 25, 26]. The toolbox uses a different approach based mixed-integer linear programming.

The problem is to test whether a certain terminal set $X_f = \{x : S_x x \leq T_x\}$ can be reached after exactly N steps, starting from a set of initial states $X_0 = \{x : S_0 x \leq T_0\}$ for some input sequence $u(0), \ldots, u(N-1)$ with $u_{\min} \leq u(k) \leq u_{\max}$, $\forall k = 0, \ldots, N-1$. Such a sequence exists if the following set of mixed-integer linear inequalities is feasible:

$$\left\{\begin{array}{rcl} S_0 x(0) & \leq & T_0 \\ x(k+1) & = & Ax(k) + B_1 u(k) + B_2 \delta(k) + B_3 z(k),\ k = 0, 1, \ldots, N-1 \\ E_2 \delta(k) & + & E_3 z(k) \leq E_1 u(k) + E_4 x(k) + E_5,\ k = 0, 1, \ldots, N-1 \\ u_{\min} & \leq & u(k) \leq u_{\max},\ k = 0, 1, \ldots, N-1 \\ S_x x(N) & \leq & T_x \end{array}\right. \tag{4.11}$$

with respect to the unknowns $u(0)$, $\delta(0)$, $z(0)$, …, $u(N-1)$, $\delta(N-1)$, $z(N-1)$, $x(0)$.

Complex conditions can be posed in the reachability analysis by adding constraints (such as linear constraints and Boolean constraints) in the MUST section of the HYSDEL file that defines the MLD model.

A more extended "safety" question is whether the state can never reach X_f at any time k, $1 \leq k \leq N$, starting from $X_0 = \{x : S_0 x \leq T_0\}$ and for some input sequence $u(0)$, …, $u(N-1)$, with $u_{\min} \leq u(k) \leq u_{\max}$, $\forall k = 0, \ldots, N-1$. Instead of solving this extended problem by solving the MILP (4.11) for all horizons between 1 and N, the Hybrid Toolbox provides the answer with one MILP, by introducing the auxiliary variable $\delta_f(k) \in \{0, 1\}$ subject to the constraint

$$[\delta_f(k) = 1] \longrightarrow [x(k+1) \in X_f] \tag{4.12}$$

and by imposing the constraint

$$\sum_{k=0}^{N-1} \delta_f(k) \geq 1.$$

The function `reach` performs the reachability analysis (4.11), as detailed in the following example.

4.5.1 Example

Consider the example in demo `reachtest.m`. The hybrid dynamics consists of three continuous states x_1, x_2, x_3, two binary states x_4, x_5, two continuous inputs u_1, u_2, one binary input u_3, and is defined by the equations

$$\begin{aligned}
x_1(k+1) &= \begin{cases} 0.1x_1(k) + 0.5x_2(k) & \text{if } (\delta_1(k) \wedge \delta_2(k)) \vee x_4(k) \text{ true} \\ -0.3x_3(k) - x_1(k) + u_1(k) & \text{otherwise} \end{cases} \\
x_2(k+1) &= \begin{cases} -0.8x_1(k) + 0.7x_3(k) - u_1(k) - u_2(k) & \text{if } \delta_3(k) \vee x_5(k) \text{ true} \\ -0.7x_1(k) - 2x_2(k) & \text{otherwise} \end{cases} \\
x_3(k+1) &= \begin{cases} -0.1x_3(k) + u_2(k) & \text{if } (\delta_3(k) \wedge x_5(k)) \vee (\delta_1(k) \wedge x_4(k)) \text{ true} \\ x_3(k) - 0.5x_1(k) - 2u_1(k) & \text{otherwise} \end{cases} \\
x_4(k+1) &= \delta_1(k) \wedge x_4(k) \\
x_5(k+1) &= ((x_4(k) \vee x_5(k)) \wedge (\delta_1(k) \vee \delta_2(k)) \vee (\delta_3(k) \wedge u_3(k)) \\
\delta_1(k) \text{ true} &\leftrightarrow [x_1(k) \leq 0] \\
\delta_2(k) \text{ true} &\leftrightarrow [x_2(k) \geq 0] \\
\delta_3(k) \text{ true} &\leftrightarrow [x_3(k) - x_2(k) \leq 1]
\end{aligned}$$

where $-10 \leq x_1 \leq 10$, $-5 \leq x_2 \leq 5$, $-20 \leq x_3 \leq 20$, $-1 \leq u_1 \leq 1$, $-2 \leq u_2 \leq 2$, as described in HYSDEL in `reachtest.hys`, and k is the time step index (sampling time=0.2 s).

We want to test if the set of states $X_f = \{x : -1 \leq x_{1,2,3} \leq 1,\ x_{4,5} \in \{0,1\}\}$ can be reached within $N = 5$ steps from an initial state $x(0)$ in the set $X_0 = \{x : -0.1 \leq x_i \leq 0.1,\ x_{4,5} \in \{0,1\}\}$ under the conditions

$$\begin{aligned}
&x_3(k) + x_2(k) \leq 0,\ \forall k = 0, \ldots, N-1 \\
&\delta_1(k) \vee \delta_2(k) \vee x_5(k) \text{ true},\ \forall k = 0, \ldots, N-1 \\
&\neg x_4(k) \vee x_5(k) \text{ true},\ \forall k = 0, \ldots, N-1.
\end{aligned}$$

The reachability analysis question is answered through the following code:

```
ts=0.2; % Sampling time
S=mld('reachtest',ts);
N=5;
Xf.A=[eye(5);-eye(5)];
Xf.b=[1 1 1 1 1 1 1 1 0 0]';
```

```
X0.A=[eye(5);-eye(5)];
X0.b=[.1 .1 .1 1 1 .1 .1 .1 0 0]';
[flag,x0,U,xf,X,T]=reach(S,[1 N],Xf,X0);
```

The answer is `flag=1`, which means that the set of states X_f is reachable from X_0 under the above dynamics and conditions. The code also returns an example of initial state $x(0) = x_0$, sequence of inputs $[u(0)\ u(1)\ \dots\ u(N-1)] = U$, and the corresponding final state $x(N) = x_f$ and state sequence $[x(0)\ x(1)\ \dots\ x(N-1)] = X$, satisfying the reachability analysis problem:

```
U =

   -0.0000   -1.0000    1.0000
   -0.6000   -1.1000    1.0000
    0.1714   -0.2714    1.0000
   -0.0800   -0.0171    1.0000
         0         0         0

X =

    0.0000   -0.0000         0    1.0000    1.0000
    0.0000    1.0000   -1.0000         0    1.0000
   -0.3000    1.0000   -1.0000         0    1.0000
    0.4700   -0.3600   -0.1714         0    1.0000
   -0.4986   -0.3989    0.0000         0    1.0000
    0.4986    0.3989   -0.0000         0    1.0000

xf =

    0.4986
    0.3989
   -0.0000
         0
    1.0000
```

In order to test whether the set of states $X_f = \{x : -1 \leq x_{1,2,3} \leq 1,\ x_{4,5} \in \{0, 1\}\}$ can be reached after exactly $N = 5$ steps, we can use the different syntax

```
[flag,x0,U,xf,X,T]=reach(S,N,Xf,X0);
```

Chapter 5

Simulink Library

This chapter describes the Simulink Library of the Hybrid Toolbox. The library offers blocks for

- Simulation of constrained controllers for linear systems (either based on on-line quadratic optimization or in explicit PWA form);
- Simulation of constrained controllers for hybrid systems (either based on on-line mixed-integer linear optimization or in explicit PWA form);
- Simulation of MLD and PWA systems.

The library is depicted in Figure 5.1. The leftmost blocks are controllers based on on-line optimization, the blocks in the middle are controllers based on explicit representations, the rightmost blocks are for simulation of MLD and PWA systems.

5.1 Constrained Control of Linear Systems

As described in Section 3.2, the receding horizon control problem is defined either a *regulation* or as a *tracking* problem. The Simulink block "Linear Constrained Controller" implements both a constrained regulator and a tracking controller based on on-line quadratic optimization. If a Kalman filter is associated with the controller, the block also internally estimates the state of the system to be controlled. In this case, the initial condition of the observer can be specified in the block mask, see Figure 5.2. In case the controller is for reference tracking, the value of the initial input $u(-1)$ can be also specified in the block mask.

5.1.1 Example: Aircraft Control

The folder `demos/linear` contains the demo `afti16.m` for the design of a controller for the AFTI-F16 aircraft [8, 19].

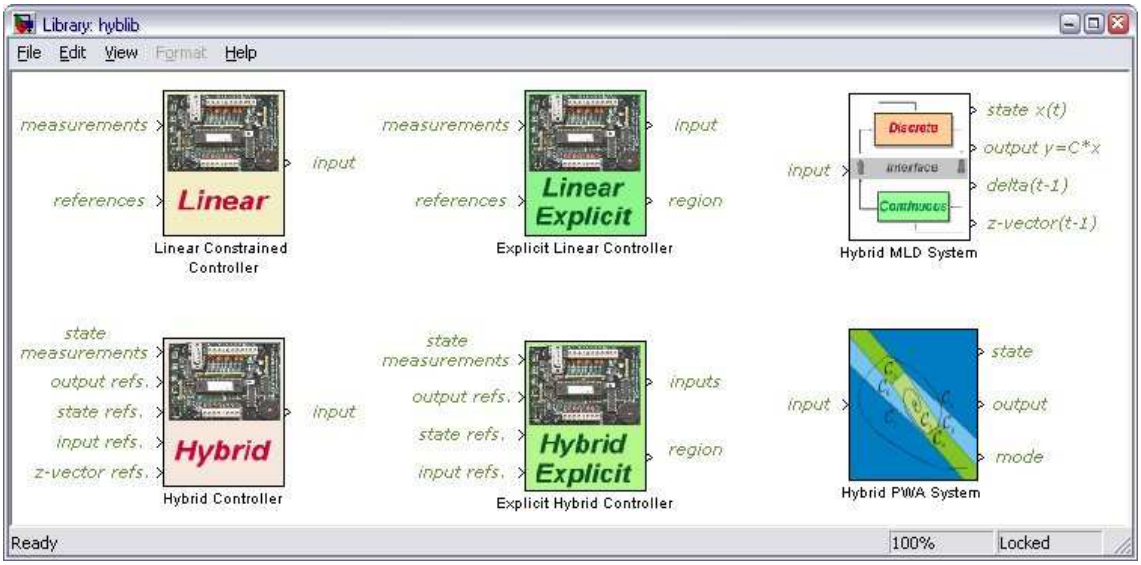


Figure 5.1: Hybrid Toolbox Simulink Library.

The linearized dynamic model for the attack and pitch angles as a function of the elevator and flaperon angles is

$$\begin{cases} \dot{x} = \begin{bmatrix} -.0151 & -60.5651 & 0 & -32.174 \\ -.0001 & -1.3411 & .9929 & 0 \\ .00018 & 43.2541 & -.86939 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} x + \\ \qquad \begin{bmatrix} -2.516 & -13.136 \\ -.1689 & -.2514 \\ -17.251 & -1.5766 \\ 0 & 0 \end{bmatrix} u \\ y = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x, \end{cases}$$

where u contains the elevator and flaperon angles, y the attack and pitch angles. The open-loop response of the system is unstable (open-loop poles: $-7.6636, -0.0075 \pm 0.0556j, 5.4530$). The model is converted to discrete-time with a sampling period $T_s = .05$ s. Both inputs are constrained between $\pm 25^\circ$. The task is to get zero offset for piecewise-constant references, avoiding instability due to input saturation.

The file `afti16.m` sets the model and the controller design up, determines the explicit form of the MPC controller, and simulate the closed loop.

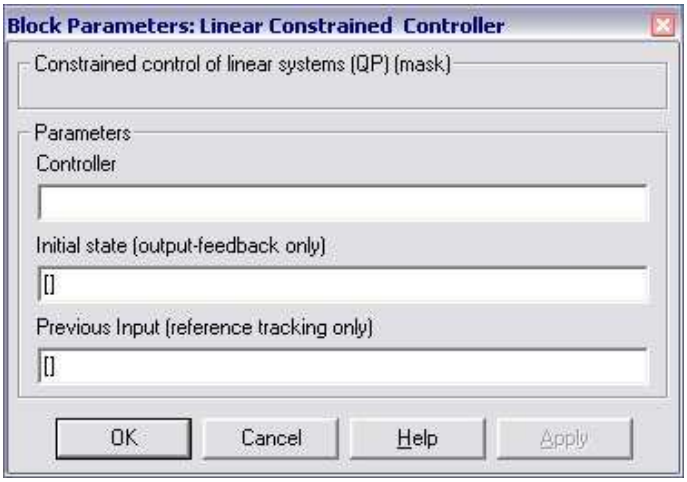


Figure 5.2: "Linear Constrained Controller": Simulink block mask.

Figure 5.3: AFTI-F16 aircraft

5.2 Constrained Control of Hybrid Systems

5.2.1 Example: Switching System

Consider again the example in demo `bm99sim.m`. We want to simulate now in Simulink the closed-loop system constituted by the hybrid system (4.3) and controller (4.8). The Simulink diagram `bm99mld.mdl`, represented in Figure 5.7 produces the plots shown in Figure 5.8.

The simulation needs to solve one MILP per sampling interval. On the other hand, we can repeat the simulation using the explicit version of the controller calculated in Section 4.4.1, as shown in the Simulink diagram `bm99exp` reported in Figure 5.9.

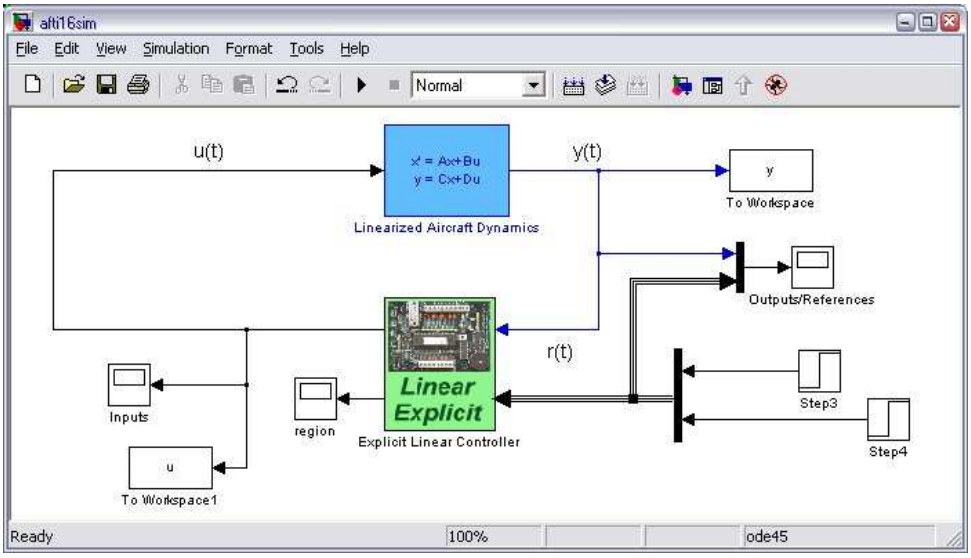
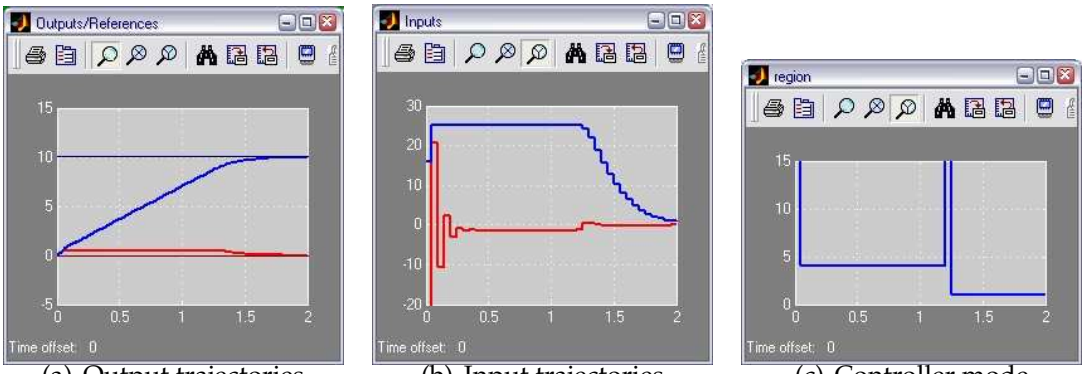


Figure 5.4: Simulation of the (linearized) dynamics of an AFTI-F16 aircraft in closed-loop with an explicit constrained linear controller.



(a) Output trajectories (b) Input trajectories (c) Controller mode

Figure 5.5: AFTI-F16 aircraft closed-loop trajectories

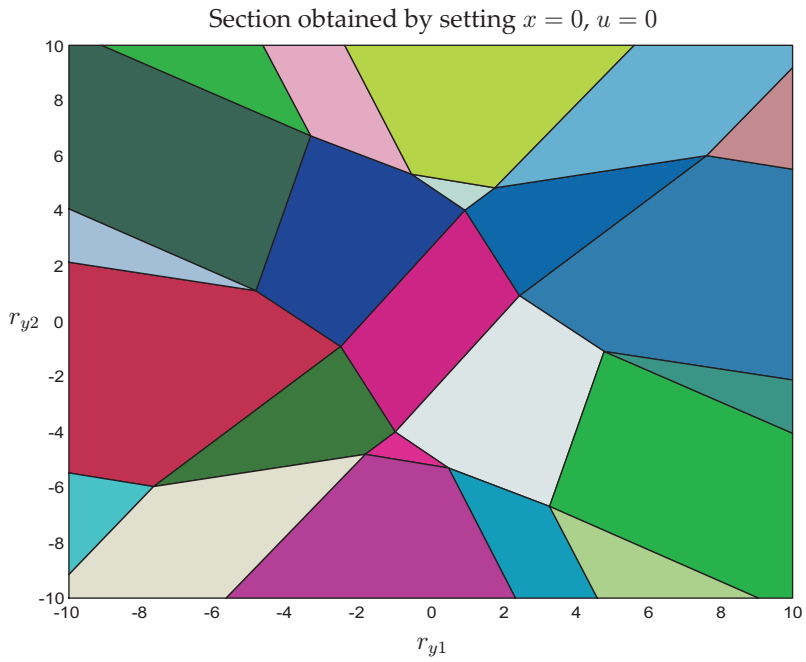


Figure 5.6: Partition associated with the explicit controller for the AFTI-F16 aircraft. Section in the output reference space, obtained by zeroing the current state x and previous input u.

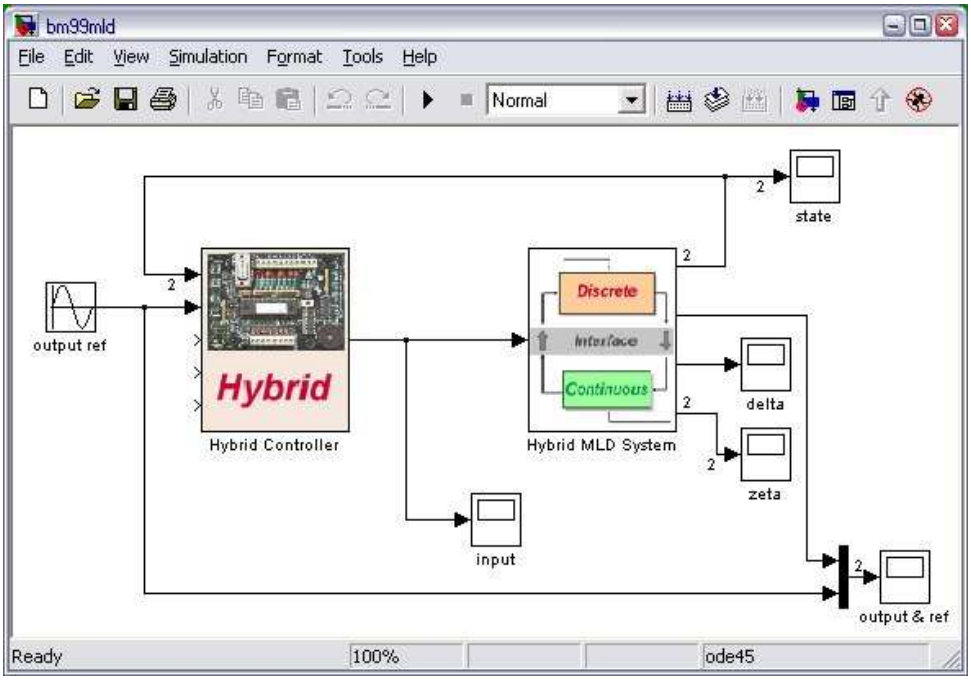
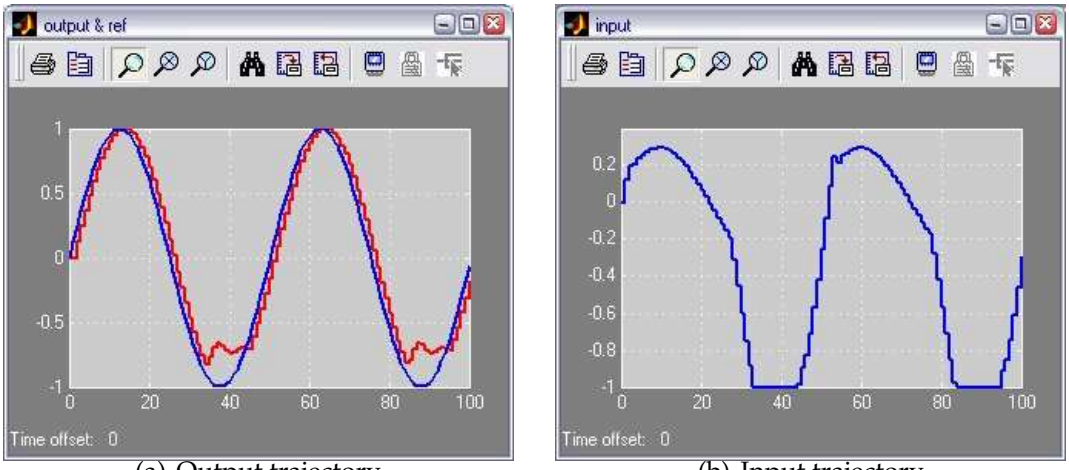


Figure 5.7: Closed-loop system constituted by the hybrid system (4.3) and controller (4.8) based on MILP.



(a) Output trajectory (b) Input trajectory

Figure 5.8: Closed-loop trajectories for the system constituted by the hybrid system (4.3) and controller (4.8) based on on-line MILP

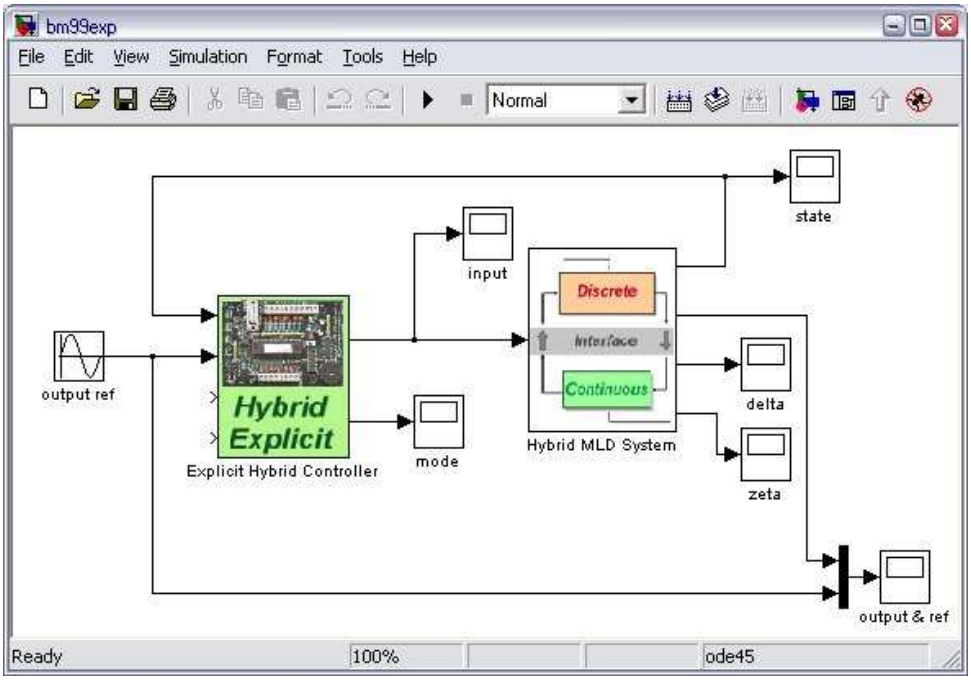


Figure 5.9: Closed-loop system constituted by the hybrid system (4.3) and the explicit version of controller (4.8), whose partition is depicted in Figure 4.4.

Chapter 6

Function Reference

The Hybrid Toolbox provides a set of Matlab functions for design, computation, and simulation of constrained controllers for linear and hybrid systems.

6.1 Functions by Category

6.1.1 MLD Systems

Methods for the `@mld` object (MLD hybrid system dynamics):

Function name	Description
`mld`	Construct an MLD system from a HYSDEL description
`update`	Update MLD dynamics
`sim`	Open-loop simulation
`reach`	Reachability analysis of MLD systems
`getfeasible`	Get a feasible state+input pair
`getbounds`	Get bounds on states and inputs
`info`	Get information from the symbolic table generated by HYSDEL
`getvar`	Get information about specific HYSDEL variables

6.1.2 PWA Systems

Methods for `@pwa` object (PWA hybrid system dynamics):

Function name	Description
`pwa`	Construct a PWA system from an MLD description
`update`	Update PWA dynamics
`sim`	Open-loop simulation
`plot`	Plot partition (only 2D)
`plotsection`	Plot sections of the partition
`fix`	Reduce PWA system by fixing input/states to given values
`pwaprops`	Help on PWA model properties

6.1.3 Constrained Optimal Control of Linear Systems

Methods for `@lincon` object (constrained control of linear systems):

Function name	Description
`lincon`	Controller based on quadratic optimization
`eval`	Evaluation of the control law using quadratic optimization
`kalman`	Design of a Kalman observer for output feedback
`setnames`	Assign input and output variable names
`sim`	Closed-loop simulation

6.1.4 Constrained Optimal Control of Hybrid Systems

Methods for `@hybcon` object (controller for hybrid systems):

Function name	Description
`hybcon`	Control of hybrid systems
`eval`	Evaluation of the hybrid control law based on MILP/MIQP
`sim`	Closed-loop simulation

6.1.5 Explicit Constrained Optimal Control of Linear and Hybrid Systems

Methods for the `@expcon` object (constrained control of linear and hybrid systems):

Function name	Description
`expcon`	Convert controller to explicit PWA form
`eval`	Evaluation of the explicit control law
`getgain`	Get the polyhedron and gain corresponding to a region number
`getregnum`	Pick up region numbers from 2D plots of controller partitions
`hwrite`	Convert PWA controller to C header file
`latex`	Convert PWA controller to LaTeX
`plot`	Plot partition (only 2D)
`plotsection`	Plot sections of the partition
`reduce`	Eliminate small regions from the partition
`kalman`	Design a Kalman observer for output feedback (only linear models)
`setnames`	Assign input and output variable names (only linear models)
`sim`	Closed-loop simulation

6.1.6 Polyhedral Computation and Partitions

Functions for polyhedral computation (contained in the directory `utils`):

Function name	Description
`polyincl`	Compute inclusion of polyhedra
`polyminus`	Compute difference of polyhedra
`polynormalize`	Normalize polyhedra inequalities
`polyplot`	Plot 2D polytopes
`polyplot3d`	Plot 3D polytopes
`lineplot`	Plot a line in a 2D figure
`polyreduce`	Reduce polyhedra to minimal hyperplane representation
`polyunion`	Compute the union of polyhedra
`joinconvex`	Join controller regions (if the union is a convex set)
`reduce`	reduce piecewise affine solutions by eliminating small regions
`pwaplot`	Plot polyhedral partitions
`pwaplotsection`	Plot sections of polyhedral partitions

6.1.7 Observer Design

Auxiliary functions for observer design (contained in the directory `utils`):

Function name	Description
`kalmdesign`	Design Kalman filter for `@lincon` and linear `@expcon` objects
`kalmanhelp`	Help on KALMDESIGN

6.1.8 C-Code, MEX Functions, S-Functions

C-code for evaluation of explicit control for linear and hybrid systems, MEX interfaces, S-functions for explicit controller evaluation and for simulation of hybrid MLD and PWA dynamics (located in the directory `utils`):

Function name	Description
`expcon.c`	C file for computing an explicit controller
`expcon.h`	Header file (generated by `expcon/hwrite`)
`linobsmex.c`	MEX-C file for computing an explicit controller for linear output-feedback systems
`linobsmex`	Help on `linobsmex.c`
`expconmex.c`	MEX-C file for computing an explicit linear/hybrid state-feedback controller
`expconmex`	Help on `expconmex.c`
`expsfun.c`	MEX-C S-function for simulating an explicit controller in Simulink
`expsfun.h`	Header file (generated by `expcon/hwrite`)
`hybsfun`	S-function for simulating hybrid controllers based on MILP in Simulink
`linsfun`	S-function for simulating constrained linear controller based on QP in Simulink
`mldsfun`	S-function for simulating MLD models
`pwasfun`	S-function for simulating PWA models

6.1.9 Solvers

Solvers for LP, QP, MILP, mpLP, mpQP, comparison of PWA functions (contained in the directory `utils`):

Function name	Description
cddmex	LP solver and polyhedral computation (by K. Fukuda)
cplexmex	Detailed help on CPLEXMEX (by N. Giorgetti)
glpkmex	LP/MILP solver (by A. Makhorin, Maltab interface by N. Giorgetti)
glpkparams	Detailed help on GLPKMEX (by N. Giorgetti)
lpsol	Interface to different LP solvers
lptype	Selection of preferred LP solver
mexpress	Detailed help on MEXPRESS (by N. Giorgetti)
mld2pwa	MLD to PWA conversion algorithm
milpsol	Interface to different MILP solvers
milptype	Selection of preferred MILP solver
minpwa	Compute the minimum of several convex PWA functions
miqpsol	Interface to different MIQP solvers
miqptype	Selection of preferred MIQP solver
miqp	Solve MILP/MIQP problems (Matlab based)
miqp3_naf	Another MILP/MIQP solver (Matlab based)
mplp	multiparametric linear programming solver
mpqp	multiparametric quadratic programming solver
mplpjoin	reduce mplp solutions
pwaopt	Compute explicit representation of hybrid optimal controllers
qpsol	Interface to different QP solvers
qptype	Selection of preferred QP solver
qpact	QP solver based on an active-set method

6.2 @LINCON – Controllers for Constrained Linear Systems

Controllers for constrained linear systems based on quadratic optimization are described by `@LINCON` objects. The constructor function `lincon` has the following syntax:

Syntax: `C=lincon(SYS,TYPE,COST,INTERVAL,LIMITS)`
Given the linear model `SYS`, if `TYPE='reg'` `lincon` builds a constrained regulator to the origin based on the optimal control problem (3.3)

$$
\begin{array}{ll}
\min & x'(t+T|t)Px(t+N|t) + \sum_{k=0}^{N-1} x'(t+k|t)Qx(t+k|t) + u'(t+k)Ru(t+k) + \rho\varepsilon^2 \\
\text{s.t.} & y_{\min} - \varepsilon \leq y(t+k|t) \leq y_{\max} + \varepsilon,\ k = 1, ..., N_{cy} \\
& u_{\min} \leq u(t+k) \leq u_{\max},\ k = 0, ..., N_{cu} \\
& u(t+k) = Kx(t+k|t),\ k \geq N_u \\
& x(t+k+1|t) = Ax(t+k|t) + Bu(t+k) \\
& y(t+k|t) = Cx(t+k|t) + Du(t+k)
\end{array}
$$

or, if `TYPE='track'`, it builds a constrained controller for output reference track-

ing based on the optimal control problem (3.6)

$$\begin{array}{ll}\min & \sum_{k=0}^{N-1}[y'(t+k|t)-r(t)]S[y(t+k|t)-r(t)]+\Delta u'(t+k)T\Delta u(t+k)+\rho\varepsilon^2\\ \text{s.t.} & y_{\min}-\varepsilon\le y(t+k|t)\le y_{\max}+\varepsilon,\ k=1,...,N_{cy}\\ & u_{\min}\le u(t+k)\le u_{\max},\ k=0,...,N_{cu}\\ & \Delta u_{\min}\le \Delta u(t+k)\le \Delta u_{\max},\ k=0,...,N_{cu}\\ & u(t+k)=0,\ k\ge N_u\\ & x(t+k+1|t)=Ax(t+k|t)+B[u(t+k-1|t)+\Delta u(t+k)]\\ & y(t+k|t)=Cx(t+k|t)+D[u(t+k-1|t)+\Delta u(t+k)]\end{array}$$

`COST` is a structure defining the parameters of the optimal control problem and has the following fields:

Property	Description		Type	Default
`Q`	state weight	$x'(k)Qx(k)$	`reg`	I
`R`	input weight	$u'(k)Ru(k)$	`reg`	$\frac{1}{10}I$
`P`	final state weight	$x'(N)Px(N)$	`reg`	`'lqr'`
`S`	output weight	$(y(k)-r)'S(y(k)-r)$	`track`	I
`T`	input increment weight	$\Delta u'(k)T\Delta u(k)$	`track`	$\frac{1}{10}I$
`rho`	slack var weight	$\rho\varepsilon^2$	`reg`, `track`	10^4 or `Inf`
`K`	feedback gain	$u(k)=Kx(k)$ for $k\ge N_u$	`reg`	LQ gain

For regulators to the origin, `COST.P` must be either a matrix of weights or a string. An accepted option is `COST.P='lqr'`, which forces the terminal weight P to be the solution of the LQR problem with weights Q, R, and K to be the corresponding LQR gain (cf. Eq. (3.4)). Another accepted option is `COST.P='lyap'`, which forces the terminal weight P to be the solution of the Lyapunov equation (3.5) and $K=0$. By default, `cost.rho`=10^4; in case the controller has no output constraints, however, `cost.rho` is always set to `Inf`, even if a finite value of `cost.rho` has been specified, as input constraints only never lead to unfeasibility of the QP problem. Note: the code building the QP problem scales the cost function matrices by dividing the cost function by the maximum singular value of the Hessian matrix. This is done *before* adding soft constraints, and hence, `cost.rho` is not scaled.

The structure `INTERVALS` defines the number of input and output optimal control steps. `INTERVALS` has the following fields:

Field name	Description	Default
`.N`	optimal control interval over which the cost function is summed	N_u
`.Nu`	number of free optimal control moves $u(0),\dots,u(N_u-1)$	2
`.Ncy`	output constraints are checked up to time $k=N_{cy}$	$N-1$
`.Ncu`	input constraints are checked up to time $k=N_{cu}$	N_u-1

The structure `LIMITS` defines upper and lower bounds on output, state, and input variables. `LIMITS` has the following fields:

Field name	Description		Type
.umin	lower bounds on inputs	$u(k) \geq u_{\min}$	reg, track
.umax	upper bounds on inputs	$u(k) \leq u_{\max}$	reg, track
.dumin	lower bounds on input increments	$\Delta u(k) \geq \Delta u_{\min}$	track
.dumax	upper bounds on input increments	$\Delta u(k) \leq \Delta u_{\max}$	track
.ymin	lower bounds on outputs	$y(k) \geq y_{\min} - \varepsilon$	reg, track
.ymax	upper bounds on outputs	$y(k) \leq y_{\max} + \varepsilon$	reg, track

Default values for limits are $\pm\infty$. Limits are treated as hard constraints if `COST.rho=+Inf`, as soft constraints otherwise.

The output argument is a `@LINCON` object which collects the matrices of the quadratic program (cf. Eq. (3.7)) corresponding to the posed optimal control problem. The quadratic program has the form:

$$\begin{array}{ll} \min & \frac{1}{2} z'Qz + \theta'C'z \\ \text{s.t.} & Gz \leq W + S\theta \end{array}$$

Correspondingly, the output argument `C` is a `@LINCON` object with the following properties:

Property	Description
Q	Hessian matrix
C	linear cost matrix
G	lhs constraint matrix
W	rhs constraint vector
S	rhs constraint matrix
model	LTI model
nx	number of states
nu	number of inputs
ny	number of outputs
type	controller type
ts	sampling time
isconstr	controller constrained/unconstrained
soft	soft constraints/hard constraints
nvar	number of optimization variables
nq	number of linear inequality constraints
npar	number of parameters (states and references)
QPsolver	QP solver
I1	matrix to extract u(0) from the optimal vector
Qinv	inverse of Hessian matrix
Observer	observer information

For output-feedback controllers, `C.Observer` contains the observer information. See Section 6.2.1 below.

Syntax: `C=lincon(SYS,TYPE,COST,INTERVAL,LIMITS,QPSOLVER)`
also specifies the type of QP solver to be used for computations. See Appendix B.2 for supported QP solvers. The default QP solver is the active set algorithm `qpact`.

6.2.1 Output-Feedback Control

Output-feedback controllers for constrained linear systems can be obtained by appending a state observer to the controller object (either a `@LINCON` or an `@EXPCON` object for linear systems). The method `kalman` allows one to associate a Kalman filter to the controller and has the following syntax:

Syntax: `KALMAN(CON,Q,R)`
designs a state observer for the linear model (A, B, C, D)=`CON.model` which controller `CON` is based on, using Kalman filtering techniques. The resulting observer is stored in the `'Observer'` field of the controller object `CON`. The controller object can be either of class `@LINCON` or `@EXPCON` (explicit controller for linear systems).

`Q` is the covariance matrix of state noise, `R` is the covariance matrix of output noise.

Syntax: `Kest=KALMAN(CON,Q,R)`
also returns the state observer as an LTI object. `Kest` receives $u(k)$ and $y(k)$ as inputs (in this order), and provides the best estimated state $x(k|k-1)$ of $x(k)$ given the past measurements $y(k-1)$, $y(k-2)$, ...

$$x(k+1|k) = Ax(k|k-1) + Bu(k) + L(y(k) - Cx(k|k-1) - Du(k)$$

Syntax: `Kest=KALMAN(CON,Q,R,ymeasured)`
assumes that only the outputs specified in vector `ymeasured` are measurable outputs. In this case, `R` is a n_{ym}-by-n_{ym} matrix, where n_{ym} is the number of measured outputs.

Syntax: `[Kest,M]=KALMAN(CON,Q,R,ymeasured)`
also returns the gain M which allows computing the measurement update

$$x(k|k) = x(k|k-1) + M(y(k) - Cx(k|k-1) - Du(k)$$

See also the command `KALMAN` in the Control Systems Toolbox. An example is contained in the demo `dcmotor` in the `demos/linear` directory.

6.3 @HYBCON – Controllers for Hybrid Systems

Controllers for hybrid systems based on mixed-integer linear optimization are described by `@HYBCON` objects. The constructor function `hybcon` has the following syntax:

Syntax: `P=hybcon(MLD,Q,N)`
Builds a hybrid controller based on the optimal control problem (4.5)–(4.7). The input arguments are the MLD system `MLD`, typically obtained from a HYSDEL file

through the constructor `mld`, the structure `Q` of weights

$$\begin{array}{rcll} \texttt{Q.y} & = & Q_y, & \text{(default: } Q_y = I\text{)} \\ \texttt{Q.u} & = & Q_u, & \text{(default: } Q_u = 0.1I\text{)} \\ \texttt{Q.x} & = & Q_x, & \text{(default: } Q_x = I\text{)} \\ \texttt{Q.z} & = & Q_z, & \text{(default: } Q_z = 0\text{)} \\ \texttt{Q.xN} & = & Q_{xN}, & \text{(default: } Q_{xN} = Q_x\text{)} \\ \texttt{Q.rho} & = & Q_\rho, & \text{(default: } Q_\rho = +\infty\text{),} \end{array}$$

the control horizon `N` (default: `N=1`). As an output, the `@HYBCON` object contains the matrices of the MILP problem

$$\begin{aligned} \min \quad & f'q \\ \text{s.t.} \quad & Aq \leq b + C_x x(t) + C_r^y r_y(t) + C_r^u r_u(t) + C_z^y r_z(t) \end{aligned} \tag{6.1}$$

where q collects slack variables and the sequence of future u, δ, and z variables, or the MIQP problem

$$\begin{aligned} \min \quad & \tfrac{1}{2}q'Hq + \theta'(t)Dq + f'q + \tfrac{1}{2}\theta(t)'Y\theta(t) + V'\theta(t) + d \\ \text{s.t.} \quad & Aq \leq b + C_x x(t) + C_r^y r_y(t) + C_r^u r_u(t) + C_z^y r_z(t) \end{aligned} \tag{6.2}$$

associated with the control law, where $\theta = [x(t), r_x(t), r_y(t), r_u(t), r_z(t)]'$ and q collects future u, δ, and z variables. The main fields of object `@HYBCON` are the following:

Property	Description
`f`	linear cost
`H`	Hessian matrix (only 2-norm)
`D`	parameter linear cost (only 2-norm)
`Y`	quadratic term (only 2-norm)
`A`	constraint matrix
`b`	constraint constant vector
`Cx`	constraint matrix for state vector x(t)
`Cr`	constraint matrix for references r(t):
`Cr.y`	constraint matrix for input reference vector $r_y(t)$
`Cr.u`	constraint matrix for input reference vector $r_u(t)$
`Cr.x`	constraint matrix for state reference vector $r_x(t)$
`Cr.z`	constraint matrix for z-reference vector $r_z(t)$
`uvar`	position of $u(0)$,...,$u(N-1)$ within vector q
`dvar`	position of $\delta(0)$,...,$\delta(N-1)$ within vector q
`zvar`	position of $z(0)$,...,$z(N-1)$ within vector q
`ivar`	position of integer variables within the optimization vector q (binary inputs and binary auxiliary variable δ)
`horizon`	prediction horizon
`model`	name of MLD variable which the controller is based on
`name`	name of HYSDEL model which generated the MLD model
`ts`	sampling time of the controller (inherited from `MLD`'s sampling time)

Syntax: `P=hybcon(MLD,Q,N,LIMITS)`
also specifies the structure `LIMITS` of upper and lower bounds on output, state, and input variables. `LIMITS` has the following fields:

Field name	Description	
`.umin`	lower bounds on inputs	$[\ u(k) \geq u_{\min}\ ]$
`.umax`	upper bounds on inputs	$[\ u(k) \leq u_{\max}\ ]$
`.xmin`	lower bounds on states	$[\ x(k) \geq x_{\min} - \rho\ ]$
`.xmax`	upper bounds on states	$[\ x(k) \leq x_{\max} + \rho\ ]$
`.ymin`	lower bounds on outputs	$[\ y(k) \geq y_{\min} - \rho\ ]$
`.ymax`	upper bounds on outputs	$[\ y(k) \leq y_{\max} + \rho\ ]$

Limits are treated as hard constraints if `Q.rho=+Inf`, otherwise as soft constraints.

Syntax: `P=hybcon(MLD,Q,N,LIMITS,REFSIGNALS)`
also specifies the structure `REFSIGNALS` denoting output/state/input/z variables for which a reference is specified, and has the following fields

Field name	Description	Default
`.y`	outputs having a reference signal	`[1:ny]`
`.u`	inputs having a reference signal	`[]`
`.x`	states having a reference signal	`[]`
`.z`	z-vectors having a reference signal	`[]`

NOTE: The weight matrices specified in the structure `Q` must have a dimension equal to the number of tracked signals, i.e., `Q.y` must have dimension = `length(REFSIGNALS.y)`, `Q.u` must have dimension = `length(REFSIGNALS.u)`, `Q.x` must have dimension = `length(REFSIGNALS.x)`, `Q.z` must have dimension = `length(REFSIGNALS.z)`, `Q.xN` must have dimension = `length(REFSIGNALS.x)`.
Example: The MLD system has 3 outputs and 2 inputs, and we want only outputs y_1 and y_3 to track certain reference signals r_{y1}, r_{y3} by minimizing $\| \begin{bmatrix} 2(y_1 - r_{y1}) \\ 5(y_3 - r_{y3}) \end{bmatrix} \|$, and no reference trajectories for inputs, states, and z-variables. We must set `REFSIGNALS.y=[1 3]`, `REFSIGNALS.u=[]`, `REFSIGNALS.x=[]`, `REFSIGNALS.z=[]`, and `Q.y=[2 0;0 5]`, `Q.u=[]`, `Q.x=[]`, `Q.z=[]`, `Q.xN=[]`. `Q.rho` is always a scalar.

Syntax: `P=hybcon(MLD,Q,N,LIMITS,REFSIGNALS,mipsolver)`
also specifies the type of MIP solver to be used for computations. See Appendices B.3, B.4 for supported MILP/MIQP solvers. The default MIP solver is the one specified in `MLD.mipsolver`.

6.4 @EXPCON – Explicit Controllers (Linear/Hybrid Systems)

Explicit controllers for linear or hybrid systems are described by `@EXPCON` objects. The constructor function `expcon` has the following syntax:

Syntax: `E=expcon(C,RANGE)`

Converts controller `C` to piecewise affine explicit form. `C` must be a constrained optimal controller for linear systems (`@lincon` object), or for hybrid systems (`@hybcon` object), or an MPC controller defined through the Model Predictive Control Toolbox for Matlab v2.0 (`@mpc` object).

The optional input argument `RANGE` is a structure defining the range of initial states and references (i.e., the parameters of the multiparametric program) for which the explicit solution is computed. `RANGE` has the following fields:

Property	**Range**		**Model**
`xmin`, `xmax`	states (for MPC objects, x includes states of plant and) disturbance models)	`xmin`$\leq x(t) \leq$`xmax`	linear and hybrid
`umin`, `umax`	inputs	`umin`$\leq u(t-1) \leq$`umax`	linear w/tracking
`refymin`, `refymax`	output refs.	`refymin`$\leq r_y(t) \leq$`refymax`	linear w/tracking and hybrid
`refxmin`, `refxmax`	state refs.	`refxmin`$\leq r_x(t) \leq$`refxmax`	hybrid
`refumin`, `refumax`	input refs.	`refumin`$\leq r_u(t) \leq$`refumax`	hybrid
`vmin`, `vmax`	measured disturbances	`vmin`$\leq v(t) \leq$`vmax`	MPC objects

For hybrid controllers, `refxmin`, `refxmax`, `refumin`, `refumax`, `refymin`, `refymax` must have dimensions consistent with the number of indices specified in `C.refsignals` (see Section 6.3).

Example: Consider again the example of Section 6.3, where we have an MLD system with 3 outputs and 2 inputs, and we want only outputs y_1 and y_3 to track certain reference signals r_{y1}, r_{y3} and no reference trajectories for inputs, states, and z-variables. Assume the state vector x has 4 components, and that we want to compute the explicit controller for the range $-10 \leq x_i \leq 10$, $i = 1, 2, 3, 4$, $-1 \leq r_{y1} \leq 1$, $-2 \leq r_{y3} \leq 2$. We must set `xmin=[-10 -10 -10 -10]`, `xmax=[10 10 10 10]`, `refymin=[-1 -2]`, `refymax=[1 2]`. The default value for unspecified ranges is $\pm 10^4$.

Note that the defined range may be different from the limits over variables imposed in the control law.

Syntax: `E=expcon(C,RANGE,OPTIONS)`
also specifies the structure `OPTIONS` defining various options for computing the explicit control law. `OPTIONS` has the following fields:

Field name	Description	Default
`.lpsolver`	LP solver (cf. Sect. B.1)	`'glpk'`
`.qpsolver`	QP solver (cf. Sect. B.2)	`'qpact'`
`.fixref`	structure with fields `'y'`, `'x'`, `'u'` defining references that are fixed at given values	`[]`
`.valueref`	structure with fields `'y'`, `'x'`, `'u'` of values at which references are fixed	`[]`
`.flattol`	tolerance for a polyhedral set to be considered flat	`1e-6`
`.waitbar`	display waitbar (only for hybrid)	`1`
`.verbose`	level of verbosity 0,1,2 of mp-PWA solver	`0`
`.mplpverbose`	level of verbosity 0,1,2 of mp-LP solver	`1`
`.uniteeps`	tolerance for judging convexity of the union of polyhedra	`1e-3`
`.join`	flag for reducing the complexity by joining regions whose union is a convex set	`1`
`.reltol`	tolerance used for several polyhedral operations	`1e-6`

The options `options.fixref` and `options.valueref`, only available for controllers of hybrid systems, are fundamental instruments for simplifying the multiparametric solution, as they allow reducing the dimension of the parameter space.

Example: Consider again the example of Section 6.3 and assume that we want to obtain a multiparametric solution only with respect to $r_{y1}(t)$, $x(t)$ for $r_{y3} = 0.5$. We must set `options.fixref.y=3`, `options.valueref.y=0.5`.

Explicit controllers based on linear models can be converted to output-feedback controllers by using a state observer, see Section 6.2.1.

Chapter 7

C-Code Generation

The Hybrid Toolbox is able to generate C-code for the explicit optimal controller. The core C-function is `expcon.c`, which computes the explicit optimal control action associated with the PWA mapping. `expcon.c` is a very simple function (about 30 lines of C code), for the evaluation of a piecewise map. In addition, for output-feedback constrained controllers for linear systems, `expcon.c` estimates the state vector. `expcon.c` requires the header file `expcon.h`. This contains all the parameters defining the explicit controller, and is generated by the method `hwrite` for `@expcon` objects. Note that the combined control law `expcon.c` + `expcon.h` can be thought as the implementation of a look-up table of linear (affine) control laws.

7.1 Mex Interfaces

The toolbox also provides the C-function `expconmex.c` for generating a mex file of a state-feedback explicit controller. `expconmex.c` includes `expmpc.c` and `expmpc.h`, and provides a control action equivalent to a state-feedback `@lincon/eval` or `@hybcon/eval` controller. For output-feedback explicit controllers for linear systems, the mex C-function `linobsmex.c` provides the control action equivalent to `@lincon/eval`.

7.2 HWRITE

Syntax: `hwrite(C)`
Writes the header file `expcon.h` for the state-feedback or output-feedback explicit controller `C`. This must be a valid `@expcon` object.

Syntax: `hwrite(C,zerotol)`
also specifies a tolerance for considering small numbers as true zeros.

Syntax: `hwrite(C,zerotol,type)`
also specifies the type (`'int'`, `'float'`, or `'double'`) for storing the parameters defining the polyhedral cells of the solution.

In addition, the following syntax is valid for output-feedback controllers based on linear models:

Syntax: `hwrite(C,zerotol,type,u1)`
also specifies the previous input $u(-1)$ at time t=-1. This is only meaningful for tracking controllers.

Syntax: `hwrite(C,zerotol,type,u1,x0)`
also specifies the initial condition for the state observer.

Contrarily to `expmpc.c`, the header file `expmpc.h` depends on the values of the explicit controller `expcon`.

The header file is saved in the `utils/` directory. Make sure that the `utils/` directory has write permissions and that the `.h` and `.dll` files are not read-only.

An example of header file automatically generated for the explicit controller in the example reported in Section 3.1 (see Figure 3.1) is reported here below.

```
#define EXPCON_CONSTRAINED
#define EXPCON_REGULATION
#define EXPCON_LINEAR_MODEL
#define EXPCON_NU 1
#define EXPCON_NX 2
#define EXPCON_NY 2
#define EXPCON_TS 1.00000000
#define EXPCON_REG 7
#define EXPCON_NTH 2
#define EXPCON_NH 20
#define EXPCON_NF 7
#define EXPCON_NYM 2
static double EXPCON_F[]={
    -0.816617,0,0,0,-0.552806,-0.552806,0,-1.74993,0,0,0,
    -1.53639,-1.53639,0};

static double EXPCON_G[]={
    0,-1,-1,1,-0.430778,0.430778,1};

static double EXPCON_H[]={
    -0.816617,0.816617,0.612406,-0.612406,0.296959,-0.816617,
    -0.97116,-0.386367,-0.296959,0.386367,0.296959,-0.386367,
    0.97116,-0.612406,-0.97116,0.386367,0.612406,-0.296959,
    0.816617,0.97116,-1.74993,1.74993,0.49571,-0.49571,
    0.933314,-1.74993,-2.6991,-1.07381,
    -0.933314,1.07381,0.933314,-1.07381,
    2.6991,-0.49571,-2.6991,1.07381,
    0.49571,-0.933314,1.74993,2.6991
    };

static double EXPCON_K[]={
    1,1,1,1,1,-1,-1,-1,
    -1,-1,-1,1,1,-1,1,1,
    -1,1,-1,-1};

static int EXPCON_len[]={
    4,3,2,2,3,3,3};
```

Note: `hwrite` does not handle explicit hybrid controllers based on quadratic penalties in the current version of the Hybrid Toolbox.

Chapter 8

Complexity and Tuning Guidelines

8.1 Complexity

The number n_r of regions in the solution to the multiparametric Quadratic Programming (mpQP) problem

$$\begin{array}{ll} \min & \frac{1}{2}z'Qz + \theta'C'z \\ \text{s.t.} & Gz \leq W + S\theta \end{array} \tag{8.1}$$

depends on the dimension n of the parameter vector $\theta \in \mathbb{R}^m$ (i.e., the number of states, reference signals, previous control inputs, measured disturbances), on the dimension of the free vector $z \in \mathbb{R}^n$ (i.e., the number of free inputs in the optimal control problem), the number of constraints q in the optimization problem (8.1), and the size of the range $\theta_{\min} \leq \theta \leq \theta_{\max}$ where the mpQP problem is solved. In (D.1), $Q \in \mathbb{R}^{n\times n}$, $C \in \mathbb{R}^{n\times m}$, $G \in \mathbb{R}^{q\times n}$, $W \in \mathbb{R}^q$, $S \in \mathbb{R}^{q\times m}$, where q is the number of constraints.

As the number of combinations of ℓ constraints out of a set of q is $\binom{q}{\ell} = \frac{q!}{(q-\ell)!\ell!}$, the number of possible combinations of active constraints at the solution of a QP is at most $\sum_{\ell=0}^{q}\binom{q}{\ell} = 2^q$. This number represents an upper-bound on the number of different linear feedback gains which describe the controller. In practice, far fewer combinations are usually generated as θ spans the box $[\theta_{\min}, \theta_{\max}]$. Furthermore, the gains for the future input moves $u(t+1), \ldots, u(t+m-1)$ are not relevant for the control law. Thus several different combinations of active constraints may lead to the same first components $u(t)$ of the solution . On the other hand, the number n_r of regions of the piecewise affine solution is in general larger than the number of feedback gains, because possible nonconvex critical regions may be split into several convex sets.

8.1.1 Dependence on the Number of Parameters

Let $q_s \triangleq \operatorname{rank} S$, $q_s \leq q$, $\theta \in \mathbb{R}^m$. For $m > q_s$ the number of polyhedral regions n_r remains constant. To see this, consider the linear transformation $\bar{\theta} = S\theta$, $\bar{\theta} \in \mathbb{R}^q$. Clearly $\bar{\theta}$ and θ define the same set of active constraints, and therefore the

number of partitions in the $\bar{\theta}$- and θ-space are the same. Therefore, the number of partitions n_r of the θ-space defining the optimal controller is insensitive to the dimension m of the parameter vector θ for all $m \geq q_s$, i.e. to the number of parameters involved in the mp-QP.

In particular, the additional parameters needed to extend MPC to reference tracking, disturbance rejection, soft constraints, do not affect significantly the number of polyhedral regions n_r (i.e., the complexity of the mp-QP), and hence the number of regions in the explicit MPC controller.

With respect to the size of the range $\theta_{\min} \leq \theta \leq \theta_{\max}$ where the mpQP problem is solved, the larger the range the larger may be the number of polyhedral cells in the partition.

8.1.2 Dependence on the Input and Output Horizon

The larger the number of constraints q, the larger the number n_r of regions in the solution, and the dependence is exponential.

The number q of constraints increases with the length of the control horizon (multiplied by the number of constrained control inputs) for input constraints. Fortunately, many MPC control problems involve input constraints only, and typically horizons of $2, 3$ steps or blocking of control moves are adopted, which reduces the number of constraints q and therefore the number of regions in the explicit MPC controller.

For output constraints, the number q of constraints increases also with the length p of the prediction horizon (multiplied by the number of constrained outputs). One way to reduce the complexity n_r of the explicit controller is to constrain only a limited number of outputs over the prediction horizon. Say $N = 20$ and you want to constraint $y(t+1|t) \geq -1$, $y(t+2|t) \geq -1$. This can be achieved by setting `ymin=-1` and $N_{cy} = 1$. The Model Predictive Control Toolbox offers more flexibility for this, as time-varying constraints are allowed.

8.2 Tuning Guidelines

The following points should be kept in mind while tuning an MPC controller:

1. For a long enough prediction horizon N, when the constraints are not active the MPC controller performs as LQ control, assuming that `P='lqr'`. Therefore, the same noise rejection/bandwidth/sensitivity considerations apply: the larger the ratio between output weight and input weight, the more aggressive the controller, but the larger the sensitivity to noise. Short prediction horizons N also make the controller more aggressive.

2. Choice of the input horizon N_u: the larger, the better the performance, because the number of degrees of freedom is larger. On the other hand, the complexity increases with N_u. Always choose N_u as small as possible, by progressively reducing it until performance remains acceptable.

3. First tune an *implicit* MPC controller (either using `lincon` or `hybcon`). When satisfied with the design, compute the *explicit* form using `expcon`. If you get too many regions, revise the MPC design.

4. Be thrifty with constraints. They may provide an excellent design, but later generate too many regions in the explicit MPC partition.

For explicit controllers for hybrid systems, the complexity also depends on the number of binary states, inputs and auxiliary variables involved in the dynamics. The use of long optimization horizons may therefore lead to very complex partitions.

Bibliography

[1] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In A.P. Ravn R.L. Grossman, A. Nerode and H. Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer-Verlag, 1993.

[2] P.J. Antsaklis. A brief introduction to the theory and applications of hybrid systems. *Proc. IEEE, Special Issue on Hybrid Systems: Theory and Applications*, 88(7):879–886, July 2000.

[3] A. Bemporad. Efficient conversion of mixed logical dynamical systems into an equivalent piecewise affine form. *IEEE Trans. Automatic Control*, 49(5):832–838, 2004.

[4] A. Bemporad, P. Borodani, and M. Mannelli. Hybrid control of an automotive robotized gearbox for reduction of consumptions and emissions. In O. Maler and A. Pnueli, editors, *Hybrid Systems: Computation and Control*, number 2623 in Lecture Notes in Computer Science, pages 81–96. Springer-Verlag, 2003.

[5] A. Bemporad, F. Borrelli, and M. Morari. Optimal controllers for hybrid systems: Stability and piecewise linear explicit form. In *Proc. 39th IEEE Conf. on Decision and Control*, pages 1810–1815, Sydney, Australia, December 2000.

[6] A. Bemporad, F. Borrelli, and M. Morari. Piecewise linear optimal controllers for hybrid systems. In *Proc. American Contr. Conf.*, pages 1190–1194, Chicago, IL, June 2000.

[7] A. Bemporad, F. Borrelli, and M. Morari. On the optimal control law for linear discrete time hybrid systems. In M. Greenstreet and C. Tomlin, editors, *Hybrid Systems: Computation and Control*, number 2289 in Lecture Notes in Computer Science, pages 105–119. Springer-Verlag, 2002.

[8] A. Bemporad, A. Casavola, and E. Mosca. Nonlinear control of constrained linear systems via predictive reference management. *IEEE Trans. Automatic Control*, AC-42(3):340–349, 1997.

[9] A. Bemporad, N. Giorgetti, I.V. Kolmanovsky, and D. Hrovat. A hybrid system approach to modeling and optimal control of DISC engines. In *Proc. 41th IEEE Conf. on Decision and Control*, pages 1582–1587, 2002.

[10] A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.

[11] A. Bemporad and M. Morari. Verification of hybrid systems via mathematical programming. In F.W. Vaandrager and J.H. van Schuppen, editors, *Hybrid Systems: Computation and Control*, volume 1569 of *Lecture Notes in Computer Science*, pages 31–45. Springer-Verlag, 1999.

[12] A. Bemporad, M. Morari, V. Dua, and E.N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

[13] A. Bemporad, F.D. Torrisi, and M. Morari. Optimization-based verification and stability characterization of piecewise affine and hybrid systems. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*, pages 45–58. Springer-Verlag, 2000.

[14] A. Bemporad, F.D. Torrisi, and M. Morari. Discrete-time hybrid modeling and verification of the batch evaporator process benchmark. *European Journal of Control*, 7(4):382–399, July 2001.

[15] P.J. Campo and M. Morari. Robust model predictive control. In *Proc. American Contr. Conf.*, volume 2, pages 1021–1026, 1987.

[16] R. DeCarlo, M. Branicky, S. Pettersson, and B. Lennartson. Perspectives and results on the stability and stabilizability of hybrid systems. *Proceedings of the IEEE*, 88(7):1069–1082, 2000.

[17] P. Grieder, F. Borrelli, F.D. Torrisi, and M. Morari. Computation of the constrained infinite time linear quadratic regulator. In *Proc. American Contr. Conf.*, Denver, Colorado, June 2003.

[18] M. Johannson and A. Rantzer. Computation of piece-wise quadratic Lyapunov functions for hybrid systems. *IEEE Trans. Automatic Control*, 43(4):555–559, 1998.

[19] P. Kapasouris, M. Athans, and G. Stein. Design of feedback control systems for unstable plants with saturating actuators. In *Proc. IFAC Symp. on Nonlinear Control System Design*, pages 302–307, Pergamon Press, 1990.

[20] R. Möbus, M. Baotić, and M. Morari. Multi-objective adaptive cruise control. In O. Maler and A. Pnueli, editors, *Hybrid Systems: Computation and Control*, number 2623 in Lecture Notes in Computer Science, pages 359–374. Springer-Verlag, 2003.

[21] S.J. Qin and T.A. Badgwell. An overview of industrial model predictive control technology. In *Chemical Process Control - V*, volume 93, no. 316, pages 232–256. AIChe Symposium Series - American Institute of Chemical Engineers, 1997.

[22] B.I. Silva, O. Stursberg, B.H. Krogh, and S. Engell. An assessment of the current status of algorithmic approaches to the verification of hybrid systems. In *Proc. 40th IEEE Conf. on Decision and Control*, pages 2867–2874, Orlando, Florida, December 2001.

[23] E.D. Sontag. Nonlinear regulation: The piecewise linear approach. *IEEE Trans. Automatic Control*, 26(2):346–358, April 1981.

[24] P. Tøndel, T. A. Johansen, and A. Bemporad. An algorithm for multi-parametric quadratic programming and explicit MPC solutions. In *Proc. 40th IEEE Conf. on Decision and Control*, pages 1199–1204, Orlando, Florida, 2001.

[25] F.D. Torrisi. *Modeling and Reach-Set Computation for Analysis and Optimal Control of Discrete Hybrid Automata*. PhD thesis, Automatic Control Labotatory - ETH, Zurich, 2003.

[26] F.D. Torrisi and A. Bemporad. Discrete-time hybrid modeling and verification. In *Proc. 40th IEEE Conf. on Decision and Control*, pages 2899–2904, Orlando, Florida, 2001.

[27] F.D. Torrisi and A. Bemporad. HYSDEL — A tool for generating computational hybrid models. *IEEE Trans. Contr. Systems Technology*, 12(2):235–249, March 2004.

Appendix A

MILP Formulation (∞-Norm)

The sum of the components of any vector

$$q \triangleq [\varepsilon_0^u, \ldots, \varepsilon_{N-1}^u, \varepsilon_0^z, \ldots, \varepsilon_{N-1}^z, \varepsilon_1^x, \ldots, \varepsilon_N^x, \varepsilon_0^y, \ldots, \varepsilon_{N-1}^y]'$$

that satisfies

$$\begin{array}{l} \mathbf{1}_m \varepsilon_k^u \geq \pm Q_u(u(k|t) - u_r) \; k = 0, 1, \ldots, N-1 \\ \mathbf{1}_{r_c} \varepsilon_k^z \geq \pm Q_z(z(k|t) - z_r) \; k = 0, 1, \ldots, N-1 \\ \mathbf{1}_n \varepsilon_k^x \geq \pm Q_x(x(k|t) - x_r) \; k = 1, 2, \ldots, N-1 \\ \mathbf{1}_n \varepsilon_T^x \geq \pm Q_{xN}(x(N|t) - x_r) \\ \mathbf{1}_p \varepsilon_k^y \geq \pm Q_y(y(k|t) - y_r) \; k = 0, 1, \ldots, N-1 \end{array} \tag{A.1}$$

represents an upper bound on $J(v_0^{T-1}, x(t))$, where $\mathbf{1}_h$ is a column vector of ones of length h, and where

$$x(k|t) = A^k x(t) + \sum_{j=0}^{k-1} A^j (B_1 u(k-1-j|t) + \tag{A.2}$$

$$B_2 \delta(k-1-j|t) + B_3 z(k-1-j|t)). \tag{A.3}$$

The current version of the software allows nonzero matrices D_1, D_2, D_3 of the MLD model, and therefore possible direct feedthrough from the input $u(t)$ to the output $y(t)$[1].

Similarly to what was shown in [15], it is easy to prove that the vector q that satisfies equations (A.1) and simultaneously minimizes

$$J(q) = \sum_{k=0}^{N-1} \varepsilon_k^u + \sum_{k=0}^{N} \varepsilon_k^x + \sum_{k=0}^{N-1} \varepsilon_k^z + \sum_{k=0}^{N-1} \varepsilon_k^y$$

also solves the original problem, i.e. the same optimum $J^*(v_0^{N-1}, x(t))$ is achieved.

[1]Note that even if $D_1 = 0$, I/O feedthrough may still result through the D_2, D_3 matrices, given that the auxiliary variables δ and z are in static functions of both $x(t), u(t)$, in general.

Therefore, problem (4.5) can be reformulated as the following MILP problem

$$
\begin{array}{lrcl}
\min\limits_{q} & J(q) & = & [1\ 1\ \ldots\ 1]q \\
\\
\text{s.t.} & \mathbf{1_m}\varepsilon_k^u & \geq & \pm Q_u(u(k|t)-u_e),\ \ k=0,1,\ldots,N-1 \\
& \mathbf{1_m}\varepsilon_k^z & \geq & \pm Q_z(z(k|t)-z_e),\ \ k=0,1,\ldots,N-1 \\
& \mathbf{1_n}\varepsilon_k^x & \geq & \pm Q_x(x(k|t)-x_e),\ \ k=1,\ldots,N-1 \\
& \mathbf{1_n}\varepsilon_N^x & \geq & \pm Q_{xN}(x(N|t)-x_e) \\
& \mathbf{1_p}\varepsilon_k^y & \leq & \pm Q_y(k|t)-y_e),\ \ k=0,\ldots,T-1 \\
& u_{\min} & \leq & u(k|t)\leq u_{\max},\ \ k=0,1,\ldots,N-1 \\
& y_{\min} & \leq & y(k|t)\leq y_{\max},\ \ k=0,\ldots,N-1 \\
& x_{\min} & \leq & x(k|t)\leq x_{\max},\ \ k=1,\ldots,N \\
& H_N x(T|t) & \leq & K_N \\
& x(k+1|t) & = & Ax(k|t)+B_1u(k)+B_2\delta(k|t)+B_3z(k|t),\ \ k\geq 0 \\
& y(k|t) & = & Cx(k|t)+D_1u(k)+D_2\delta(k|t)+D_3z(k|t) \\
& E_2\delta(k|t) & + & E_3z(k|t)\leq E_1u(k)+E_4x(k|t)+E_5\ \ k\geq 0 \\
& \overline{E}_2\delta(k|t) & + & \overline{E}_3z(k|t)=\overline{E}_1u(k)+\overline{E}_4x(k|t)+\overline{E}_5,\ \ k\geq 0
\end{array}
\tag{A.4}
$$

where the variable $x(0|t)$ appears only in the constraints in (A.4) as a vector parameter. Problem (A.4) can be rewritten in the more compact MILP form

$$
\begin{array}{rll}
q_t^* \triangleq & \arg\min\limits_{q} & f_c^T q_c + f_d^T q_d \\
\\
& \text{s.t.} & G_c q_c + G_c q_d \leq S + Fx(t)
\end{array}
\tag{A.5}
$$

where the matrices G, S, F can be straightforwardly defined from (A.4), and q_c, q_d represent the continuous and binary components, respectively, of the optimization vector q. The case of quadratic cost functions leads to an MIQP, whose derivation is very similar and is therefore omitted here.

Appendix B

Supported Solvers

B.1 Linear Programming

B.1.1 LPSOL, LPTYPE

- `solver='glpk'` uses the LP from the GLPK GNU library developed by A. Makhorin through the Maltab interface developed by N. Giorgetti `glpkmex.dll` (default).
- `solver='lp'` uses `lp.m` for LP from the Optimization Toolbox for Matlab (performance is scarce).
- `solver='nag'` uses `e04mbf` or `e04mf` for LP from the NAG Foundation Toolbox for Matlab, depending on the available version of NAG.
- `solver='cplex'` uses the LP solver of CPLEX (Ilog, Inc.) through the Matlab interface to CPLEX developed by F.D. Torrisi and maintained by M. Baotic.
- `solver='qpact'` uses the active set method `qpact.dll` also for solving LPs (reliability is scarce).
- `solver='linprog'` uses `linprog.m` for LP from the Optimization Toolbox for Matlab (performance is scarce).
- `solver='cdd'` uses `cddmex.dll` by K. Fukuda for LP, through the Matlab interface developed by F.D. Torrisi and maintained by M. Baotic.

B.2 Quadratic Programming

B.2.1 QPSOL, QPTYPE

- `solver='qpact'` uses the active set method `qpact.dll` for solving QPs (default).

- `solver='qp'` uses `qp.m` from the Optimization Toolbox for Matlab (performance is scarce).
- `solver='nag'` uses `e04naf` or `e04nf` from the NAG Foundation Toolbox for Matlab, depending of the available version.
- `solver='cplex'` uses the QP solver of CPLEX (Ilog, Inc.) through the Matlab interface to CPLEX developed by F.D. Torrisi and maintained by M. Baotic.
- `solver='quadprog'` uses `quadprog.m` for QP from the Optimization Toolbox for Matlab (performance is scarce).

B.3 Mixed-Integer Linear Programming

B.3.1 MILPSOL, MILPTYPE

- `solver='glpk'` uses the MILP from the GLPK GNU library developed by A. Makhorin through the Maltab interface developed by N. Giorgetti `glpkmex.dll` (default).
- `solver='matlab'` uses `miqp.m` + `lp.m` for MILP (performance is scarce).
- `solver='nag'` uses `miqp3_naf.m` + `e04naf` from the NAG Foundation Toolbox for MILP (performance is not excellent).
- `solver='cplex'` uses the MILP solver of CPLEX (Ilog, Inc.) through the Matlab interface to CPLEX developed by N. Giorgetti (most powerful and reliable).
- `solver='xpress'` uses the MIQP solver of Xpress-MP (Dash Optimization) through the Matlab interface developed by N. Giorgetti.
- `solver='linprog'` uses `miqp.m` + `linprog.m` for MILP (performance is scarce).

B.4 Mixed-Integer Quadratic Programming

B.4.1 MIQPSOL, MIQPTYPE

- `solver='miqp'` uses `miqp.m` + `quadprog.m` for MIQP (default, slow).
- `solver='nag'` uses `miqp.m` + `e04mbf` from the NAG Foundation Toolbox for MIQP (performance is not excellent).
- `solver='cplex'` uses the MIQP solver of CPLEX (Ilog, Inc.) through the Matlab interface developed by N. Giorgetti (most powerful and reliable).

- `solver='xpress'` uses the MIQP solver of Xpress-MP (Dash Optimization) through the Matlab interface developed by N. Giorgetti.
- `solver='qpact'` uses `miqp.m` + `qpact.dll` for MIQP (performance is scarce).

Appendix C

Storage of Polyhedral Partitions

A polyhedral partition of neighboring convex polytopes $H_i\theta \leq K_i$ in the θ-space and the optimizer $z(\theta) = F_i\theta + G_i$ in each region #i is stored as a structure with the following fields:

- `H,K,i1,i2`: The matrices H_i, K_i of the polyhedral region #i are stored in `H(i1(i):i2(i),:)` and `K(i1(i):i2(i),:)`.
- `F,G`: The gains F_i, G_i of the optimizer $x(\theta) = F_i\theta + G_i$ are stored in `F((i-1)*n+1:i*n,:)`, `G((i-1)*n+1:i*n)`.
- `rCheb`: `rCheb[i]` is the Chebychev radius of region #i, i.e., the radius of the largest ball contained in $H_i\theta \leq K_i$. It provides and information of how flat the region #i is.
- `act,i3,i4`: The combination of active constraints corresponding to region #i is stored in `act(i3(i):i4(i),:)`.
- `unconstr_num`: The number of the region where no constraints are active (only for mpQP problems)
- `nr`: Total number of regions in the partition.

Appendix D

Auxiliary Functions

D.1 GETCONTROLLER

Syntax: `expcon=getcontroller(mpqpsol,nu,uniteeps,flattol,join,lpsolver)`

`getcontroller` computes the constrained controller from the mpQP solution `mpqpsol`. Contrarily to the structure `mpqpsol` used to store the solution to a mpQP problem, in `expcon` the gains F_i, G_i have `nu` components, where `nu` is the number of control inputs to the system, instead of `nu` multiplied by the input prediction horizon.

If the flag `join=1`, pairs of regions where the controller gains are the same are united, provided that the union is a convex set. `uniteeps` defines the tolerance used to detect that the gains are equal, $\|[F_i\ G_i] - [F_j\ G_j]\|_\infty \leq$ `uniteeps`.

`flattol` is a relaxation tolerance. Polyhedral regions of the mpQP solution whose Chebychev radius is smaller than `flattol` are eliminated. The remaining regions $H_i x \leq K_i$ are enlarged into $H_i x \leq K_i +$ `flattol`$\|H_i\|$ (see `reduce`). This guarantees that no "holes" remain in the partition. A positive `flattol` is always recommended (e.g.: 10^{-6}), as the mpQP solution may contain thin "holes" due to numerical precision.

The output argument `expcon` is a structure with the following fields: `H,K,G,F, i1,i2,nr,thmin,thmax,nu,npar`. Contrarily to the structure `mpqpsol` used to store the solution to a mpQP problem, `expcon` does not contain the fields `i3,i4,act`, as different combinations of active constraints may correspond to the same region of the controller after the union of polyhedra.

Syntax: `[expcon,colors]=getcontroller(mpqpsol,...)`
also returns a matrix of RGB colors, with as many rows as there are regions in the control law, that can be used to plot piecewise affine partitions, so that regions having the same affine gain are depicted in the same color. See `pwaplot`.

`getcontroller` is located in the `@expcon/private` folder.

D.2 REDUCE

Syntax: `sol1=reduce(sol,flattol)`

`REDUCE` is used to eliminate small regions from the solution. `sol` can be either an mpQP solution structure, or an explicit MPC control structure. The regions whose Chebychev radius is smaller than `flattol` are removed from the solution, and the remaining regions are enlarged so that no hole remains. After removal, regions may be overlapping, to guarantee the coverage of the whole parameter set $\theta_{\min} \leq \theta \leq \theta_{\max}$. See `getcontroller` for details on the relaxation tolerance `flattol`.

D.3 PWAEVAL

Syntax: `[u,j]=pwaeval(expmpc,x)`

`PWAEVAL` is the M-function form of the explicit MPC controller described by the structure `expmpc`. It computes the MPC control action `u` for the current vector of parameters `x` using the PWA map stored in the structure `expmpc`. The index `j` is the region of the PWA map containing `x`. In case regions overlap because a large relaxation tolerance `flattol` was used, `j` is the first region where `x` is found to belong to.

D.4 PWAPLOT

Syntax: `pwaplot(sol,colors)`

`pwaplot` plots the polyhedral partition corresponding to the multiparametric solution / explicit controller `sol`. Only 2-D plots are supported. If the parameters are more than two, plots can be obtained through `pwaplotsection`. The input argument `colors` is optional and specifies a n_r-by-3 vector of RGB colors, where n_r is the number of regions in the polyhedral partition. An example of output of `pwaplot` is depicted in Figure 5.6.

D.5 PWAPLOTSECTION

Syntax: `sol1=pwaplotsection(sol,index,values,plotflag)`

`pwaplotsection` obtains a section of the partition defined by the explicit controller or multiparametric solution `sol` by fixing $\theta(\texttt{index}) = \texttt{values}$. The output

argument `sol1` is optional, and stores the new 2-D partition obtained by fixing the specified parameters. `plotflag` is an optional input flag, when `plotflag=0` the section is not plotted.

D.6 GETGAIN

Syntax: `[Fi,Gi,Hi,Ki]=getgain(expcon,reg)`

`getgain` gets the polyhedron $H_i\theta \leq K_i$ and the gains F_i, G_i corresponding to the region number `reg` in the explicit controller object `expcon`.

D.7 GETREGNUM

Syntax: `[reg,x]=getregnum(expcon,n)`

When a 2-D partition is plotted, `getregnum` allows to pick up `n` points on the screen with the mouse and obtain the corresponding region numbers `reg`. `x` stores the picked-up points.

D.8 POLYREDUCE

Syntax: `[C,D,isemptypoly,keptrows,lpsolved,x0]=polyreduce(A,B,`
`solver,removetol,checkempty,x0,zerotol)`

Given the polyhedron `A`$x \leq$`B`, returns an equivalent polyhedron `C`$x \leq$`D` by eliminating redundant constrain inequalities. `isemptypoly=1` if the given polyhedron is empty. `keptrows` is an array collecting the indices of nonredundant rows of `A`$x \leq$`B`. `lpsolved` is the number of LP solved to reduce the polyhedron. The output argument `x0` is a feasible point in `A`$x \leq$`B`, or `x0=NaN` if the polyhedron is empty.

D.9 POLYPLOT

Syntax: `[V,handle]=polyplot(A,B,c)`

2-D plot routine for polyhedra. `polyplot` plots the polygon `A`$x \leq$`B`. It it assumed that `A`$x \leq$`B` does not have redundant constraints (see `polyreduce`).

`polyplot(V)` plots the polytope obtained by the convex hull of points in the cell array `V`. `V=polyplot(A,B)` only returns the (ordered) vertices in the cell array V. `[V,handle]=polyplot(A,B)` or `handle=polyplot(V)` also returns the handle to the PATCH object that represents the polygon.

`polyplot(A,B,c)`, `polyplot(V,c)` draws the polytope with fill color `c=[r g b]`.

D.10 LINEPLOT

Syntax: `lineplot(a,b,width,color)`

2-D plot routine for plotting lines. `lineplot` plots the line `a`$'x =$`b`. The optional argument `width` is the width of the line, `color` is the fill color of the line, `color=[r g b]`.

D.11 POLYPLOT3D

Syntax: `handles=polyplot3d(A,B,c)`

3-D plot routine for polyhedra. `polyplot3d` plots the polyhedron `A`$x \leq$`B`. It it assumed that `A`$x \leq$`B` does not have redundant constraints (see `polyreduce`).

`polyplot(A,B,c)`, `polyplot(V,c)` draws the polytope with fill color `c=[r g b]`.

D.12 MPLP

Syntax: `mplpsol=mplp(c,A,b,S,thmin,thmax,verbose,solver,envelope,Hth,Kth)`

This function solves the multiparametric Linear Programming (mpLP)

$$\begin{array}{ll} \min & c'x \\ \text{s.t.} & Ax \leq b + S\theta \end{array} \tag{D.1}$$

within the box $\theta_{\min} \leq \theta \leq \theta_{\max}$. In (D.1), $x \in \mathbb{R}^n$ is the optimization vector, $\theta \in \mathbb{R}^m$ is the vector of parameters, $c \in \mathbb{R}^{n \times 1}$, $G \in \mathbb{R}^{q \times n}$, $W \in \mathbb{R}^q$, $S \in \mathbb{R}^{q \times m}$, where q is the number of constraints.

The output arguments of the function is a partition of neighboring convex polytopes $H_i\theta \leq K_i$ in the θ-space and the optimizer $x(\theta) = F_i\theta + G_i \in \mathbb{R}^n$ in each region #i. These are stored in the structure `mplpsol`, see Appendix C for details. The input argument `verbose` is the level of verbosity, between 0 (lowest) and 2 (highest), `solver` specifies the LP solver to be used (see Section B.1). The flag `envelope` specifies if the (convex) polyhedral hyperplane representation of the set of feasible parameters (i.e., of the union of the critical regions) must be computed. This is stored in `mplpsol.Aenv`, `mplpsol.Benv` (default: `envelope=0`).

The extra arguments `Hth`, `Kth` specify a polyhedral region in the parameters space. Only the critical regions of the feasible set of parameters that intersect the polyhedron $\{\theta : H_{th}\theta \leq K_{th}\}$ are taken into account in the solution.

D.13 MPQP

Syntax: `mpqpsol=mpqp(Q,C,A,b,S,thmin,thmax)`

This function solves the multiparametric Quadratic Programming (mpQP)

$$\begin{array}{ll} \min & \frac{1}{2}x'Qx + \theta'C'x \\ \text{s.t.} & Ax \leq b + S\theta \end{array} \tag{D.2}$$

within the box $\theta_{\min} \leq \theta \leq \theta_{\max}$, using the method proposed in [24]. In (D.2), $x \in \mathbb{R}^n$ is the optimization vector, $\theta \in \mathbb{R}^m$ is the vector of parameters, $Q \in \mathbb{R}^{n\times n}$, $C \in \mathbb{R}^{n\times m}$, $G \in \mathbb{R}^{q\times n}$, $W \in \mathbb{R}^q$, $S \in \mathbb{R}^{q\times m}$, where q is the number of constraints.

The output arguments of the function is a partition of neighboring convex polytopes $H_i\theta \leq K_i$ in the θ-space and the optimizer $x(\theta) = F_i\theta + G_i \in \mathbb{R}^n$ in each region #i. These are stored in the structure `mpqpsol`, see Appendix C for details.

Syntax: `mpqpsol=mpqp(Q,C,A,b,S,thmin,thmax,verbose,qpsolver,...`
`lpsolver,envelope,Hth,Kth)`

The optional input arguments have the following meaning: `verbose` is the level of verbosity, between 0 (lowest) and 2 (highest), `qpsolver` specifies the QP solver to be used (see Section B.2), `lpsolver` the LP solver to be used (see Section B.1), `envelope=1` also computes the polyhedral hyperplane representation of the set of feasible parameters, i.e., of the union of the critical regions (the envelope is stored in `mpqpsol.Aenv`, `mpqpsol.Benv`, `Hth` and `Kth` impose to only compute the regions of the feasible set of parameters that intersect the polyhedron $\{\theta : H_{th}\theta \leq K_{th}\}$.