FURNACES CONTROL USING MPC HYBRID

Miroslav Makýš and Štefan Kozák

Slovak University of Technology, Faculty of Informatics and Information Technology, Faculty of Electrical Engineering and Information Technology
Ilkovičova 3, 812 19 Bratislava, Slovak Republic
Tel.: +421 2 60291281 e-mail:makys@fiit.stuba.sk, kozak@stuba.sk

INTRODUCTION

Since the MPC control method is computationally intensive and therefore not useful for rapid and complex systems several authors in their articles devoted to the creation of an explicit control law. The advantage of this approach is that all the calculations that would otherwise be done online are done in advance and to control one only selects the optimal control action based on the current state.

One of the options was proposed by Bemporad in the publication [5], where by combination of reachability analysis and multiparametric linear programming is obtained control law in from

$$u(t) = F_i \begin{bmatrix} x(t) \\ r(t) \end{bmatrix} + g_i$$

$$\begin{bmatrix} x(t) \\ r(t) \end{bmatrix} \in \Omega_i = \{ \begin{bmatrix} x \\ r \end{bmatrix} : H_i \begin{bmatrix} x \\ r \end{bmatrix} \leq S_i \}, i = 1,2,....,s$$

where

$\bigcup_{i=1}^{s} \Omega_i$	is set of states and reference values for which the solution of control problem is feasible,
$u(t)$	is vector of input variables,
$x(t)$	is vector of continuous state variables,
$r(r)$	is vector of discrete state variables,
F_i	is matrix of appropriate dimensions,
G_i	is vector of appropriate length,
s	is number of partitions state space is divided to.

After determining the optimal control law formula its use in practice is very quick. However the actual calculation of complex tasks can take too much time.

Potočnik et al. dealt with the creation of an explicit controller for systems with only binary inputs in publication [4].

The algorithm described in this publication is based on reachability analysis which is driven by value of performance function. The algorithm simulates the behaviour of the hybrid system by building a diagram - tree. Nodes of tree represent system states (the top is initial position), the edges represent input. Each tree node is assigned to a corresponding value of performance function on the basis of which next searching through the tree is done. To reduce the computational complexity, non-perspective branches that have very high value of performance function are cut off. When the entire tree is searched we know the optimal sequence of inputs for a given initial state. We apply the first calculated input and repeat the process for a new state, so we do MPC philosophy.

PROBLEM COMPLEXITY

Solution of the control problem with H steps prediction in each step is a sequence of inputs $V_0^{H-1} = \{v(0),...,v(H-1)\}$ where $v(i)$ represent discrete input in the time i. If the system has m discrete inputs and each input can take n values, it is necessary to examine n^{m*H} possibilities in each step. It follows that it is a problem with exponential complexity - the time needed to solve the problem increases exponentially with the size of the problem.

OPTIMIZATION BASED ON REACHABILITY ANALYSIS

With the reachability analysis can be found all states to which the system can get from the initial state. Of course, search for all states would be unnecessary loss of time because many of them do not lead to required state. It is effective to combine reachability analysis with detection of states which do not lead to the optimal solution and ignore them. In this case we used branch and bound method which searches through the tree and cuts off non-perspective branches.

Tree of evolution shown in figure Fig.1 is a structure representing the evolution of system states when system is examined by reachability analysis. Tree nodes represent system states that are reachable from the initial state - the initial state is the top of the tree. Oriented edges represent input of system, input of edge is a state in which we apply the input, output of edges is the state after input is applied. Each node is also assigned to the value of performance function that is used to detect non-perspective states.

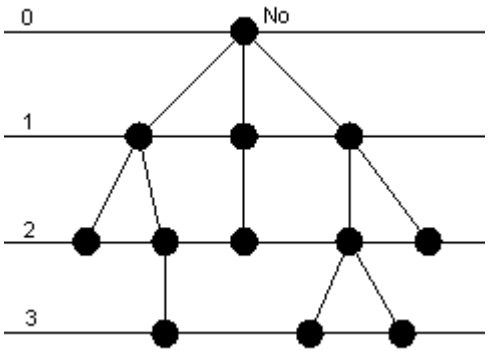

Fig 1: Tree of evolution

Before the reachability analysis is started initialization is done, which means that the top of the tree is created corresponding to the initial state and the optimal value of performance function is set to infinity.

Evolution of tree is realized so that among unexplored tree leaves is always selected the one which has the smallest value of performance function. The system (which is in a state corresponding to the selected leaf) is simulated with all possible inputs and the leaf is marked as explored. After simulation each of inputs performance function is evaluated. If value of performance function is smaller than current optimal value new leaf is added to the tree. This new leaf is connected to previous node with edge which determines input to previous state applied in previous step. If new node depth is equal to prediction horizon (we made a prediction for the required number of steps) the optimal value of performance function is set to this value of this node, node status is marked as optimal and node is marked to be examined. The evolution of the tree continues until all nodes are examined. At the end, therefore, we know the optimal sequence of H inputs which drives the system from the initial state as close as possible to the reference state so that the value of the performance function is minimal.

A very important parameter of algorithm is its time efficiency. We strive algorithm to occupy shortest time possible. Therefore, when selecting node to be further explored it is convenient to take as a criterion value of performance function and then examine the node with the smallest one to avoid wasting time with nodes that do not lead to an optimal solution. When we reach the required number of prediction steps, the optimal value of performance function is changed from infinity to the value pertaining to the last node added. This enables us to cut off the number of nodes which will definitely not lead to an optimal solution - there are all those whose performance function value is greater than the optimum. In fact very important is selection of performance function which must be depending on the number of steps monotonically increasing - if once exceeds a certain threshold, it certainly does not fall under it. Since the commonly used performance function

$$J = \left\|Q_{xN}(x(N)-n_r)\right\|_2 + \sum_{k=1}^{N-1}\left\|Q_x(x(k)-x_r)\right\|_2 + \sum_{k=0}^{N-1}\left\|Q_u(u(k)-u_r)\right\|_2 + \left\|Q_z(z(k)-z_r)\right\|_2 + \left\|Q_y(y(k)-y_r)\right\|_2$$

where N is prediction horizon,

$x(k)$ is predicted state in time k,

$u(k)$ is input in time k,

$z(k)$ is predicted value of auxiliary variable in time k,

$y(k)$ is predicted output in time k,

n_r, x_r, u_r, z_r, y_r are reference values of state, input, auxiliary variable z and output,

is the sum of squares of deviations, i.e. the sum of non-negative values, it satisfies required property.

When looking for explicit control law using multiparametric programming we get solution in form

$$u(t) = F_i \begin{bmatrix} x(t) \\ r(t) \end{bmatrix} + g_i .$$

In case when all the inputs are discrete the formula gets a bit simpler

$$u(t) = G_i$$

$$X(k) \in P_i^0$$

where P_i^0 is area states with the same optimal input.

IDENTITYING OF STATE AREAS

The next step is to identify the states that have the same optimal input. One possibility would be to determine optimal entry points for the entire state network and on that basis determine the boundaries of areas, such as in figure Fig.2.

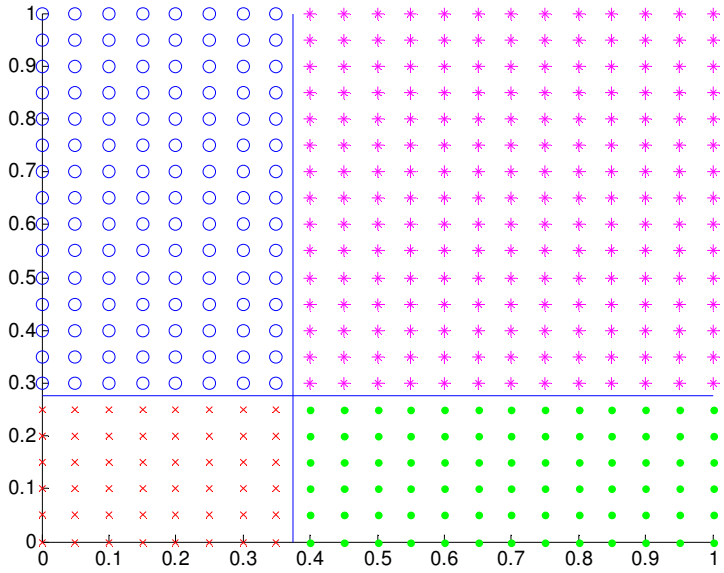

Fig. 2: Determining of states areas with the same optimal input

However this is very inefficient way because that is looking for the optimal sequence of inputs for a number of points unnecessarily.

To determine areas was chosen following philosophy:

If there is rectangular part of the state space which threshold of two or more areas passes through then vertices of this rectangular part do not have the same optimal input. And vice-versa if all vertices of rectangular part have the same optimal input then through this part does not pass the threshold of two areas – there is one optimal input for all states in the part.

The first rectangular part of the state space we want to verify if there is a threshold passing through it is whole state space defined by the upper and lower constraints for each state variable. In case of two-state system, we need therefore to determine the optimal input for 4 points. If these inputs are not equal it is necessary to divide the examined part into smaller parts and go on until rectangular parts small enough that we know the boundaries of areas with sufficient accuracy. The boundaries of areas that have the same optimal input are given by all vertices of the smallest examined parts through which threshold passes. Example of division of the state space is shown in figure Fig.4.

ASSIGNING STATE TO AREA

When the state space is partitioned, it is important how we determine which area current state belongs to. The number of areas P_i^0 is equal to the number of optimal discrete inputs. In general these areas can be complex and may be nonconvex. Therefore the next task is to assign the current state to one area and determine the optimum input. This must be done in real time. One solution to this problem is to use probabilistic neural network (PNS). PNS are appropriate to address classification of problems. PNS defines the distance (probabilistic function) input from the class of prototypes x_j^i, where i is the index of the class a $j=1, 2, ..., Ni$, where Ni is the number of prototypes in class i. Probabilistic function is defined:

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

and probability for class i is:

$$Pi(x) = \frac{L_i(x)}{\sum_{j=1}^{M} L_j(x)}$$

where M is number of classes,

d is space dimension.

Once we know boundaries of particular areas we can train probabilistic neural network and use it to determine optimal input.

DISADVANTAGE OF DISCRETE INPUTS

One big limitation of this approach is the discretion of inputs. Most of controlled systems does not have naturally discreet all inputs and to achieve enough accurate control it is necessary to calculate the exact value of inputs, not just select one of the options. Therefore, such control may not be fully accurate and values of controlled states will oscillate around reference values as shown in figure Fig.3.

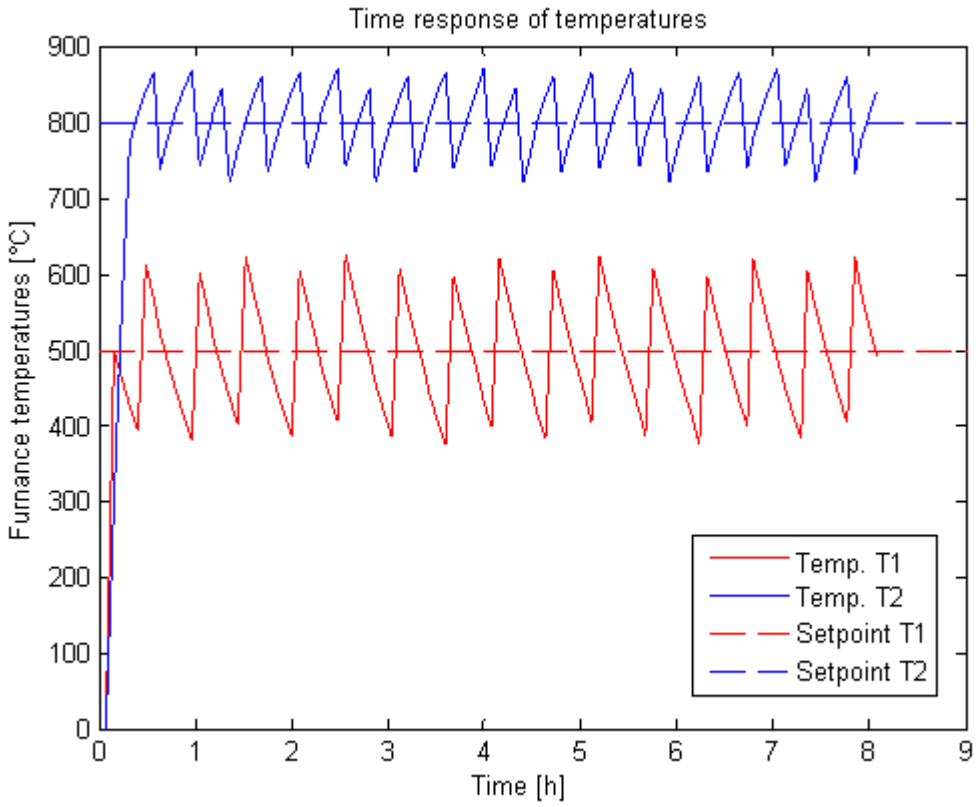

Fig. 3: Time response of finances temperatures

We have stabilized system states in the range of values that is closer to reference values than initial range and we have used just small range of inputs. System state is thus stabilized to some range by using a range of input variables. This can be used to make control more precise by optimizing input values for stabilized range of state and input values. This can be repeated until the desired accuracy is achieved. On figure Fig.4 is depicted decomposition of state space after the first optimization and on figure Fig.5 is depicted decomposition of state space after control refinement.

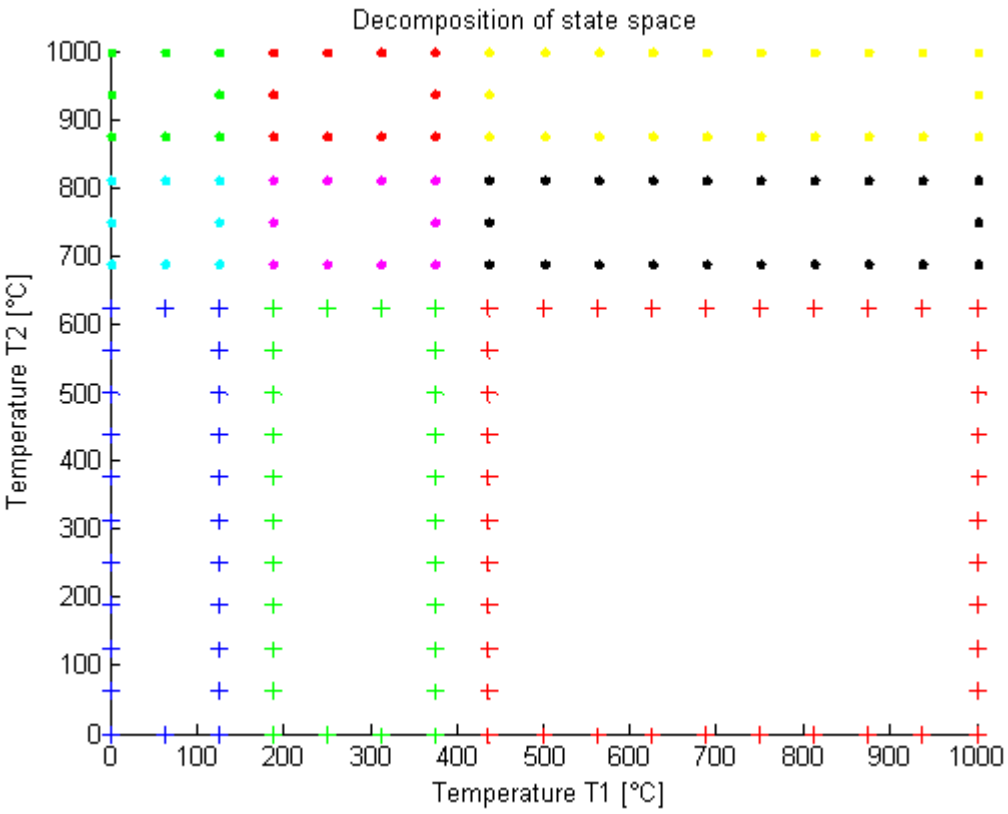

Fig. 4: Decomposition of state space

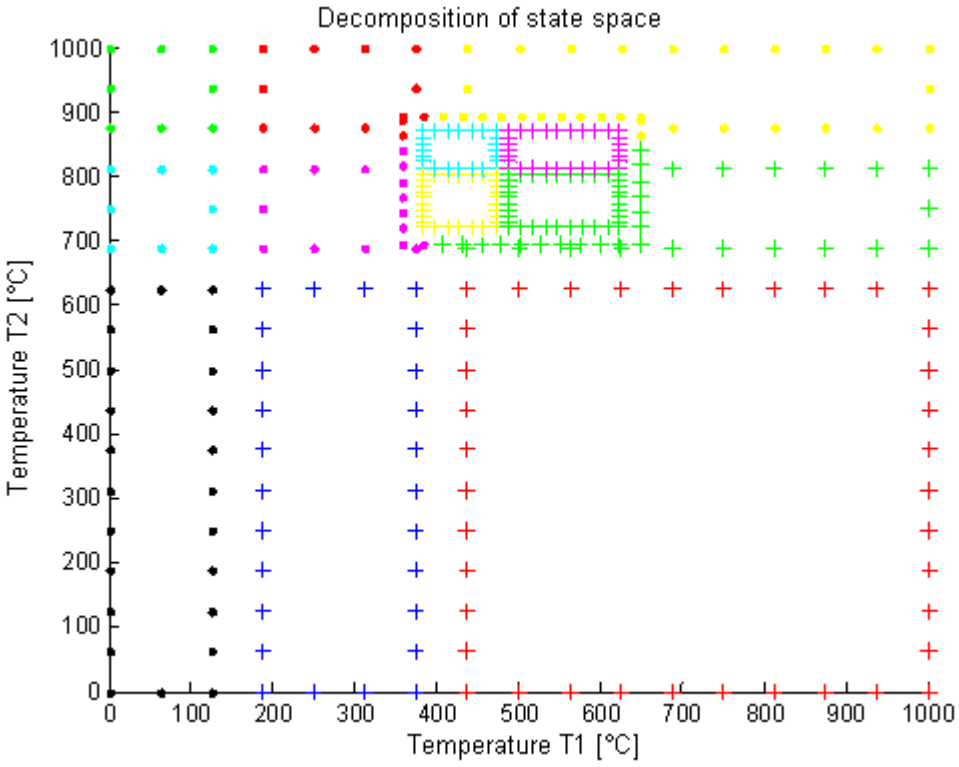

Fig. 5: Decomposition of state space

On figure Fig.6 is depicted time response of furnaces temperatures.

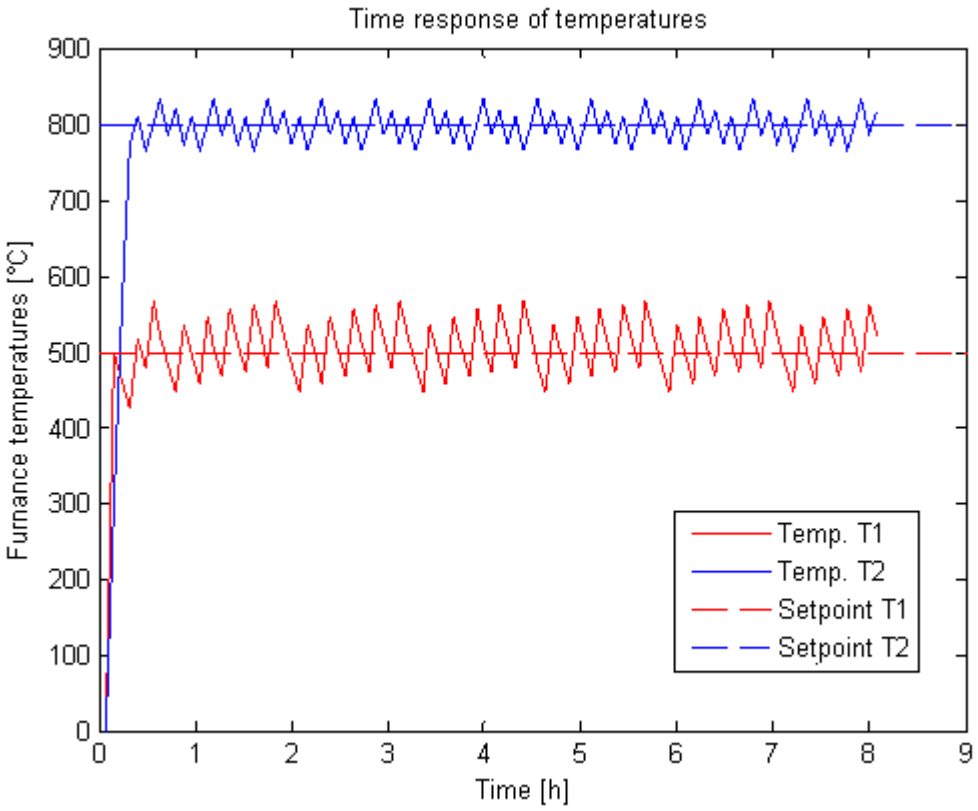

Fig. 6: Time response of furnaces temperatures

IMPROVEMENT OF ALGORITHM

Although the branch and bound method allows us to cut off many branches which would certainly not lead to an optimal solution, for big problems is this algorithm still very time consuming because of the many options that it has to search. The fact that one branch does not lead to an optimal solution, we know just after the value of performance function at its end exceeds the optimum value of the criterion function. Our aim is therefore to discover that the branch does not lead to optimal solution as soon as possible.

When two different positive numbers are raised by the same number its difference is bigger. And the higher the number is the bigger the difference will be. Assuming that the evolution of the system leads to the reference state (which should be fulfilled), the increase in value of performance function should be smaller and smaller. It follows that if we do not use the sum of squares of deviations as performance function, but the sum of power of bigger numbers of deviation, the time required to implement the algorithm is reduced significantly. This way we get another (it is questionable whether the worse) results.

The big disadvantage of this algorithm remains that in this way we manage the system to only one reference value.

CONTROL OF TWO FURNANCES TEMPERATURES

Described hybrid systems control approach we have tested on a theoretical example where we try to control the temperatures of two furnaces to the desired value. Behaviour furnace is described by the following equations:

if x_1 <= 300

$$x_1(k+1) = 0.92 * x_1(k) + 0.5 * u_1(k)$$

else

$$x_1(k+1) = 0.78 * x_1(k) + 0.5 * u_1(k)$$

if x_2 <= 350

$$x_2(k+1) = 0.85 * x_2(k) + 0.3 * u_2(k)$$

else

$$x_2(k+1) = 0.72 * x_2(k) + 0.3 * u_2(k)$$

$$0 \le u_1 \le 1000$$

$$0 \le u_2 \le 1000$$

where x_1, x_2 are furnaces temperatures,

u_1, u_2 are furnaces inputs.

This furnaces model takes into account that higher furnace temperature is the faster is furnace cold down. Sampling period was selected 7min and goal was to regulate the temperature of the first furnace x_1 to 500 ° C and the second furnace temperature x_2 to 800 ° C.

The simulation results and course of inputs is depicted on figures Fig. 7 and Fig.8. These results were achieved after six optimization iterations, where goal was control deviation less than 1%. Table Tab. 1 includes dependence of time taken to compute control algorithm of used performance function.

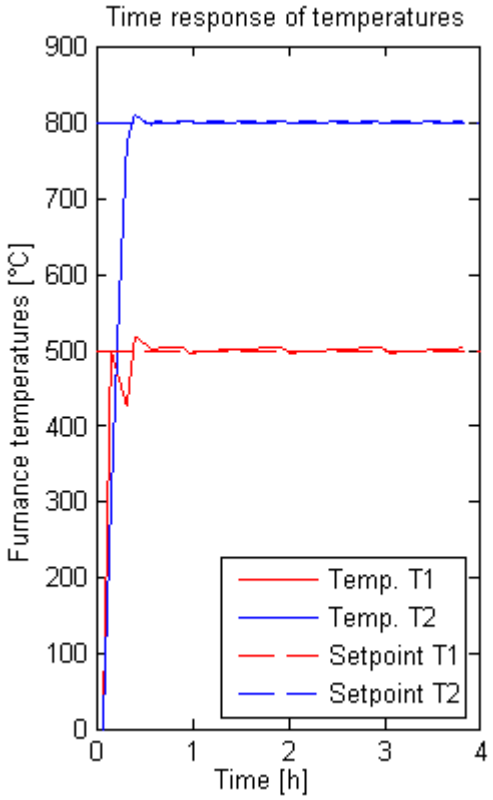

Fig. 7: Time response of furnaces temperatures

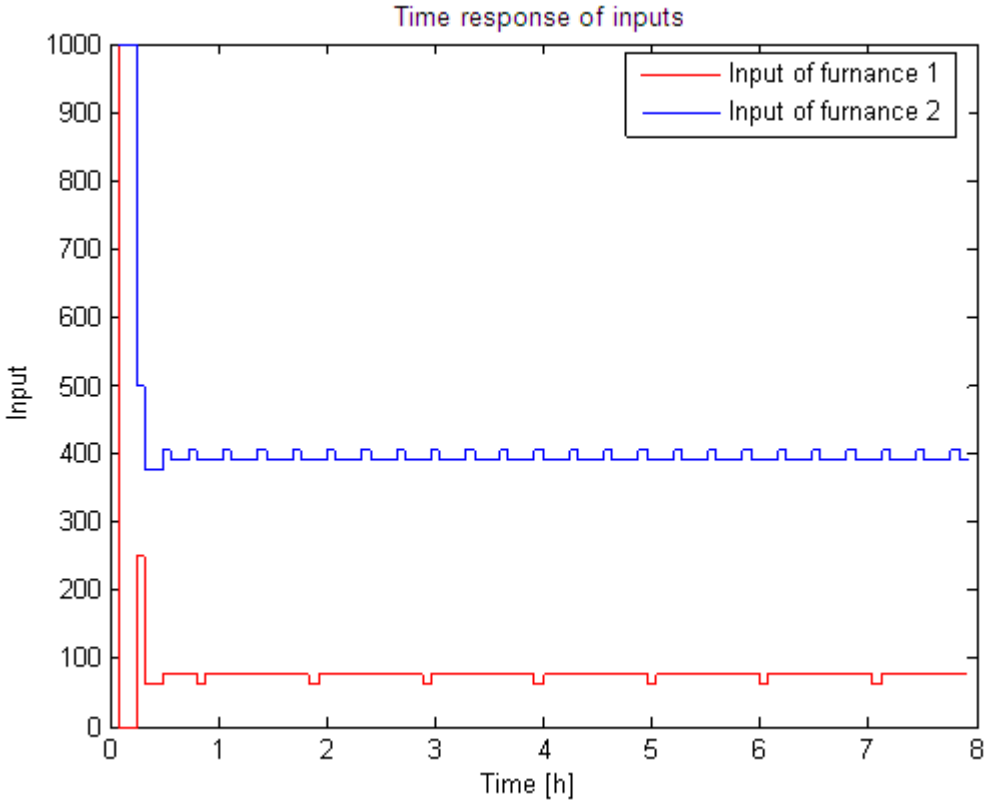

Fig. 8: Time response of inputs

		Prediction horizon		
		2	3	4
Performance function norm	2	80,3 s	160,1 s	336,5 s
	4	80,1 s	148,7 s	271 s
	6	79,5 s	146,8 s	251 s

Tab. 1: Dependence of time taken to compute control algorithm of used performance function and prediction horizon

CONCLUSION

The advantage of examining approach to system control is that to determine an explicit control law it is not necessary to address the complex problem of multi-parametric programming. For discrete inputs is less time-consuming to search through the tree of evolution and on-line implementation of controller is also quick.

This project was supported by grant of VG 1/0804/08 (VEGA) project.

REFERENCES

[1] BEMPORAD, A.: HYBRID TOOLBOX for real-time applications. 2008. http://www.dii.unisi.it/cgi-bin/hybtbx_download.cgi?getpaper&paper=HybTbxWin

[2] TORRISI, F.D. et al.: HYSDEL 2.0.5 – User Manual. 2002. http://www.dii.unisi.it/cgi-bin/hybtbx_download.cgi?getpaper&paper=HybTbxWin

[3] TORRISI, F.D., BEMPORAD, A.: HYSDEL – A tool for generating computational hybrid models.2004. http://www.dii.unisi.it/cgi-bin/ab_download.cgi?getpaper&paper=TB04a

[4] POTOČKIN, B et al.: Model-base Predictive Control of Hybrid Systems: A Probabilistic Neural-network Approach to Real-time Control. 2006. http://msc.fe.uni-lj.si/Papers%5CJIRS_Potocnik.pdf

[5] BEMPORAD, A.: The explicit linear quadratic regulator of constrained systems. 2006. http://www.dii.unisi.it/~bemporad/

[6] BAK, T., IZADI-ZAMANABADI, R.: Lecture Notes – Hybrid Systems. 2004. http://www.control.aau.dk/~raf/hybrid/hs.pdf

[7] TOMLIN, C.J.: AA278A Lecture 1: Introduction. 2005. http://www.stanford.edu/class/aa278a/lecture1.pdf

[8] BEMPORAD, A., BORRELLI, F., MORARI, M.: Piecewise Linear Optimal Controllers for Hybrid Systems. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=876688

[9] BEMPORAD, A., MORARI, M.: Control of systems integrating logic, dynamics, and constraints. 1998. http://citeseerx.ist.psu.edu/icons/pdf.gif;jsessionid=857E713B9EE9DAE8B0D8CB3B9EE98FDE