J Intell Robot Syst
DOI 10.1007/s10846-007-9180-7

# Model-based Predictive Control of Hybrid Systems: A Probabilistic Neural-network Approach to Real-time Control

**Boštjan Potočnik · Gašper Mušič · Igor Škrjanc · Borut Zupančič**

Received: 7 September 2006 / Accepted: 14 August 2007
© Springer Science + Business Media B.V. 2007

**Abstract** This paper proposes an approach for reducing the computational complexity of a model-predictive-control strategy for discrete-time hybrid systems with discrete inputs only. Existing solutions are based on dynamic programming and multi-parametric programming approaches, while the one proposed in this paper is based on a modified version of performance-driven reachability analyses. The algorithm abstracts the behaviour of the hybrid system by building a 'tree of evolution'. The nodes of the tree represent the reachable states of a process, and the branches correspond to input combinations leading to designated states. A cost-function value is associated with each node and based on this value the exploration of the tree is driven. For any initial state, an input sequence is thus obtained, driving the system optimally over a finite horizon. According to the model predictive strategy, only the first input is actually applied to the system. The number of possible discrete input combinations is finite and the feasible set of the states of the system may be partitioned according to the optimization results. In the proposed approach, the partitioning is performed offline and a *probabilistic neural network* (PNN) is then trained by the set of points at the borders of the state-space partitions. The trained PNN is used as a system-state-based control-law classifier. Thus, the online computational effort is minimized and the control can be implemented in real time.

**Keywords** Hybrid systems · Model predictive control · Reachability analysis · Probabilistic neural networks

## 1 Introduction

Hybrid systems were recognized as an emerging research area within the control community in the past decade. With improvements to the control equipment

B. Potočnik · G. Mušič (✉) · I. Škrjanc · B. Zupančič
Faculty of Electrical Engineering, University of Ljubljana,
Tržaška 25, 1000 Ljubljana, Slovenia
e-mail: gasper.music@fe.uni-lj.si.

the complexity of modern computer-control systems increases. Various aspects of discrete-event operation, such as controller switching, changing operating modes, communication delays, and interactions between different control levels within the computer-control systems are becoming increasingly important. Hybrid systems, defined as systems with interacting continuous and discrete-event dynamics, are the most appropriate theoretical framework to address these issues.

Many modelling formalisms for hybrid systems were proposed in the engineering literature. Of particular interest are the discrete-time modelling formalisms. *Discrete hybrid automata* (DHA) [28] can be considered as a basis for hybrid system modelling as they can be modelled using the Hybrid System Description Language (HYSDEL). Using an appropriate compiler, a DHA model described by the HYSDEL modelling language can be translated to different modelling frameworks, such as *mixed logical dynamical* (MLD), *piecewise affine* (PWA), *linear complementarity*, *extended linear complementarity* or *max-min-plus-scaling* systems [17, 28]. In this paper the MLD modelling framework [5] will be adopted, as it is suitable for solving optimal and model predictive control problems.

Optimal control approaches for hybrid systems have been thoroughly investigated in recent years. The optimal control of hybrid systems in manufacturing is addressed in Pepyne and Cassandras [21] Cassandras et al. [12] and Gokbayrak and Cassandras [15], where the authors combine time-driven and event-driven methodologies to solve optimal control problems. An algorithm to optimize the switching sequences for a class of switched linear problems is presented in Lincoln and Rantzer [19], where the algorithm searches for solutions that are arbitrarily close to the optimal ones. A similar problem is addressed in Barton et al. [2], where the potential for numerical optimization procedures to make optimal sequencing decisions in hybrid dynamic systems is explored. A computational approach based on ideas from dynamic programming and convex optimization is presented in Hedlund and Rantzer [16]. Piecewise linear quadratic optimal control is addressed in Rantzer and Johansson [25], where the use of piecewise quadratic cost functions is extended from a stability analysis of piecewise linear systems. Optimal control based on a reachability analysis, and where the inputs of the system are continuous, is addressed in Bemporad et al. [4].

On the other hand, model predictive control was successfully applied to complex constrained control problems in industry. This suggests the application of model-predictive-control approaches also for hybrid systems. A model-predictive-control technique is presented in Bemporad and Morari [5]; this is able to stabilize the MLD system on the desired reference trajectories, and the online optimization procedures are solved through *mixed-integer quadratic programming* (MIQP). In Bemporad et al. [7] the authors show how to formulate an optimal control problem for a constrained linear discrete-time system as a multi-parametric program and how to solve such a program. The basic ideas were extended to PWA systems and thoroughly investigated in Kerrigan and Mayne [18], Baotić et al. [1], Borrelli et al. [10, 11], and Borrelli [9], where both optimal and model-predictive-control approaches are discussed. The latter, in particular, have been successfully applied to many real problems [20].

In this paper we study and discuss the solution to a model predictive control for linear discrete-time hybrid systems with discrete inputs only, where the system is described as a MLD system [5]. Many of the control approaches are limited to

discrete-time hybrid systems because many of the complex mathematical issues are removed. In many applications the command variables are intrinsically discrete, either because such a system design is simpler or for other technological reasons.

The existing mixed-integer programming approaches can also be applied to systems with discrete inputs, but only in cases with a small number of discrete inputs and a small number of auxiliary internal system variables ($\delta$ and $z$ variables in MLD terminology - see Section 2). The reasons lie in the computational complexity of the optimization problem. An new approach is described in this paper, which is more efficient and also more general in terms of the allowable cost function.

Despite the reduced computational complexity the application of the proposed control algorithm in real time remains difficult, particularly in cases where a short sampling time is imposed by the dynamics of the system. The mixed-integer programming approaches try to solve this by multi-parametric programming, which is applied offline, and the control law is explicitly expressed as a function of the system state [9]. However, the offline computational effort is large and with the increased number of discrete inputs and discrete internal auxiliary variables it is not possible to obtain a solution in a reasonable time. The reason is in the large number of multi-parametric programs that need to be solved when the control law is computed in a dynamic programming fashion. In this paper an alternative method is proposed, where the partition of the state space is determined by the control-law calculation based on a reachability analysis, and a *probabilistic neural network* (PNN) is trained by a set of points close to the borders of the state-space partitions. The PNN is then used as a system-state-based control-law classifier in real time.

The paper is organized as follows. In Section 2 we address the MLD modelling framework, as the model predictive approaches based on this model will be discussed in the paper. The problems of the model predictive control of hybrid systems with discrete inputs and the proposed solution are addressed in Section 3. The proposed algorithm is applied to a system of two furnaces, and the results are discussed in Section 4. The conclusions are given in Section 5.

## 2 MLD Systems

Discrete-time models of hybrid systems are a combination of logic, finite-state machines, linear discrete-time dynamic systems and constraints [5]. The interaction between continuous and discrete/logic dynamics is shown in Fig. 1, where both parts are connected through interfaces. The MLD modelling framework is based on the idea of translating logic relations, discrete/logic dynamics, *A/D* (analog to digital (logic)), *D/A* conversion and logic constraints into mixed-integer linear inequalities. These inequalities are combined with the continuous dynamical part, which is described by linear difference equations. The resulting MLD system is described by the following relations:

$$x(k+1) = Ax(k) + B_1u(k) + B_2\delta(k) + B_3z(k) \tag{1a}$$

$$y(k) = Cx(k) + D_1u(k) + D_2\delta(k) + D_3z(k) \tag{1b}$$

$$E_2\delta(k) + E_3z(k) \leq E_1u(k) + E_4x(k) + E_5, \tag{1c}$$

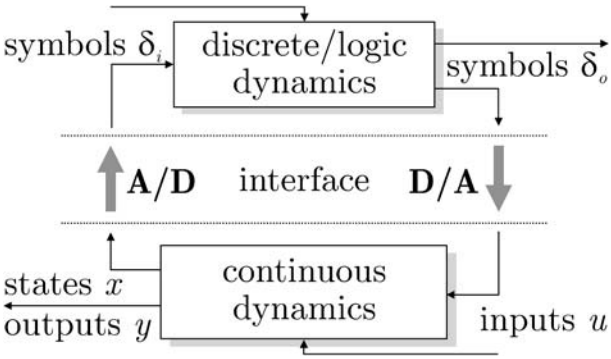



**Fig. 1** Hybrid control system—discrete and continuous dynamics interact through interfaces

where $x(k) \in \mathbb{R}^{n_c} \times \{0,1\}^{n_b}$ is a vector of continuous and logic states, $u(k) \in \mathbb{R}^{m_c} \times \{0,1\}^{m_b}$ are the inputs, $y(k) \in \mathbb{R}^{p_c} \times \{0,1\}^{p_b}$ are the outputs and $\delta \in \{0,1\}^{r_b}$ and $z \in \mathbb{R}^{r_c}$ are the logic and continuous auxiliary variables, respectively. The inequalities (1c) can also include physical constraints over continuous variables (states and inputs). Given the current state $x(k)$ and the input $u(k)$, the time evolution of Eq. 1 is determined by solving $\delta(k)$ and $z(k)$ from Eq. 1c, and then updating $x(k+1)$ and $y(k)$ from Eq. 1a, Eq. 1b. The MLD system Eq. 1 is assumed to be well posed if for a given state $x(k)$ and a given input $u(k)$ the inequalities Eq. 1c have a unique solution for $\delta(k)$ and $z(k)$. Because they are so extensive, the details of the translation techniques from logic relations, discrete/logic dynamics, *A/D* [analog to digital (logic)], *D/A* conversion and logic constraints into mixed-integer linear inequalities, are not given here. For a more detailed description of the MLD form and translation techniques the reader is referred to Bemporad and Morari [5].

MLD models can be efficiently built by using the HYSDEL modelling language, which allows a description of the DHA hybrid dynamics in textual form. Using an associated compiler this form can be translated into MLD form [5]. For a more detailed description of the syntax and the functionality of the HYSDEL modelling language and the associated compiler (HYSDEL tool) the reader is referred to Torrisi and Bemporad [28].

Several control approaches referenced in the paper are based on a PWA representation of hybrid systems. Due to the equivalence of the MLD and PWA modelling frameworks [17], similar control approaches can be used for MLD systems. Like with MLD systems, PWA models can be efficiently built using the HYSDEL modelling language, since the resulting MLD model can be efficiently converted into a PWA form [3]. Furthermore, a direct translation from a HYSDEL model (DHA) to the PWA form is possible, without the intermediate MLD step [24]. For a more detailed description of PWA systems, the reader is referred to Sontag [26] and Ferrari-Trecate et al. [14] and the references therein.

## 3 Model Predictive Control of Hybrid Systems

Predictive control amounts to finding the control sequence $V_0^{H-1} = \{v(0), ..., v(h), ..., v(H-1)\}$ in a horizon $H$, which transfers the initial state $x(k|k)$ as close

as possible to the final state $x_f$ in a horizon time $T = H \cdot T_s$ ($T_s$ is the sampling time), thus minimizing the performance index. Only the first sample of the optimal control sequence $V_0^{H-1}$ is actually applied to the plant at the time step $k$. At the time $k+1$ a new sequence is evaluated to replace the previous one.

### 3.1 Multi-parametric Mixed-integer Programming Approach

In this section the multi-parametric program solution to the predictive control problem will be briefly reviewed. The aim is to demonstrate the approach's strengths and weaknesses when dealing with hybrid systems that have discrete inputs.

In Borrelli [9] the author presents a constrained model predictive problem, which is based on a PWA model that is subject to hard input and state constraints. A cost function is defined as

$$J\left(x(k), V_0^{H-1}\right) \triangleq \|x(k+H|k)-x_f\|_P^p + \sum_{h=0}^{H-1} \|x(k+h|k)-x_f\|_Q^p + \|v(h)-u_f\|_R^p \tag{2}$$

where $\|x\|_Q^p$ denotes the $p$–norm of the vector $Qx$ if $p = 1, \infty$ or $x'Qx$ for $p = 2$. $Q$, $R$ and $P$ are weighting matrices with the following properties: $Q = Q' \geq 0$, $R = R' > 0$, $P \geq 0$ for $p = 2$, and are full column rank for $p = 1, \infty$. $h = 0, ..., H-1$ is the prediction step, $x(k+h|k) \triangleq x(k+h, x(k), V_0^{h-1})$ is the predicted state at $h$, $y(k+h|k)$ is defined similarly, and $V_0^{H-1} = \{v(0), ..., v(H-1)\}$ is the optimal input sequence defined by the optimization algorithm.

Applying the MLD modelling framework the following model predictive problem is formulated:

$$J^*(x(k)) \triangleq \min_{V_0^{H-1}} J\left(x(k), V_0^{H-1}\right) \tag{3a}$$

subj. to

$$\begin{aligned} x(k+h+1|k) = {} & Ax(k+h|k)+B_1v(h)+B_2\delta(k+h|k) \\ & +B_3z(k+h|k)\,E_2\delta(k+h|k)+E_3z(k+h|k) \\ \leq {} & E_1v(h)+E_4x(k+h|k)+E_5. \end{aligned} \tag{3b}$$

Following the predictive control approach, only the first element in the sequence $V_0^{H-1}$ is applied as an input to the system, i.e., $u(k) = v(0)$.

Later, the problem Eq. 3 can be formulated as a MIQP when $p = 2$ norm is used [5], or a *Mixed-Integer Linear Program* (MILP) when $p = 1, \infty$ norm is used:

$$\begin{aligned} & \min_{\mathcal{V}} \mathcal{V}'H_1\mathcal{V} + \mathcal{V}'H_2x(k) + x'(k)H_1x(k) + f_1'\mathcal{V} + f_2'x(k) + c \\ & \text{subj. to } G\mathcal{V} \leq S + Fx(k), \end{aligned} \tag{4}$$

where $\mathcal{V} = [\Omega^T,\ \Delta^T,\ \Xi^T]^T$ with $\Omega = [v(0)^T, ..., v(H-1)^T]^T$ (note that $\Omega$ represents the input sequence $V_0^{H-1}$), $\Delta = [\delta(0)^T, ..., \delta(H-1)^T]^T$, $\Xi = [z(0)^T, ..., z(H-1)^T]^T$,

and $H_1$, $H_2$, $H_3$, $f_1$, $f_2$, $c$ are matrices of suitable dimensions. $H_1$, $H_2$, and $H_3$ are null matrices if problem Eq. 4 is an MILP [9].

Given the value of the current (initial) state $x(k)$, the MIQP (or MILP) Eq. 4 can be solved to obtain the optimal input sequence $\mathcal{V}$, i.e., $\Omega \Rightarrow V_0^{H-1}$. By applying multi-parametric programming [7, 9, 13], where the current (initial) state $x(k)$ is considered as a parameter, the explicit form of the optimal state feedback $u(x(k))$ of the problem Eq. 4 can be obtained. For $p = 2$ norm the optimization problem is treated as a multi-parametric MIQP (mp-MIQP), while for $p = 1, \infty$ norm the optimization problem can be treated as a multi-parametric MILP (mp-MILP). The solution to the mp-MILP (mp-MIQP) is the explicitly defined control law of the form [1]:

$$u(x(k)) = F_i^0 x(k) + G_i^0 \text{ if } x(k) \in P_i^0, \tag{5}$$

where $P_i^0$, $i = 1, ..., N^0$, is a polyhedral partition of the feasible set of the states $x(k)$ at the time step $k$, and is different for $p = 1, \infty$ or $p = 2$ norm [11].

By looking at the definition of the optimization vector $\mathcal{V}$ in Eq. 4 and considering the current state $x(k)$ as a parameter, we can see that the mp-MILP (MIQP) approach solves the problem in extended $(x, v, \delta, z)$ space, which is numerically very difficult to handle and increases the size of the problem drastically. The reason is two fold. Firstly, the complexity of the optimization problem grows exponentially with the number of binary variables. Therefore, including binary variables $\delta$, which are actually explicitly defined by the MLD system, state $x$ and input $v$, into the optimization problem is not efficient. Secondly, the continuous variables $z$ actually represent equality constraints that were translated into two sets of inequalities (the consequence of the MLD modelling framework [5]) and which are also explicitly defined by the MLD system, state $x$ and input $v$. In Baotić et al. [1] an efficient algorithm for computing the solution for the $p = 1, \infty$ norm case and in Borrelli et al. [10, 11] and Borrelli [9] for $p = 2$ norm is given. Both approaches solve the problem in $(x, v)$ space and the problems presented previously are removed. The algorithms are based on a dynamic programming approach and mp-LP(QP) solvers. The limitation of the mentioned algorithms is that they are efficient only for hybrid systems where the states $x(k)$ and the inputs $u(k)$ are defined in $x(k) \in \mathbb{R}^{n_c}$ and $u(k) = v(0) \in \mathbb{R}^{m_c}$, respectively. In spite of these limitations, however, the approaches were successfully applied to many real problems [20].

When dealing with hybrid systems that also contain binary states or binary inputs, i.e., $x(k) \in \mathbb{R}^{n_c} \times \{0, 1\}^{n_b}$ and $u(k) \in \mathbb{R}^{m_c} \times \{0, 1\}^{m_b}$, the above-mentioned efficient algorithms are not practical. It is possible to write a standard dynamic-programming recursion (backwards in time) but such a computation would be prohibitive for all but the simplest of problems. This is a consequence of the drastic increase in the complexity of the corresponding multi-parametric programs that have to be solved, and furthermore, a large number of such programs need to be solved during dynamic programming recursion. For example, in the case of binary inputs, all the possible combinations of binary inputs must be enumerated [11]. Due to the increased computational complexity, a different approach is preferable when dealing with binary states or binary inputs. Such an alternative approach for the case where the hybrid system can be influenced through binary (discrete) inputs only will be discussed in the following.

## 3.2 Performance-driven Approach

The algorithm that will be introduced in what follows is based on a performance-driven reachability analysis [23]. The algorithm abstracts the behaviour of the hybrid system by building a "tree of evolution". A cost-function value is associated with each node of the tree, and based on this value the exploration of the tree is driven. Based on the cost-function value the imperspective branches of the tree are pruned to reduce the computational complexity. As soon as the exploration of the tree is finished, the corresponding input is applied to the system and the procedure is repeated.

### *3.2.1 Complexity of the Control Problem with Discrete Inputs*

The solution to a control problem at every time step $k$ is the control sequence $V_0^{H-1} = \{v_b(0), \ldots, v_b(H-1)\}$, where $u(k) = v_b(0)$ represents the discrete input to the system at step $k$. The subscript $b$ is used to emphasize the fact that we are dealing with binary inputs only. If the system has $m_b$ discrete inputs and no continuous inputs, that means $v_b(h) \in \{0,1\}^{m_b}$ and $V_0^{H-1} \in \{0,1\}^{m_b \cdot H}$. Because all the inputs are binary, there are $2^{m_b \cdot H}$ possible combinations for $V_0^{H-1}$. Hence the optimization problem is NP-hard and the computational time required to solve the problem grows exponentially with the problem size.

### *3.2.2 Optimization Using a Reachability Analysis*

As a result of the constraints Eq. 1c all the combinations of inputs are not, in general, feasible. One way to rule out infeasible inputs is to use a reachability analysis. Such an approach for hybrid systems with continuous inputs is presented in Bemporad et al. [4, 8]. The idea consists of reach-set computation and switching detection, based on the repetitive use of linear programming, coupled to a quadratic programming solver, which selectively drives the exploration. A similar idea is used here, but the presented approach relies on the efficient discrete-time simulation of the underlying MLD model, without the use of linear programming. Instead of using a linear-program feasibility test, the values of the MLD auxiliary variables are explicitly calculated from the previous state and the MLD data structure [22].

Through a reachability analysis it is possible to extract the reachable states of the system; although enumerating all of them would not be effective, as many of them will be far away from the optimal trajectory. Therefore, it is reasonable to combine a reachability analysis with procedures that can detect reachable states that do not lead to the optimal solution and remove them from the exploration procedure. The approach of Bemporad et al. [8] achieves this by adjoining a partial cost to each node, and then the partial cost is calculated by LP or QP optimization, depending on the type of optimization problem. In the case of binary inputs, a MILP or MIQP solver should be used, which increases the computational burden significantly. Limiting to cases with discrete inputs only, the use of mixed-integer programming is unnecessary, and in this paper a more efficient *branch and bound* optimization algorithm is applied, which explores the derived tree-of-evolution structure directly. Within this approach, the concept of the tree of evolution is slightly modified, i.e., the nodes do not represent reachable regions in the state space but points in the state space that

are reached by a certain input combination. The calculation of the partial cost is then straightforward; it is determined by the path from the initial node to the observed node of the tree. The procedure of generating and exploring the tree of evolution will be described in detail in the following section.

### *3.2.3 Tree of Evolution*

By exploiting the reachability analysis technique we are able to abstract the possible evolution of the system over a horizon of $H$ steps into a tree of evolution [23], as shown in Fig. 2. The nodes of the tree represent states that are reachable from a given initial state, and the branches connect two nodes if a transition exists between the corresponding states. Each branch has an associated discrete input applied to the system causing the transition. For a given root node $\mathcal{N}_0$, representing the current (initial) state $x(k|k)$, the reachable states $x^i(k+h|k)$ are computed and inserted into the tree as nodes $\mathcal{N}_i$, while the corresponding discrete inputs $v_b^i(h)$ are associated with the corresponding branches connecting two nodes. $i \in \{1, 2, ...\}$ represents the successive index of the nodes and branches inserted into the tree. To each new node $\mathcal{N}_i$ a cost value $J_i$ is associated. The value is calculated based on the state and the input sequence that has brought the system to the corresponding state. The search for the optimal control sequence is propagated from a selected node of the tree, whose selection is based on the associated cost value $J_i$, e.g., a node with the lowest $J_i$ may be selected. As soon as a new starting node is selected, new states reachable from the selected node are computed and inserted as new nodes. The construction of the tree of evolution proceeds according to a depth-first strategy until one of the following conditions occurs:

- The step horizon limit is reached ($h = H$).
- The value of the cost function at the current node is greater than the current optimal value ($J_i \geq J_{opt}$, where initially $J_{opt} = \infty$).

A node that satisfies one of the above conditions is labelled as explored. If a node satisfies the first condition, the associated value of the cost function $J_i$ becomes the

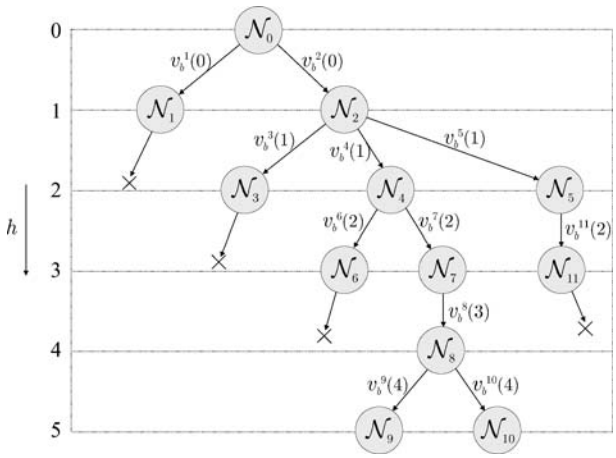



**Fig. 2** The tree of evolution

current optimal value ($J_{opt} = J_i$), and the control sequence $V_0^{H-1}$ that leads from the initial node $\mathcal{N}_0$ to the current node $\mathcal{N}_i$ becomes the current optimizer. The exploration continues with the selection of another unexplored node until all the nodes are explored. When this is accomplished, the control sequence $V_0^{H-1}$ leading to the node with the associated $J_i = J_{opt}$ becomes the optimal one and the input $u(k) = v_b(0)$ is applied as an input to the system. In a standard model-predictive-control approach the optimization is applied online at every time step so the whole optimization procedure, generating a new tree of evolution, is repeated at $k+1$.

### *3.2.4 Cost Function and Node-selection Criterion*

The selection of the cost function and the node-selection criterion have a great influence on the size of the tree of evolution and, indirectly, on the time efficiency of the control algorithm. The best node-selection criterion is to propagate the tree of evolution in a direction that minimizes the value of the cost function. At the same time the cost value $J_i$ associated with a node is used to detect nodes that are not going to lead to the optimal solution. To achieve that, the cost function must have certain properties that we describe below.

In the case of a classical optimal control problem we typically choose a cost function of the form:

$$J\left(x(k), V_0^{H-1}\right) = \|x(k+H|k) - x_f\|_P^p + \sum_{h=0}^{H-1} \|x(k+h|k) - x_f\|_Q^p + \|v_b(h) - u_f\|_R^p \quad (6)$$

where $\|x\|_Q^p$ represents the $p$–norm for $p = 1, 2, \infty$, and where $Q$, $R$ and $P$ are weighting matrices with the same properties as in Eq. 2.

For a node $\mathcal{N}_i$ inserted into the tree of evolution at the prediction step $h < H$ the associated cost value (based on Eq. 6) is

$$J_i\left(x(k), V_0^{h-1}, h\right) = \sum_{j=0}^{h} \|x(k+j|k) - x_f\|_Q^p + \sum_{j=0}^{h-1} \|v_b(j) - u_f\|_R^p. \quad (7)$$

It is reasonable to detect the nodes $\mathcal{N}_i$ that do not lead to the optimal solution at the step instance $h < H$ ($H$ is horizon) by comparing $J_i(h)$ to $J_{\text{opt}}$. When the cost value becomes greater than the current optimal one ($J_i \geq J_{opt}$) we want to ensure that by continuing the exploration no better solution than the current one can be found. To achieve this the cost function Eq. 7 has to be monotonically increasing with the prediction step. The cost function Eq. 6, Eq. 7 fulfils the mentioned condition for $p = 1, 2, \infty$ norm, i.e.,

$$J_i\left(x(k), V_0^h, h+1\right) - J_i\left(x(k), V_0^{h-1}, h\right) \quad (8a)$$

$$\sum_{j=0}^{h+1} \|x(k+j|k) - x_f\|_Q^p + \sum_{j=0}^{h} \|v_b(j) - u_f\|_R^p - \sum_{j=0}^{h} \|x(k+j|k) - x_f\|_Q^p - \sum_{j=0}^{h-1} \|v_b(j) - u_f\|_R^p \quad (8b)$$

$$\|x(k+h+1|k) - x_f\|_Q^p + \|v_b(h) - u_f\|_R^p \geq 0 \qquad \text{for } h = 1, \ldots, H-2. \quad (8c)$$

and at the horizon, the corresponding difference is $\|x(k+H|k)-x_f\|_P^p + \|v_b(H-1)-u_f\|_R^p \geq 0$.

### 3.3 Multi-parametric Solution Using PNNs

Despite the efficiency of the proposed performance-driven approach to model predictive control the related computational burden is often too high for the control to be implemented in real time. This motivates the search for a multi-parametric solution that could be calculated offline.

As shown previously, the solution of a multi-parametric programming approach is of the form Eq. 5, where we want to obtain the explicit solution of the control law and where the state of a system $x(k)$ represents the independent variable. In the case of purely discrete inputs the explicit control law is simplified to:

$$u(x(k|k)) = G_i^0 \text{ if } x(k) \in P_i^0, \tag{9}$$

where $P_i^0, i = 1, ..., N^0$, is a partition of the feasible set of the states $x(k)$ at the time step $k$ and is different for $p = 1, \infty$ or $p = 2$ norm. The question is how to represent the partitions $P_i^0$. The multi-parametric programming approaches presented in Section 3.1 describe the explicit control law Eq. 5 as a PWA function, where the partitions $P_i^0$ are represented by a set of inequalities.

On the other hand, with the performance-driven approach presented in Section 3.2, a single value of an optimal control input can be calculated given the current state $x(k)$ of a system with purely discrete inputs. Since the explicit control law Eq. 9 maps the state space to a finite set of discrete values, partitions $P_i^0$ may be characterized by solving the optimal control problem for a set of points in the state space. If the points with the same value of the corresponding optimal control law are used as a class of prototypes, a control law in the form Eq. 9 can be effectively approximated by using PNNs, as will be shown in the next section.

#### *3.3.1 PNN as a Control-law Classifier*

The number of partitions $P_i^0$ equals the number of discrete control laws $G_i^0$. The partitions $P_i^0$ are in general complex and may also be non-convex. It is therefore a demanding task to classify in real time the current state-space vector into a certain class and define the optimal discrete input value. The most appropriate way to define the class of input to the current state vector is the PNN [27]. The PNN is a pattern-classification algorithm that belongs to a huge group of nearest-neighbour-like algorithms. It corresponds to the neural-network family because of its natural mapping onto a two-layer feed-forward network. The PNN defines the distance, i.e., the likelihood function, of the current state-space vector $x$ from a class of prototypes $x_j^i$, where $i$ stands for the class, $j = 1, ..., N_i$, where $N_i$ represents the number of prototypes, and $d$ stands for the dimensionality of the measurement space:

$$L_i(x) = \frac{1}{N_i(2\pi)^{d/2}\sigma^d} \sum_{j=1}^{N_i} e^{-\left(x-x_j^i\right)^2/2\sigma^2} \tag{10}$$

The conditional probability for class $i$ is equal

$$P_i(x) = \frac{L_i(x)}{\sum_{j=1}^{M} L_j(x)}, \tag{11}$$

where $M$ stands for the number of classes. The observed pattern corresponds to the class with the highest conditional probability.

The PNN is actually a standard Bayesian classifier. It can perform the pattern classification. When applied in its standard form, the PNN does not need to be trained using classical learning algorithms. The training vectors simply become the weight vectors in the first layer of the network. This is an advantage of the PNN, but it could also be a disadvantage when dealing with a huge number of training data. This can be solved by using clustering algorithms to reduce the amount of the training data. The quality of the optimal control law approximation depends on the quality of the partitions $P_i^0$ approximation in the feasible state space. This means that each partition should be defined as exactly as possible, which is particulary important near to the boundaries of partitions.

### *3.3.2 Training of a PNN*

The training of a PNN is easy and instantaneous because the weight vectors in the first layer of the network are defined by the training vectors. The only free parameter is $\sigma$, the variance of the Gaussian, which is defined heuristically during the training procedure by the evaluation of some criterion, as it is the number of classification errors during the cross-validation. The PNN classifier can be used in real time because as soon as one pattern representing each class has been observed, the network can begin to generalize to a new pattern. The shape of the decision boundary can be as complex as desired. The boundaries between the subspaces can be modelled more or less exactly and are defined by parameter $\sigma$.

Working with a PNN is simple and obvious, especially when working within the MATLAB$^{TM}$ environment. The main task that has to be performed is to provide classes of prototypes. In our case the defined class of prototypes is defined by the states $x(k)$ for which the discrete control law is the same, i.e., $X_i = \{x(k) : u(k) = G_i^0\}$ for $i = 1, ..., N_i$, where $N_i$ is the number of different control laws. It is reasonable to provide, for each class of prototypes, the states $x(k)$ that are as close as possible to the border with other classes.

These prototype states or training states can be defined using a simulation. For a chosen initial state $x(0)$, which represents the state of the system at instance $k$, the appropriate control law $G_i^0$ is calculated by applying the performance-driven reachability-based model predictive approach presented earlier. The pair $(x(0), G_i^0)$ represents one element in the class of the prototypes $i$. How many elements will be in each class of prototypes and how we will chose the initial states $x(0)$ can differ from case to case. Of course it is reasonable to find as many boundary initial states $x(0)$ with the associated control laws $G_i^0$ in as few simulation runs as possible. The boundary states can be defined by iterative division of the subspace between two neighboring states which imply different optimal discrete control laws. The iterative procedure is repeated until a certain tolerance partitioning is reached.

### *3.3.3 Proposed Control Design Procedure*

The above-proposed design procedure and the related boundary points' calculation are summarized in two algorithms. Starting with the process model and the problem statement Eq. 3 the first algorithm summarizes the main design steps:

*Algorithm 1*:

Describe the controlled process as a MLD system;
Choose a cost function;
Formulate model predictive control problem Eq. 3;
Call Algorithm 2 to provide classes of prototypes;
Train PNN;
Implement PNN as a real-time control-law classifier;

Another algorithm is called to provide classes of prototypes that are required to train the PNN. These may be constructed in different ways. The algorithm used in this paper is sketched as follows:

*Algorithm 2*:

```
Read state limits from the MLD system;
Define initial subspace based on lower and upper state bounds;
Define initial grid step as a difference between lower and upper bounds;
repeat
    Determine new (smaller) grid step;
    for all subspaces do
        Choose a subspace;
        Create step based grid in the subspace;
        for all grid points do
            x(0) := chosen grid point;
            G_i^0 := optimal control value;
            (* calculated by the performance driven reachability analysis *)
            Add the pair (x(0), G_i^0) to the list of points;
            if new border point then
                Add the subspace defined by x(0) and grid step to the list of
                new subspaces;
            end
        end
    end
    subspaces := new subspaces;
until grid step ≤ tolerance
Form classes of points with the same G_i^0;
Remove interior points;
```

In the proposed algorithm, the lower bound, the upper bound, the grid step and the tolerance are vectors of size $n \times 1$, where $n = n_c + n_b$ is the number of states. Subspaces where the border points are searched for are rectangular in shape and are given by pairs of lower and upper bounds. In the beginning there is only one initial subspace defined by the state bounds of the corresponding MLD model. During

the iterative search, new subspaces are generated whenever the border is detected, i.e., whenever the calculated $G_i^0$ differs from the value at the already-processed neighbouring points in the grid. The search continues until a required tolerance is reached. Finally, the set of points is classified according to the adjoined control value $G_i^0$, and the points surrounded by points of the same class are removed.

The use of a multi-parametric solution using PNNs on an example of the alternate heating of two furnaces will be presented in the section below.

## 4 Example: Alternate Heating of Two Furnaces

The presented model-predictive-control approach was applied to a temperature-control example proposed in Hedlund and Rantzer [16] and studied in Bemporad and Morari [6]. The temperature of the two furnaces should be controlled to a given set point by alternate heating. Only three modes of operation are allowed: heat only the first furnace, heat only the second furnace, do not heat. The furnaces can only be fed by a fixed amount of energy $u_0$. The system is described by the following PWA model:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u_0, \text{ if the first furnace is heated} \tag{12a}$$

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_0, \text{ if the second furnace is heated} \tag{12b}$$

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u_0, \text{ if there is no heating} \tag{12c}$$

The system Eq. 12 has two continuous states that correspond to the temperature of the furnaces, $x_1$ corresponds to the temperature of the first furnace and $x_2$ to the temperature of the second, and two binary inputs, heat furnace 1 and heat furnace 2, which cannot be active simultaneously. The system Eq. 12 was converted to discrete time using the step-invariance method with the sample time $T_s = 0.08$ s and was modelled using the HYSDEL modelling language [28] (see Appendix). The HYSDEL compiler returned the following MLD system:

$$x(k+1) = \begin{bmatrix} 0.9231 & 0 \\ 0 & 0.8521 \end{bmatrix} x(k) + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} z(k) \tag{13a}$$

$$\begin{bmatrix} 0 & 0 \\ -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{bmatrix} z(k) \leq \begin{bmatrix} -1.0000 & -1.0000 \\ -0.0615 & 0 \\ 0.0615 & 0 \\ 0 & -0.0591 \\ 0 & 0.0591 \end{bmatrix} u(k) + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{13b}$$

where $z(k)$ is an auxiliary continuous variable representing the influence of the amount of power $u_0$ on the temperature (state $x$) with regard to the binary input $u = [u_1, u_2]^T$, representing the heating of the first or the second furnace, respectively. The matrices $E_{i=1,\ldots,5}$ are suitably defined by the MLD transformation procedure [5, 28] and $u_0 = 0.8$.

**Table 1** Possible discrete inputs

| Inputs | 1 | 2 | 3 |
|---|---|---|---|
| $u_1$ | 1 | 0 | 0 |
| $u_2$ | 0 | 1 | 0 |

### 4.1 Complexity of the Predictive Control Algorithm

The solution to the model predictive control is a sequence $V_0^{H-1}$. At each sampling instance two inputs can influence the system, i.e., one for the first and one for the second furnace (see Appendix). The horizon applied was set to $H = 3$, $H = 5$ and $H = 7$, respectively. Considering the number of discrete inputs and the length of the horizon there exist $2^{2\cdot3} = 64$, $2^{2\cdot5} = 1024$ and $2^{2\cdot7} = 16384$ possible combinations of the sequence $V_0^{H-1}$. Of course not all the input combinations are possible, e.g., considering three possible input combinations (see Table 1) and horizons $H = 3, 5, 7$ there exist $3^3 = 27, 3^5 = 243, 3^7 = 2187$ feasible control sequences, and many of them lead to non-optimal solutions.

### 4.2 Cost-function Selection

The goal is to optimally control the temperature of two furnaces to the desired values $x_f = [1/4,\ 1/8]^T$, minimizing the following quadratic performance index:

$$J = \|x(k+H|k) - x_f\|_P^2 + \sum_{h=0}^{H-1} \|x(k+h|k) - x_f\|_Q^2, \tag{14}$$

where $h$ represents the prediction step, the horizon was set to $H = 3, 5, 7$, respectively, and the weighting matrices were set to $Q = P = \left[\begin{smallmatrix} 5 & 0 \\ 0 & 10 \end{smallmatrix}\right]$.

### 4.3 Results

A control algorithm was tested with a simulation using MATLAB 6.1 on a Pentium 4, 2.4-GHz machine. The tree of evolution was calculated during each time sample, resulting in an optimal control sequence that brings the system from the current (initial) state $x(k)$ as close as possible to the desired state $x_f$ in the finite horizon.

**Table 2** The complexity and time efficiency

| Horizon | CPU time (200 steps) | Avg. CPU time per sample | Average number of explored nodes | Maximum number of nodes in the tree | Comparison: CPU time by MIQP[a] | CPU time by MIQP (CPLEX) |
|---|---|---|---|---|---|---|
| $H = 3$ | 2.5 s | 0.0125 s | 12 | 27 ($2^{m_b H} = 64$) | 37.5 s | 3.5 s |
| $H = 5$ | 6.4 s | 0.032 s | 36 | 243 ($2^{m_b H} = 1024$) | 252 s | 8.6 s |
| $H = 7$ | 14.8 s | 0.074 s | 91 | 2187 ($2^{m_b H} = 16384$) | 1252 s | 19.7 s |

[a] http://control.ee.ethz.ch/~hybrid/miqp/

Only the first input vector was applied to the system. The complexity and the time efficiency of the proposed model-predictive-control approach for all the horizons are given in Table 2, where the results are given for $k_{\max} = 200$, i.e., for the time interval $t = [0,\ 16]$ s. The shown CPU times are the times for solving the stated model-predictive-control problem by online optimization. The solution was not calculated offline as a parametric solution but the optimization problem was solved for a single point in the state space at each time sample. Calculation times for the reachability-based approach are shown and for comparison, the time needed to calculate the equivalent solutions using the MIQP [5] is given. Two MIQP solvers were used, a solver written in Matlab, and a commercial CPLEX solver. It is clear that the speed of the calculation strongly depends on the solver implementation. Nevertheless, the proposed reachability-based approach performs better, despite being implemented in Matlab script.

The average number of explored tree-of-evolution nodes per sample and the maximum number of nodes are also shown in the table. The number of explored nodes is significantly smaller in all cases, which shows the effect of pruning the imperspective branches of the tree. As a result, the average CPU time per sample remains small

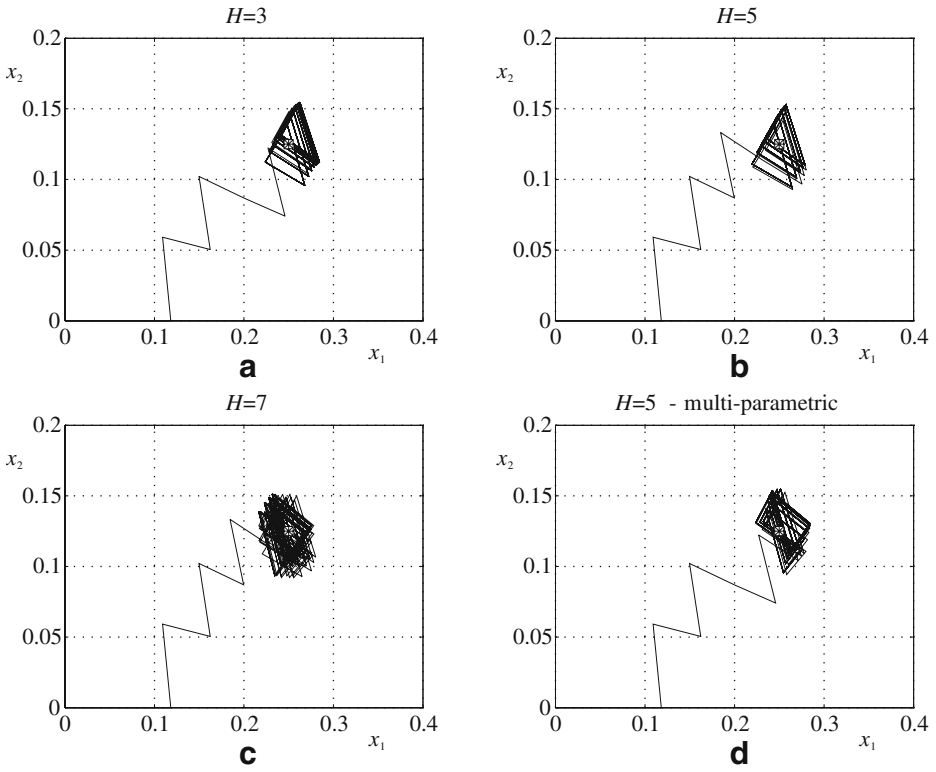



**Fig. 3** Temperature trajectory for the furnaces using the model predictive controller

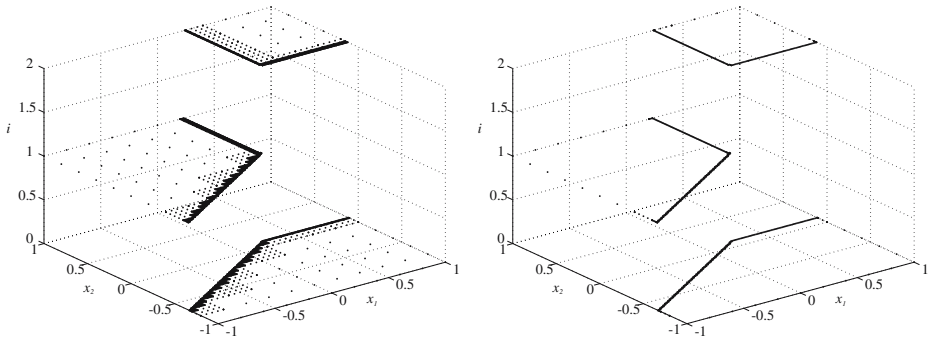
enough, i.e., smaller than the sample time, to enable the implementation of the control in real time for the given horizons. It should be noted, however, that for the longer horizons this is not the case anymore.

The resulting optimal trajectories are shown in Fig. 3a,b and c for a horizon $H = 3, 5$ and 7, respectively.

### 4.4 Multi-parametric Approach Using PNNs

The solution of the multi-parametric approach is the form Eq. 9. In the case of the alternate heating of two furnaces the temperature of the two furnaces should be controlled to a given set point by applying only one of the three possible discrete inputs to the system (see Table 1). This means that the multi-parametric solution has three different control laws, i.e., $G_0^0 = [0\ 1]^T$, $G_1^0 = [1\ 0]^T$ and $G_2^0 = [0\ 0]^T$, and therefore three different prototype classes. To be able to train the PNN, the corresponding regions in the state space have to be identified. The goal is to determine the prototype classes, i.e., the pairs $(x(0), G_i^0)$ where the states $x(0)$ lie as close as possible to the border with other classes, in as few simulation runs as possible.

Considering we do not know anything about the position and the shape of the partitions $P_i^0$, we start to search the border (initial) states $x(0)$ over the whole state space $(x_1, x_2)$ using a sequential approach. The control law is calculated in every step and the border is detected when the control law is changed between two calculation steps. The state space was first searched using the initial (larger) step. Then in the smaller border areas (the areas where the border was detected) the procedure was repeated using a smaller step. The procedure was repeated until the obtained information about the borders was exact enough. The search procedure used the reachability-based model-predictive-control approach, presented earlier, with the horizon set to $H = 5$. Figure 4a shows the initial states $x(0)$ used during the simulation and the corresponding control laws $G_i^0$. In the final set of prototype classes only those pairs $(x(0), G_i^0)$ that lie on the border (see Fig. 4b) were included. In this way the size of the PNN is reduced and the performance of the PNN is improved.




**a** Simulation states $x(0)$ with appropriate control laws $G_i^0$ (classes)

**b** Border states $x(0)$ with appropriate control laws $G_i^0$ (classes)

**Fig. 4** Simulation states (**a**) and border states (**b**) used for learning PNN

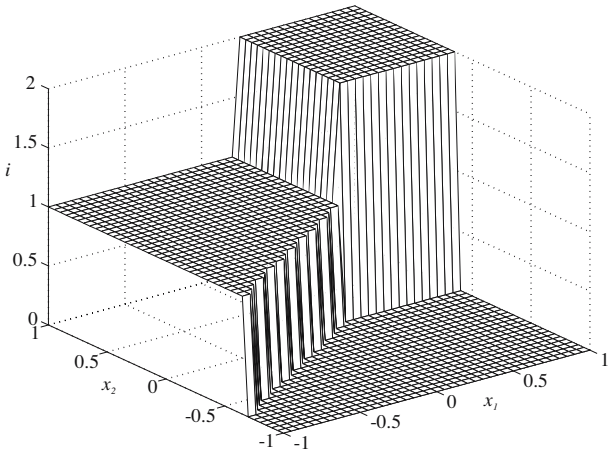



**Fig. 5** The output function of the PNN

The time used for training the PNN with the data presented in Fig. 4b was 0.43 s. The result is the PNN, which classifies the initial state $x(k)$ into the classes $i = 0, 1, 2$, and thus selects an appropriate control law $G_i^0$. The output function of the obtained PNN is shown in Fig. 5. The result of the predictive control using the trained PNN, for the time interval $t = [0, 16]$ s, was obtained in 1.3 s and is shown in Fig. 3d. In comparison with the result shown in Table 2 for $H = 5$ this is approximately five times faster. At this point we should point out that the time needed for the preparation of the PNN, i.e., searching for the prototype pairs and training the PNN, was 91.3 s. This is performed only at the beginning (offline), later in the control loop only the trained PNN is used. As a consequence, the longer prediction horizons can also be used for real-time control.

## 5 Conclusions

The model-predictive-control approach for hybrid systems with discrete inputs only was discussed. Obviously, the complexity of the model-predictive-control problem grows extremely rapidly with the number of binary (discrete) variables; therefore, a special approach is needed.

In the paper a model predictive approach is proposed where the optimization problem associated with the prediction is solved by abstracting the behaviour of the hybrid system into a 'tree of evolution'. The main advantage of this approach is that the tree can be cut from both sides, top and bottom, which results in a considerable reduction in the size of the tree. To further reduce the online computational burden the solution using a multi-parametric approach is presented, where a PNN is used to capture the partition of the state space. The learning is performed offline, while during the operation of the controller the system state is time-efficiently classified into a class that selects an appropriate control law.

The efficiency of the proposed algorithm, including the PNN-based multi-parametric approach, was demonstrated on the example of the alternate heating of two furnaces. The obtained results are comparable to the results obtained in

Hedlund and Rantzer [16] and Bemporad and Morari [6]. Future research will focus on exploring the possibilities of real applications of the proposed approach.

## Appendix: HYSDEL Code of a Two Furnace System ([6])

```
SYSTEM furnace {
INTERFACE {
  STATE {
    REAL x1 [-1, 1];     /* temperature furnace 1 */
    REAL x2 [-1, 1];}    /* temperature furnace 2 */
  INPUT {
    BOOL heat1, heat2;} /* binary  inputs */
  PARAMETER {
    REAL Ts=0.08;        /* sample time */
    REAL b1=1-exp(-Ts); /* constants of discretization */
    REAL a1=exp(-Ts);
    REAL b2=(1-exp(-2*Ts))/2;
    REAL a2=exp(-2*Ts);
    REAL u0=0.8;}}       /* fixed amount of energy */
IMPLEMENTATION {
  AUX {
    REAL u1, u2;}        /* aux. variables */
  DA {
    u1={IF heat1 THEN b1*u0};
    u2={IF heat2 THEN b2*u0};}
  CONTINUOUS {
    x1=a1*x1+u1;
    x2=a2*x2+u2;}
  MUST {
    ~heat1|~heat2;}}
}
}
```

## References


1. Baotić, M., Christophersen, F.J., Morari, M.: A new algorithm for constrained finite time optimal control of hybrid systems with a linear performance index. In: European Control Conference. University of Cambridge, 2003
2. Barton, P.I., Banga, J.R., Galán, S.: Optimization of hybrid discrete/continuous dynamic systems. Comput. Chem. Eng. **24**, 2171–2182 (2000)
3. Bemporad, A.: Efficient conversion of mixed logical dynamical systems into an equivalent piecewise affine form. IEEE Trans. Automat. Contr. **49**(5), 832–838 (2004)
4. Bemporad, A., Giovanardi, L., Torrisi, F.D.: Performance driven reachability analysis for optimal scheduling and control of hybrid systems. In: Proceedings of the 39th IEEE Conference on Decision and Control, pp. 969–974. IEEE, Sydney (2000)
5. Bemporad, A., Morari, M.: Control of systems integrating logic, dynamics, and constraints. Automatica **35**(3), 407–427 (1999)
6. Bemporad, A., Morari, M.: Optimization-based hybrid control tools. In: Proceedings of the American Control Conference, pp. 1689–1703. IEEE, Arlington (2001)
7. Bemporad, A., Morari, M., Dua, V., Pistikopoulos, E.N.: The explicit linear quadratic regulator for constrained systems. Automatica **38**(1), 3–20 (2002)

8. Bemporad, A., Torrisi, F.D., Morari, M.: Optimization-based verification and stability characterization of piecewise affine and hybrid systems. In: Hybrid systems: Computation and Control, Lecture Notes in Computer Science, vol. 1790, pp. 45–58. Springer, New York (2000)
9. Borrelli, F.: Constrained optimal control of linear and hybrid systems. In: Lecture Notes in Control and Information Sciences, vol. 290. Springer, New York (2003)
10. Borrelli, F., Baotić, M., Bemporad, A., Morari, M.: An efficient algorithm for computing the state feedback solution to optimal control of discrete time hybrid systems. In: Proceedings of the American Control Conference, pp. 4717–4722. Denver, 2003
11. Borrelli, F., Baotić, M., Bemporad, A., Morari, M.: Dynamic programming for constrained optimal control of discrete-time linear hybrid systems. Automatica **41**(10), 1709–1721 (2005)
12. Cassandras, C.G., Pepyne, D.L., Wardi, Y.: Optimal control of a class of hybrid systems. IEEE Trans. Automat. Contr. **46**(3), 398–415 (2001)
13. Dua, V., Bozinis, N.A., Pistikopoulos, E.N.: A multiparametric programming approach for mixed-integer quadratic engineering problems. Comput. Chem. Eng. **26**(4–5), 715–733 (2002)
14. Ferrari-Trecate, G., Cuzzola, F.A., Mignone, D., Morari, M.: Analysis of discrete-time piecewise affine and hybrid systems. Automatica **38**(12), 2139–2146 (2002)
15. Gokbayrak, K., Cassandras, C.G.: A hierarchical decomposition method for optimal control of hybrid systems. In: Proceedings of the 38th IEEE Conference on Decision and Control, pp. 1816–1821. Phoenix, 1999
16. Hedlund, S., Rantzer, A.: Optimal control of hybrid systems. In: Proceedings of the 38th IEEE Conference on Decision and Control, pp. 3972–3976. Phoenix, 1999
17. Heemels, W.P.M.H., De Schutter, B., Bemporad, A.: Equivalence of hybrid dynamical models. Automatica **37**(7), 1085–1091 (2001)
18. Kerrigan, E.C., Mayne, D.Q.: Optimal control of constrained, piecewise affine systems with bounded disturbances. In: Proceedings of the 41st IEEE Conference on Decision and Control. Las Vegas, Nevada, USA (2002)
19. Lincoln, B., Rantzer, A.: Optimizing linear system switching. In: Proceedings of the 40th IEEE Conference on Decision and Control, pp. 2063–2068. Orlando, 2001
20. Morari, M., Baotic, M., Borrelli, F.: Hybrid systems modeling and control. Eur. J. Control **9**(2–3), 177–189 (2003)
21. Pepyne, D.L., Cassandras, C.G.: Optimal control of hybrid systems in manufacturing. Proc. IEEE **88**(7), 1108–1123 (2000)
22. Potočnik, B., Bemporad, A., Torrisi, F.D., Mušič, G., Zupančič, B.: Hysdel modeling and simulation of hybrid dynamical systems. In: 4th MATHMOD Vienna, Proceedings (ARGESIM Report, no. 24). Vienna University of Technology, Vienna (2003)
23. Potočnik, B., Bemporad, A., Torrisi, F.D., Mušič, G., Zupančič, B.: Hybrid modelling and optimal control of a multiproduct batch plant. Control Eng. Pract. **12**(9), 1127–1137 (2004)
24. Potočnik, B., Mušič, G., Zupančič, B.: A new technique for translating discrete hybrid automata into piecewise affine system. Math. Comput. Model. Dyn. Syst. **10**(1), 41–57 (2004)
25. Rantzer, A., Johansson, M.: Piecewise linear quadratic optimal control. IEEE Trans. Automat. Contr. **45**(5), 629–637 (2000)
26. Sontag, E.D.: Nonlinear regulation: the piecewise linear approach. IEEE Trans. Automat. Contr. **26**(2), 346–358 (1981)
27. Specht, D.: Probabilistic neural networks. Neural Netw. **3**, 109–118 (1990)
28. Torrisi, F.D., Bemporad, A.: HYSDEL—a tool for generating computational hybrid models. IEEE Trans. Control Syst. Technol. **12**(2), 235–249 (2004)