Efficient Evaluation of Piecewise Control Laws defined over a Large Number of Polyhedra

Frank J. Christophersen, Michal Kvasnica, Colin N. Jones, and Manfred Morari
Automatic Control Laboratory, ETH Zurich, ETL K13.1, CH – 8092 Zurich, Switzerland
fjc | kvasnica | cjones | morari@control.ee.ethz.ch

***Abstract*— We consider the class of piecewise state feedback control laws applied to discrete-time systems, motivated by recent work on the computation of closed-form MPC controllers. The on-line evaluation of such a control law requires the determination of the state space region in which the measured state lies, in order to decide which 'piece' of the piecewise control law to apply. This procedure is called the point location problem, and the rate at which it can be solved determines the minimal sampling time of the system. In this paper we present a novel and computationally efficient search tree algorithm utilizing the concept of bounding boxes and interval trees that significantly improves this point-location search for piecewise control laws defined over a large number of (possibly overlapping) polyhedra. Furthermore, the required off-line preprocessing is low and so the approach can be applied to very complex controllers. The algorithm is compared with existing methods in the literature and its effectiveness is demonstrated for large examples.**

***Keywords*— constrained systems, discrete-time systems, point location problem, set membership test, explicit control, hybrid systems, piecewise affine systems, multi-parametric programming, receding horizon control, MPC.**

I. Introduction

In this paper we consider the *point-location* or *set membership problem* [28] for the class of discrete-time control problems with linear state and input constraints for which an explicit time-invariant piecewise state feedback control law over a set of possibly overlapping polyhedral regions is given. The point-location problem comes into play on-line when evaluating the control law. One must identify the state space region in which the measured state lies at the current sampling instance. As the number of defining regions grows, a purely *sequential search* (also known as *exhaustive search*) through the regions is not sufficient to achieve high sampling rates. Hence, it is important to find an efficient on-line search strategy in order to evaluate the control action 'in time' without the need of a heavy additional memory and preprocessing demand.

This work is motivated, but not limited, by the recent developments in the field of controller synthesis for hybrid systems [31], [15], [29], [5], [19]. A significant amount of the research in this field has focused on solving constrained optimal control problems, both for continuous-time and discrete-time hybrid systems. We consider the class of constrained discrete-time *piecewise affine* (PWA) systems [29] that are obtained by partitioning the extended state-input space into polyhedral regions and associating with each region a different affine state update equation.

For piecewise affine systems the *constrained finite time optimal control* (CFTOC) problem can be solved by means of multi-parametric programming [5], [7], [1], [21] and the resulting solution is a time-varying piecewise affine state feedback control law. If the solution to the CFTOC problem is used in a *receding horizon control* [26], [23] strategy (or *model predictive control* (MPC)) the time-varying PWA state feedback control law becomes time-invariant and can serve as a control 'look-up table' on-line, thus enabling receding horizon control to be used for fast sampled systems. However, due to the combinatorial nature of the problem the number of state space regions over which the control look-up table is defined grows in the worst case exponentially [5], [4] and therefore efficient on-line search strategies are required to achieve fast sampling rates.

In this paper we present a novel, computationally efficient algorithm that performs the aforementioned point-location search for *general* closed-form piecewise (possibly nonlinear) state feedback control laws defined over a finite number of polyhedra or over a finite number of regions for which a bounding box [2] computation is feasible. Moreover, control laws that do not form a polyhedral partition, but are composed of a collection of *overlapping* polytopic sets, are included naturally in the algorithm. The proposed point-location search algorithm offers a significant improvement in computation time at the cost of a low additional memory storage demand and *very low pre-computation* time for the construction of the search tree. This enables the algorithm to work for controller partitions with a large number of regions, which is demonstrated on numerical examples. In order to show its efficiency, the algorithm is compared with the procedure proposed in [30] where a binary search tree is pre-computed over the controller state space partition.

II. Notation

$|\mathcal{I}|$ denotes the cardinality of the discrete set $\mathcal{I}$, $\mathcal{B}^{(d)}$ is the projection of the set $\mathcal{B}$ onto the d-th dimension, and $[z]_d$ refers to the d-th component of some vector z.

III. Point Location Problem

We now consider *arbitrary* discrete-time control problems with a closed-form (possibly nonlinear) time-invariant piecewise state feedback control law of the form

$$\mu(x(t)) := \mu_i(x(t)), \qquad \text{if} \quad x(t) \in \mathcal{P}_i, \tag{1}$$

where $i = 1, \ldots, N_{\mathcal{P}}$. $x(t) \in \mathbb{R}^{n_x}$ denotes the state of the controlled system at time $t \geq 0$, $\mu_i(\bullet) \in \mathbb{R}^{n_u}$ are nonlinear control functions (or oracles), and the sets $\mathcal{P}_i$ are compact and possibly *overlapping*, i.e. there exists $\mathcal{P}_i$ and $\mathcal{P}_j$ with $i \neq j$ such that $\mathcal{P}_i \cap \mathcal{P}_j$ is full-dimensional. Moreover, $\mathcal{P} := \{\mathcal{P}_i\}_{i=1}^{N_{\mathcal{P}}}$ denotes the collection of sets $\mathcal{P}_i$.

In an on-line application the closed-form piecewise control law is $u(t) = \mu(x(t))$, where $u \in \mathbb{R}^{n_u}$ denotes the control input. In order to evaluate the control one needs to identify the state space region $\mathcal{P}_i$ in which the measured state $x(t)$ lies at the sampling instance t, i.e.

Algorithm III.1 (Control evaluation)

```
1. measure the state $x(t)$ at time instance $t$
2. search for the index set of regions $\mathcal{I}$ such that $x(t) \in \mathcal{P}_i$
   for all $i \in \mathcal{I}$
      IF $\mathcal{I} = \emptyset$ THEN problem infeasible STOP
      IF $|\mathcal{I}| > 1$ THEN pick one element $i^\star \in \mathcal{I}$
3. apply the control input $u(t) = \mu_{i^\star}(x(t))$ to the system
4. wait for the new sampling time $t + 1$, goto (1.)
```

The second step in Algorithm III.1 is also known as the *point-location* or the *set membership problem* [28]: in other words, given a point $x \in \mathbb{R}^{n_x}$ and a set of sets $\{\mathcal{P}_i\}_{i=1}^{N_{\mathcal{P}}}$, the goal is to list the set of indices $\mathcal{I}$ such that $x \in \mathcal{P}_i$ for all $i \in \mathcal{I}$.

IV. Constrained Finite Time Optimal Control for Linear Hybrid Systems

An interesting example of control problems where point-location plays an important role is described in the following.

A. Linear Hybrid Systems

Piecewise affine (PWA) systems are equivalent to many other hybrid system classes [29], [16] such as mixed logical dynamical systems [3], linear complementary systems [15], and max-min-plus-scaling systems [10] and thus form a very general class of linear hybrid systems.

Moreover, piecewise affine systems can be used to identify or approximate generic nonlinear systems via multiple linearizations at different operating points [29], [11], [27]. Although hybrid systems (and in particular PWA systems) are a special class of nonlinear systems, most of the nonlinear system and control theory does not apply because it requires certain smoothness assumptions. For the same reason we also cannot simply use linear control theory in some approximate manner to design controllers for PWA systems.

Consider the class of discrete-time, stabilizable, linear hybrid systems that can be described as constrained *piecewise affine* (PWA) systems of the following form

$$\begin{aligned} x(t+1) &= f_{\text{PWA}}(x(t), u(t)) \\ &:= A_d x(t) + B_d u(t) + a_d, \quad \text{if } \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \in \mathcal{D}_d, \end{aligned} \tag{2}$$

where $t \geq 0$, the domain $\mathcal{D} := \cup_{i=d}^{N_{\mathcal{D}}} \mathcal{D}_d$ of $f_{\text{PWA}}(\bullet, \bullet)$ is a non-empty compact set in $\mathbb{R}^{n_x+n_u}$ with $N_{\mathcal{D}} < \infty$ the number of system dynamics, and $\{\mathcal{D}_d\}_{d=1}^{N_{\mathcal{D}}}$ denotes a polyhedral partition of the domain $\mathcal{D}$, i.e. $\mathcal{D}_d := \left\{ \left[\begin{smallmatrix} x \\ u \end{smallmatrix}\right] \in \mathbb{R}^{n_x+n_u} \mid D_d^x x + D_d^u u \leq D_d^0 \right\}$ and $\text{int}(\mathcal{D}_d) \cap \text{int}(\mathcal{D}_j) = \emptyset$ for all $d \neq j$.

Remark IV.1 (Constraints). Note that linear state and input constraints of the general form $C^x x + C^u u \leq C^0$ are naturally incorporated in the description of $\mathcal{D}_d$. □

B. Constrained Finite Time Optimal Control

As an example we define for the aforementioned piecewise affine system (2) the *constrained finite time optimal control* (CFTOC) problem

$$J_T^*(x(0)) := \min_{U_T} \; J_T(x(0), U_T) \tag{3a}$$

$$\text{s.t.} \begin{cases} x(t+1) = f_{\text{PWA}}(x(t), u(t)) \\ x(T) \in \mathcal{X}^f, \end{cases} \tag{3b}$$

where

$$J_T(x(0), U_T) := \ell_T(x(T)) + \sum_{t=0}^{T-1} \ell(x(t), u(t)) \tag{3c}$$

is the *cost function* (also called *performance index*), $\ell(\bullet, \bullet)$ the *stage cost*, $\ell_T(\bullet)$ the *final penalty function*, U_T is *the optimization variable* defined as the input sequence $U_T := \{u(t)\}_{t=0}^{T-1}$, $T < \infty$ is *the prediction horizon*, and $\mathcal{X}^f$ is a compact *terminal set* in $\mathbb{R}^{n_x}$. With a slight abuse of notation, when the CFTOC problem (3a)–(3b) has multiple solutions, i.e. when the optimizer is not unique, $U_T^*(x(0)) := \{u^*(t)\}_{t=0}^{T-1}$ denotes one (arbitrarily chosen) realization from the set of possible optimizers.

The CFTOC problem (3a)–(3b) implicitly defines the set of feasible initial states $\mathcal{X}_T \subset \mathbb{R}^{n_x}$ ($x(0) \in \mathcal{X}_T$) and the set of feasible inputs $\mathcal{U}_{T-t} \subset \mathbb{R}^{n_u}$ ($u(t) \in \mathcal{U}_{T-t}$, $t = 0, \ldots, T-1$), cf. Remark IV.1. In the context of this paper, the goal in this section is to give an explicit (closed form) expression for $u^*(t) : \mathcal{X}_T \to \mathcal{U}_{T-t}$, $t = 0, \ldots, T-1$.

Consider the two following restrictions to the CFTOC problem

Problem IV.2 (PWA system, 1-/∞-norm based cost).

$$\ell(x(t), u(t)) := \|Qx(t)\|_p + \|Ru(t)\|_p, \tag{4a}$$

$$\ell_T(x(T)) := \|Px(T)\|_p, \tag{4b}$$

where $\|\bullet\|_p$ with $p \in \{1, \infty\}$ denotes the standard vector 1-/∞-norm [18], and

Problem IV.3 (Constr. LTI system, quadratic cost).

$$f_{\text{PWA}}(x(t), u(t)) := Ax(t) + Bu(t), \text{ if } \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \in \mathcal{D}, \tag{5a}$$

$$\ell(x(t), u(t)) := x(t)' Q x(t) + u(t)' R u(t), \tag{5b}$$

$$\ell_T(x(T)) := x(T)' P x(T). \tag{5c}$$

In both, CFTOC Problem IV.2 and IV.3, the solution is a time-varying piecewise affine state feedback control law defined over a polyhedral partition, which is stated in the following theorem and proved in e.g. [24], [5].

Theorem IV.4 (Solution to CFTOC). The solution to the optimal control problem (3a)–(3b), restricted to Problem IV.2 or IV.3, is a time-varying piecewise affine function of the initial state $x(0)$

$$\mu_{\text{PWA}}(x(0),t) = K_{T-t,i}\,x(0) + L_{T-t,i}, \quad \text{if} \quad x(0) \in \mathcal{P}_i$$

with $u^*(t) = \mu_{\text{PWA}}(x(0),t)$, where $t = 0,\dots,T-1$, and $\{\mathcal{P}_i\}_{i=1}^{N_\mathcal{P}}$ is a polyhedral partition of the set of feasible states $x(0)$, $\mathcal{X}_T = \cup_{i=1}^{N_\mathcal{P}}\mathcal{P}_i$, with the closure of $\mathcal{P}_i$ given by $\bar{\mathcal{P}}_i = \{x \in \mathbb{R}^{n_x} \mid P_i^x x \le P_i^0\}$. ■

In the case that a *receding horizon* (RH) control policy or a *model predictive controller* (MPC) [26], [23] is used in closed-loop, the control is given as a time-invariant state feedback control law of the form

$$\mu_{\text{RH}}(x(t)) := K_{T,i}\,x(t) + L_{T,i}, \qquad \text{if} \quad x(t) \in \mathcal{P}_i, \tag{6}$$

where $i = 1,\dots,N_\mathcal{P}$ and $u(t) = \mu_{\text{RH}}(x(t))$ for $t \ge 0$. Note that the closed-form receding horizon control law (6) is a special case of control law (1), considered in this paper.

V. Alternative Search Approaches

Due to the combinatorial nature of the CFTOC problem (3a)–(3b), the controller complexity, or the number $N_\mathcal{P}$ of state space regions $\mathcal{P}_i$, can grow exponentially with its parameters in the worst case [5], [4]. Hence, for general control problems, a purely sequential search through the regions is not sufficient in an on-line application. It is therefore important to utilize efficient on-line search strategies in order to evaluate the control action 'in time' without the need of a heavy additional memory demand.

Several authors addressed the point-location/memory storage issue but with moderate success for geometrically complex regions or controllers defined over a large number of regions. A few interesting ideas are mentioned in the following. For the solution to the particular CFTOC Problem IV.2 when, additionally, the system is constrained and linear, i.e.

$$f_{\text{PWA}}(x(t),u(t)) := Ax(t) + Bu(t), \text{ with } \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \in \mathcal{D},$$

the authors in [6] propose a search algorithm based on the convexity of the piecewise affine value function. Even though this algorithm reduces the storage space significantly, the storage demand as well as the search time are still linear in the number of regions. [20] addresses this issue for the same CFTOC problem class by demonstrating a link between the piecewise affine value function of [6] and power diagrams (extended Voronoi diagrams). Utilizing standard Voronoi search methods the search time then reduces to $O(\log(N_\mathcal{P}))$.

To the author's knowledge only two other approaches tackle the more general problem, where the only restriction is that the domain of the control law is a non-overlapping polyhedral partition of the state space. (Note that this is more restrictive than the algorithm presented here, cf. Section I and VI.) [13], [12] aim at pre-computing a minimal polyhedral representation of the original controller partition in order to reduce storage and search complexity. However, the computation is 'practically' limited to a small number of regions with a small number of facets[1] [14], since the pre-computation time grows exponentially. Relaxations to a larger number of regions is possible at the cost of data storage and a higher search complexity.

An alternative approach, which will be used here for comparison, was proposed by Tøndel *et al.* in [30], where a binary search tree is constructed on the basis of the geometric structure of the polyhedral partition[2] by utilizing the facets of the regions as separating hyperplanes to divide the polyhedral partition at each tree level. This however, can lead to a worst case combinatorial number of subdivisions of existing regions and therefore to an additional increase in the number of regions to be considered during the search procedure. The on-line point-location search time is in the best case logarithmic in the number of regions $N_\mathcal{P}$, but worst case linear in the total number of facets, which makes the procedure equivalent to sequential search in the worst case. Moreover, note that the total number of facets, N_F, is typically larger than the original number of regions in the partition, i.e. $N_\text{F} > N_\mathcal{P}$. Although the scheme works very well for polyhedral partitions that have a 'simple' geometric structure and/or have a small number of regions, it is computationally prohibitive in the preprocessing time for more complex controller partitions. This is due to the fact that the first step of the pre-processing is to determine on which side of every facet defining hyperplane each region lies, which requires $2N_\text{F}N_\mathcal{P}$ linear programs, thereby making this method untenable for moderate to large problems, i.e. greater than $10\,000$ regions, cf. Section VIII-C. The memory storage requirement for the binary search tree is (in the worst case) in the order of $n_x N_\text{F}$.

VI. The Proposed Search Algorithm

The proposed search algorithm is based on minimal volume *bounding boxes* $\mathcal{B}_i$ for each region $\mathcal{P}_i$, which are defined as

$$\mathcal{B}_i := \{x \in \mathbb{R}^{n_x} \mid l_i \le x \le u_i\},$$

where the lower and upper bounds l_i and u_i are given by

$$\begin{aligned}(l_i,u_i) := \arg\min_{l,u} \quad & \text{vol}\,(\mathcal{B}(l,u)) \\ \text{subj. to} \quad & \mathcal{B}(l,u) = \{x \in \mathbb{R}^{n_x} \mid l \le x \le u\} \supseteq \mathcal{P}_i.\end{aligned}$$

[1]A facet of a polyhedron $\mathcal{P}$ of dimension n_x is any (n_x-1)-dimensional intersection of $\mathcal{P}$ with a tangent hyperplane.

[2]Even though in the introduction of [30] it is mentioned that overlapping regions and 'holes' in the domain of the controller are handled by the proposed algorithm, these cases are not explicitly treated in the algorithm nor it is directly apparent how this will influence the complexity of the algorithm.

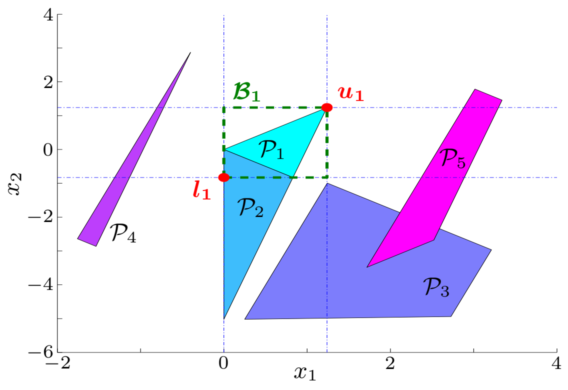

Fig. 1: Overlapping collection of polytopic sets $\{\mathcal{P}_i\}_{i=1}^5$ with bounding box $\mathcal{B}_1$ of $\mathcal{P}_1$.

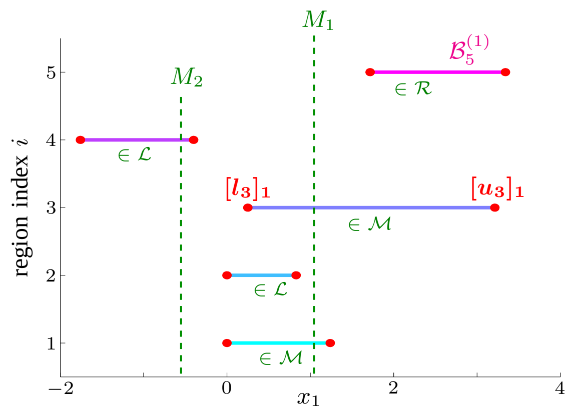

Fig. 2: Projection $\mathcal{B}_i^{(1)}$ of the bounding boxes of the polytopic set-collection $\{\mathcal{P}_i\}_{i=1}^5$ of Figure 1 onto the x_1-space sorted by the region's index. Indicators for the construction of the first node level of the first dimension of the interval tree are represented in green.

In other words, $\mathcal{B}_i$ is the 'smallest' axis-aligned n_x-dimensional hyper-rectangle that contains $\mathcal{P}_i$. An example bounding box $\mathcal{B}_1$ can be seen in Figure 1.

Remark VI.1. Note that if the regions $\mathcal{P}_i$ are polytopes, then a minimal volume bounding box can be computed using $2n_x$ linear programs of dimension n_x [2]. □

For a given query point, or measured state $x(t)$, the proposed algorithm operates in two stages. First, a list $\mathcal{I}^\mathcal{B}$ of bounding boxes containing the point $x(t)$ is computed, i.e. $x(t) \in \mathcal{B}_i$ for all $i \in \mathcal{I}^\mathcal{B}$ (Section VI-A). Second, for each index $i \in \mathcal{I}^\mathcal{B}$, the region $\mathcal{P}_i$ is tested to determine if it contains $x(t)$ (Section VI-B). In the following $x(t)$ is simply denoted by x for brevity.

The first stage of this procedure is extremely efficient and computationally 'inexpensive', since the containing bounding boxes can be reported in logarithmic time. This can be done by breaking the search down into one-dimensional range queries, which is possible due to the axis-aligned nature of the bounding boxes. The complexity of the second stage of the algorithm is a function of the overlap between the bounding boxes of adjacent regions. A significant advantage of the proposed search tree is a very simple and effective preprocessing step, which allows the method to be applied to controllers defined over a very large number of regions, i.e. several tens of thousands. As is shown in Section VIII, there are several large problems of interest to control which have a structure that makes this procedure efficient.

Remark VI.2 (Overlapping regions). Note that overlapping regions are treated naturally and without any additional heuristics by the algorithm. □

A. Bounding Box Search Tree

In this section we will detail the procedure for reporting the set of indices $\mathcal{I}^\mathcal{B}$ of all bounding boxes that contain a given point x. The algorithm relies on the fact that one can decompose the search of a query point $x \in \mathbb{R}^{n_x}$ in a set of bounding boxes in an n_x-dimensional space into n_x separate one-dimensional sequential or parallel searches, because the bounding boxes are all axis-aligned.

The basic steps for constructing the search tree are given in Algorithm VI.3.

Algorithm VI.3 (Building the search tree)

```
1. compute the bounding box $\mathcal{B}_i$ for each $\mathcal{P}_i$
2. project each bounding box $\mathcal{B}_i$ onto each dimension
   $d = 1, \dots, n_x$: define $\mathcal{B}_i^{(d)}$ as the resulting interval
3. build an $n_x$-dimensional interval tree
```

Note that Step 2 of Algorithm VI.3 for axis-aligned bounding boxes is merely a coordinate extraction of the corner points l_i and u_i.

The proposed search algorithm is an extension to the well known concept of *interval trees* [9], [8]. Standard interval trees are efficiently ordered binary search trees for determining a set of possibly overlapping one-dimensional line segments that contain a given point or line segment. Consider Figure 2, in which the intervals of the bounding boxes in the first dimension for the example in Figure 1 are shown. The intervals are spread vertically, ordered by their respective index i, to make them easier to see.

Each node of the search tree, cf. Figure 3 and 2, is associated with a median point M. For example the root node T in Figure 3 is associated with the point M_1 in Figure 2. The node splits the set of intervals into three sets: The set $\mathcal{L}$, consisting of those entirely on the left of the point M, $\mathcal{R}$ those entirely on the right and $\mathcal{M}$, those that intersect it. The set $\mathcal{M}$ is stored in the node and the left and right branches of the tree are formed by choosing points above and below M and repeating this procedure on $\mathcal{L}$ and $\mathcal{R}$, respectively. By choosing the point M to be the median

$$M := \frac{1}{2}\left(\min_{i\in\mathcal{J}}\{[l_i]_d\} + \max_{i\in\mathcal{J}}\{[u_i]_d\}\right)$$

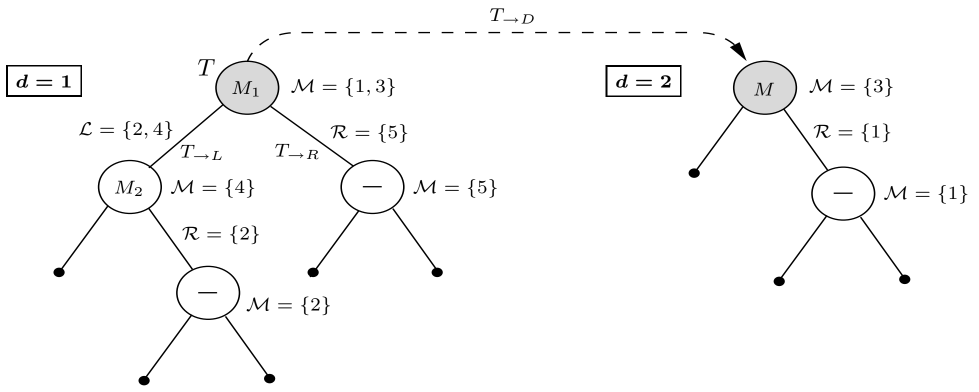

Fig. 3: Two-dimensional interval tree for the collection of polytopes $\mathcal{P}$ in Figure 1. The gray indicated node in $d=1$ is explored further in $d=2$.

of the considered intervals $\mathcal{J}$ at a given step, the number of intervals at each level of the tree drops logarithmically. This standard interval tree for the example in Figure 2 is shown in the left ($d=1$) of Figure 3.

The tree can then be used on-line to determine the set $\mathcal{I}^{\mathcal{B}}$ of intervals containing a given point $[x]_1$, which is the first dimension of the query point x, as follows. Beginning at the root node T, the point $[x]_1$ is compared to the point M_1 associated with the root node. If we assume that the point $[x]_1$ is larger than M_1, then it is contained in all intervals in the set $\mathcal{M}$ whose right endpoint $[u_i]_1$ is larger than $[x]_1$, since M_1 is less than $[x]_1$ and is also contained in the interval. Note that this search over the set $\mathcal{M}$ can be done in logarithmic time by pre-sorting the endpoints of the intervals in $\mathcal{M}$. Finally, the tree is followed down the right branch, denoted $T_{\to R}$ in Figure 3, and this procedure is repeated recursively. If the point $[x]_1$ is less than M_1, then a similar procedure is carried out on the lower bounds and the left branch is followed, which is labeled $T_{\to L}$ in Figure 3.

Now this standard method is extended to higher dimensions by building an interval tree over the sets $\mathcal{M}$ at each node using the next dimension $[x]_2$, i.e. $d=2$. In Figure 3, the tree on the left resembles the interval tree for the first measured dimension $[x]_1$. The root node T contains several elements $\mathcal{M}=\{1,3\}$, i.e. $|\mathcal{M}|>1$, and therefore an interval tree, labeled $T_{\to D}$ in Figure 3, over the second dimension ($d=2$) is constructed for this node, in which only the elements $\{1,3\}$ are considered and where the search is performed for the second dimension of the measured variable $[x]_2$ only. This tree is shown on the right of the figure. By continuing in this manner, the approach is extended to arbitrary dimensions n_x.

B. Local Search

As already mentioned, the interval search tree only provides a list of candidates $\mathcal{I}^{\mathcal{B}}$ that are possible solutions to the point-location problem. In order to identify the particular index set $\mathcal{I}\subseteq\mathcal{I}^{\mathcal{B}}$ that actually contains the measured point $x(t)$, cf. Step 2 of Algorithm III.1, a local search algorithm needs to be executed on the list of candidate regions by exhaustively testing a set membership $x\in\mathcal{P}_i$ for all $i\in\mathcal{I}^{\mathcal{B}}$.

If the cost function associated with a solution of a CFTOC Problem IV.2 is convex, one can use the approach of [6] in which the local search can be performed in $(2n_x+2)|\mathcal{I}^{\mathcal{B}}|$ arithmetic operations.

VII. Complexity

A. Preprocessing

The preprocessing phase for the proposed algorithm occurs in two steps. First, the bounding boxes for each region are computed, and then the n_x-dimensional interval tree is built. The calculation of a bounding box requires two linear programs per dimension per region. Therefore, if there are $N_{\mathcal{P}}$ regions, then the calculation of the bounding boxes requires exactly $2n_xN_{\mathcal{P}}$ linear programs of dimension n_x. The construction of the interval tree can be performed in $O(n_xN_{\mathcal{P}}\log(N_{\mathcal{P}}))$ [9] and as can be seen from the examples in Section VIII, the required computation is insignificant compared to the computation of the bounding boxes.

Note that as the preprocessing for this algorithm requires two linear programs per region, it is guaranteed to take significantly less time than the initial computation of the controller. It follows that this approach can be applied to any system for which an explicit controller can be calculated. Note also that bounding boxes are computed in some parametric solvers as the solution is computed [22], making the additional off-line computation negligible.

B. Storage

The algorithm requires the storage of the defining inequalities for each region as well as the structure of the tree. The tree has a 3-ary structure, two branches for the left and right, and one for the next dimension. In each non-leaf node of the tree is stored a median point M_i and pointers for each of the three branches, totaling four numbers. The leaf nodes then store the indices of the bounding boxes that will be checked during the local search. A completely unbalanced tree in which each leaf node contains exactly one bounding box is the most space consuming configuration. This 'worst case' tree would have $N_{\mathcal{P}} - 1$ non-leaf nodes and $N_{\mathcal{P}}$ leaves for a worst case total storage requirement of $4(N_{\mathcal{P}} - 1) + N_{\mathcal{P}}$ numbers (pointers or reals). Note that this worst case complexity is linear in the number of regions and independent of the state dimension.

C. *On-line Complexity*

The interval tree can be traversed in $O(\log(N_{\mathcal{P}}) + |\mathcal{I}^{\mathcal{B}}|)$ time, where $|\mathcal{I}^{\mathcal{B}}|$ is the number of intervals returned [9]. However, all current methods of doing the secondary search over the list of $|\mathcal{I}^{\mathcal{B}}|$ potential regions returned must be done in linear time. The worst-case complexity is therefore determined by the maximum number of regions that can be returned by the interval tree search, or equally the maximum number of bounding boxes that contain a single point. In the worst case, a point would exist that is contained in every bounding box and therefore the local search for this case would in fact be a complete global search and the resulting worst-case efficiency would be $|\mathcal{I}^{\mathcal{B}}| = N_{\mathcal{P}}$. It is demonstrated by example in the following section that there exist control problems for which the proposed method offers a significant improvement over current approaches.

VIII. Numerical Examples

A. *Constrained LTI System*

The proposed algorithm of Section VI was applied to the following linear system with three states and two inputs

$$x(t+1) = \begin{bmatrix} 7/10 & -1/10 & 0 \\ 1/5 & -1/2 & 1/10 \\ 0 & 1/10 & 1/10 \end{bmatrix} x(t) + \begin{bmatrix} 1/10 & 0 \\ 1/10 & 1 \\ 1/10 & 0 \end{bmatrix} u(t).$$

The system is subject to input constraints, $-5\mathbb{1}_2 \le u(t) \le 5\mathbb{1}_2$, and state constraints, $-20\mathbb{1}_3 \le x(t) \le 20\mathbb{1}_3$, where $\mathbb{1}_m := [1\ 1\ \dots\ 1]' \in \mathbb{R}^m$. The CFTOC Problem IV.2 is solved with $p = 1$, $T = 8$, $Q = I_3$, $R = \frac{1}{10} I_2$, and $P = 0_{3\times 3}$. The receding horizon state feedback control law (6) consists of $2\,568$ polyhedral regions in $\mathbb{R}^3$.

As can be seen from Table I, the algorithm presented in Section VI is required to solve $2 \cdot 3 \cdot 2\,568 = 15\,408$ linear programs in the preprocessing phase and needs to store $15\,408$ real numbers to represent the bounding boxes, as well as $4\,424$ pointers in order to represent the tree. Since the cost function for this example is piecewise affine and convex, it is possible to use the method in [6] for the local search, cf. Section VI-B, which requires an additional storage of $10\,272$ real numbers.

In comparison, the binary search tree of [30] for this case consists of 815 unique hyperplanes. For each such hyperplane $2N_{\mathcal{P}}$ LPs must be solved in the preprocessing phase to compute the index set which corresponds to $4\,185\,840$ linear programs. An actual additional $1\,184\,782$ linear programs are needed to construct the tree, which does not correspond to the worst case scenario.

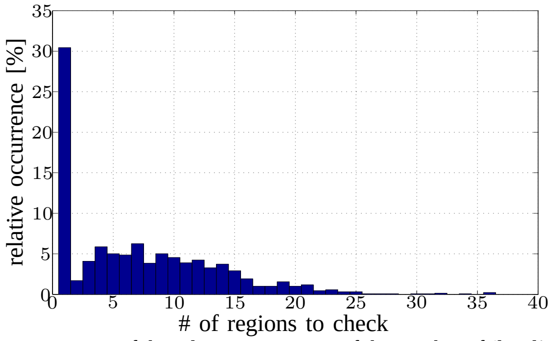

Fig. 4: Histogram of the relative occurrence of the number of 'local' regions for the example of Section VIII-A.

In order for the proposed method to identify the control law on-line, one has to perform 707 floating point operations to traverse the interval tree in the worst case. Since the tree only provides a necessary condition for the point-location problem, one has to perform a local search on the regions identified by the tree as possible candidates (Section VI-B). To provide a worst case bound, an exhaustive check for all possible intersections of the intervals stored in the presented tree was performed. In the worst case 36 regions need to be checked using the method of [6], which corresponds to 216 flops. However, as can be seen from Figure 4, a unique control law is automatically reported by the here proposed search tree in $31\,\%$ of all cases without the requirement of doing a secondary local search. In addition, approximately $90\,\%$ of all search queries do not require an exhaustive check of more than 15 regions.

	sequential search	Alg. in [30]	***Alg. VI.3*** ([6] locally)
number of LPs (off-line)	—	5 370 622	15 408
runtime (off-line)	—	10 384 secs	10 secs
on-line arithm. operations (worst case)	106 295	110	923

Tab. I: Computational complexity for the example of Section VIII-A.

B. *Constrained PWA System*

Consider the following piecewise affine system from [25]

$$x(t+1) = \begin{cases} A_1 x(t) + Bu(t), & \text{if } [0,\,1,\,0]x(t) \le 1, \\ A_2 x(t) + Bu(t) + a, & \text{otherwise,} \end{cases}$$

where

$$A_1 = \begin{bmatrix} 1 & 1/2 & 3/10 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, A_2 = \begin{bmatrix} 1 & 1/5 & 3/10 \\ 0 & 1/2 & 1 \\ 0 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, a = \begin{bmatrix} 3/10 \\ 1/2 \\ 0 \end{bmatrix}.$$

The system is subject to input constraints, $-1 \le u(t) \le 1$, and state constraints, $[-10, -5, -10]' \le x(t) \le [10, 5, 10]'$. With $p = 1$, $T = 7$, $Q = I_{3\times 3}$, $R = 1/10$, and $P = 0_{3\times 3}$. The solution to the CFTOC Problem IV.2 resulted in a receding horizon state feedback control law (6) defined over $2\,222$ polyhedral regions in $\mathbb{R}^3$.

The off-line construction of the interval search tree for this example required $13\,332$ LPs to be solved, compared to

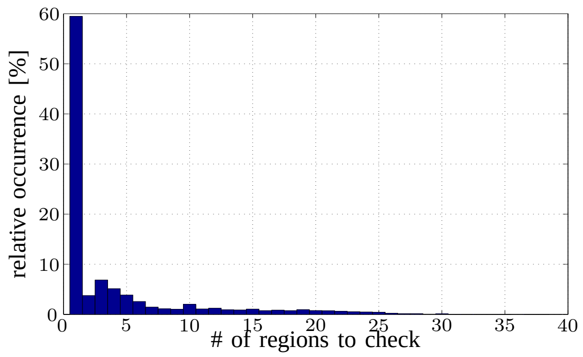

Fig. 5: Histogram of the relative occurrence of the number of 'local' regions for the example of Section VIII-B.

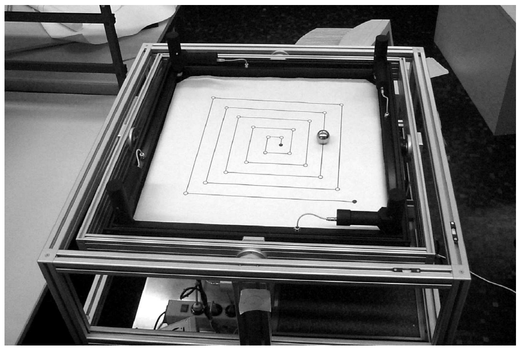

Fig. 6: Ball & Plate laboratory setup. The ball follows a pre-specified trajectory.

$7.9 \cdot 10^6$ linear programs which are needed to construct the binary search tree of [30] (this does not correspond to the worst case scenario). Since the cost function of a given CFTOC solution is not necessarily convex, one cannot use the method of [6] to perform the local search, and therefore one must perform a sequential search as outlined in Section VI-B on possible candidates. Using the same methodology as in the previous example, it was found that at most 39 regions have to be searched exhaustively. This, however, takes at most 1 720 flops. The worst case number of floating point operations needed to traverse the interval tree is 882. Moreover, almost $60\,\%$ of all search queries result in a unique control law during the first phase of the algorithm (Section VI-A), cf. Figure 5. Therefore no additional sequential searches are necessary in these cases. Results on the computational complexity are summarized in Table II.

	sequential search	Alg. in [30]	***Alg. VI.3***
number of LPs (off-line)	—	7 913 462	13 332
runtime (off-line)	—	12 810 secs	4.8 secs
on-line arithm. operations (worst case)	97 984	352	2 602

Tab. II: Computational complexity for the example of Section VIII-B.

C. Ball & Plate System

The mechanical 'Ball & Plate' system was introduced in [5], [17]. The experiment consists of a ball rolling over a gimbal-suspended plate actuated by two independent motors, cf. Figure 6. The control objective is to make the ball follow a prescribed trajectory, while minimizing the control effort. The dynamical model for the y-axis of such a device is given by

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 700 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -34.69 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 3.1119 \end{bmatrix} u(t), \qquad (7)$$

where $x := [y,\ \dot{y},\ \alpha,\ \dot{\alpha}]'$ is the state. $-30 \leq y \leq 30$ and $-15 \leq \dot{y} \leq 15$ are the position and velocity of the ball with respect to the y-coordinate, $-0.26 \leq \alpha \leq 0.26$ and $-1 \leq \dot{\alpha} \leq 1$ denote the angular position and angular velocity of the plate, respectively. The input voltage to the motor is assumed to be constrained by $-10 \leq u \leq 10$. In order to take the tracking requirements into account, the state vector is extended with an additional element, which contains the reference signal, hence the augmented state vector is in $\mathbb{R}^5$. The model (7) was then discretized with sampling time $T_s = 0.03$ and a closed-form PWA feedback control law (6) was derived for the CFTOC Problem IV.3, where the following parameters $T = 10$, $Q = \text{diag}([6,\ 0.1,\ 500,\ 100,\ 6])$, $R = 1$, and $P = Q$ were considered. The controller obtained using the Multi-Parametric Toolbox [22] for MATLAB® is defined over 22 286 regions in $\mathbb{R}^5$.

The computational results for the respective search trees are summarized in Table III. Due to the high number of regions, the binary search tree of [30] was not applicable to this example (denoted by $\star$ in Table III), since it would require the solution of $3.5 \cdot 10^9$ LPs already in the preprocessing stage to determine the index set before building the binary search tree. In contrast, in the preprocessing stage, the here proposed algorithm has to solve 228 860 LPs to obtain the bounding boxes for all regions. The overall time needed to construct the complete search tree, including the computation of the bounding boxes, was just 80 seconds. 9 324 pointers are needed to represent the tree structure, and 222 860 floating point numbers are needed to describe the bounding boxes.

To estimate the average and the worst case number of arithmetic operations needed to identify the control law on-line, we have investigated 10 000 random initial conditions over the whole feasible state space. It can be seen from the histogram distribution depicted in Figure 7 that the search tree algorithm identifies at most 500 regions as possible candidates in $86\,\%$ of all tested initial conditions. The subsequent sequential search over 500 regions corresponds

	sequential search	Alg. in [30]	***Alg. VI.3***
number of LPs (off-line)	—	$3.5 \cdot 10^9$	222 860
runtime (off-line)	—	$\star$	80 secs
on-line arithm. operations (worst case)	1 400 178	$\star$	208 849

Tab. III: Computational complexity for the example in Section VIII-C. $\star$ denotes that the algorithm in [30] is not computable for this example.

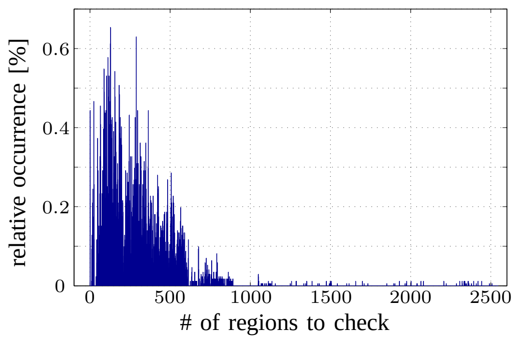

Fig. 7: Histogram of the relative occurrence of the number of 'local' regions for the example of Section VIII-C.

to $30\,000$ floating point operations. In 99% of all tested cases the algorithm identifies at most 1000 regions for subsequent local search, which corresponds to at most $60\,000$ flops. In the worst case, the search tree will identify as many as $2\,544$ regions as possible candidates for a sequential search. Notice that this number represents, in the worst case, only $11\,\%$ of the total number of regions. This amounts to a maximum of $152\,640$ flops, whereas traversal of the tree contributes another $56\,209$ flops. The sequential search through all regions, on the other hand, would require $1.4 \cdot 10^6$ operations and is currently the only other method that can be applied to such a large system. The total number of flops which are needed to be performed on-line is thus (in the worst case) reduced by at least one order of magnitude. To give a sensible feeling for this number of floating point operations, note that a $3\,$GHz Pentium 4 computer can execute approximately $800 \cdot 10^6$ flops/sec. Given this performance the controlled system can be run at a sampling rate of $4\,$kHz in the case of the presented search tree, whereas the sequential search has a limit of $500\,$Hz.

IX. Software Implementation

The presented algorithm is implemented in the Multi-Parametric Toolbox (MPT) [22] for MATLAB®. The toolbox can be downloaded free of charge at: `http://control.ee.ethz.ch/~mpt/`

References

[1] M. Baotić, F. J. Christophersen, and M. Morari, "A new Algorithm for Constrained Finite Time Optimal Control of Hybrid Systems with a Linear Performance Index," in *Proc. of the European Control Conference*, Cambridge, UK, Sept. 2003, available from http://control.ee.ethz.ch/index.cgi?page=publications.

[2] A. Bemporad, C. Filippi, and F. D. Torrisi, "Inner and outer approximation of polytopes using boxes," *Computational Geometry: Theory and Applications*, vol. 27, no. 2, pp. 151–178, 2004.

[3] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, no. 3, pp. 407–427, Mar. 1999.

[4] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, Jan. 2002.

[5] F. Borrelli, *Constrained Optimal Control of Linear and Hybrid Systems*, ser. Lecture Notes in Control and Information Sciences. Springer-Verlag, 2003, vol. 290.

[6] F. Borrelli, M. Baotić, A. Bemporad, and M. Morari, "Efficient On-Line Computation of Constrained Optimal Control," in *Proc. of the Conf. on Decision & Control*, Orlando, Florida, USA, Dec. 2001.

[7] ——, "An Efficient Algorithm for Computing the State Feedback Solution to Optimal Control of Discrete Time Hybrid Systems," in *Proc. on the American Control Conference*, Denver, Colorado, USA, June 2003, pp. 4717–4722.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press and McGraw-Hill, 2001.

[9] M. de Berg, O. Schwarzkopf, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 2nd ed. Springer Verlag, 2000.

[10] B. De Schutter and T. Van den Boom, "On model predictive control for max-min-plus-scaling discrete event systems," *Automatica*, vol. 37, no. 7, pp. 1049–1056, 2001.

[11] G. Ferrari-Trecate, M. Muselli, D. Liberati, and M. Morari, "A clustering technique for the identification of piecewise affine systems," *Automatica*, vol. 39, no. 2, pp. 205–217, Feb. 2003.

[12] T. Geyer, F. D. Torrisi, and M. Morari, "Efficient Mode Enumeration of Compositional Hybrid Models," in *Proc. of the Intern. Workshop on Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, vol. 2623. Springer-Verlag, 2003, pp. 216–232.

[13] ——, "Optimal Complexity Reduction of Piecewise Affine Models Based on Hyperplane Arrangements," in *Proc. on the American Control Conference*, Boston, Massachusetts, USA, June 2004, pp. 1190–1195.

[14] B. Grünbaum, *Convex Polytopes*, 2nd ed. Springer-Verlag, 2000.

[15] W. P. M. H. Heemels, "Linear Complementarity Systems: A Study in Hybrid Dynamics," Ph.D. dissertation, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, Nov. 1999.

[16] W. P. M. H. Heemels, B. De Schutter, and A. Bemporad, "Equivalence of hybrid dynamical models," *Automatica*, vol. 37, no. 7, pp. 1085–1091, 2001.

[17] O. Hermann, "Regelung eines ball and plate systems," Diploma thesis, ETH Zurich, Zurich, Switzerland, 1996.

[18] R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge University Press, 1985.

[19] M. Johansson, *Piecewise Linear Control Systems: A Computational Approach*, ser. Lecture Notes in Control and Information Sciences. Springer-Verlag, 2003, vol. 284.

[20] C. N. Jones, P. Grieder, and S. V. Raković, "A Logarithmic Solution to the Point Location Problem for Closed-Form Linear MPC," in *IFAC World Congress*, Prague, Czech Republic, July 2005.

[21] E. C. Kerrigan and D. Q. Mayne, "Optimal control of constrained, piecewise affine systems with bounded disturbances," in *Proc. of the Conf. on Decision & Control*, Las Vegas, Nevada, USA, Dec. 2002, pp. 1552–1557.

[22] M. Kvasnica, P. Grieder, and M. Baotić, "Multi-Parametric Toolbox (MPT)," 2004, available from http://control.ee.ethz.ch/~mpt/.

[23] J. M. Maciejowski, *Predictive Control with Constraints*. Prentice Hall, 2002.

[24] D. Q. Mayne, "Constrained Optimal Control," European Control Conference, Plenary Lecture, Sept. 2001.

[25] D. Q. Mayne and S. V. Raković, "Model predictive control of constrained piecewise affine discrete-time systems," *Int. Journal on Robust and Nonlinear Control*, vol. 13, no. 3-4, pp. 261–279, 2003.

[26] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, June 2000.

[27] J. Roll, A. Bemporad, and L. Ljung, "Identification of piecewise affine systems via mixed-integer programming," *Automatica*, vol. 40, pp. 37–50, 2004.

[28] J. Snoeyink, "Point Location," in *Handbook of Discrete and Computational Geometry*, J. E. Goodman and J. O'Rouke, Eds. Boca Raton, New York: CRC Press, 1997, ch. 30, pp. 558–574.

[29] E. D. Sontag, "Nonlinear regulation: The piecewise linear approach," *IEEE Trans. on Automatic Control*, vol. 26, no. 2, pp. 346–358, Apr. 1981.

[30] P. Tøndel, T. A. Johansen, and A. Bemporad, "Evaluation of Piecewise Affine Control via Binary Search Tree," *Automatica*, vol. 39, no. 5, pp. 945–950, May 2003.

[31] A. van der Schaft and H. Schumacher, *An Introduction to Hybrid Dynamical Systems*, ser. Lecture Notes in Control and Information Sciences. Springer-Verlag, 2000, vol. 251.