Efficient point location via subdivision walking with application to explicit MPC

Yang Wang, Colin Jones and Jan Maciejowski

Abstract— **An explicit (or closed-form) solution to Model Predictive Control (MPC) results in a polyhedral subdivision of the state-space when the system and constraints are linear, and the cost is linear or quadratic. Within each region the optimal control law is an affine function of the current state, so the online evaluation is reduced to determining the region containing the current state measurement, known as a point-location or set membership problem. In this paper we present the subdivision walking method, which is based on the idea of travelling from a seed point in a known seeded region, in the direction of the state measurement, by walking from one region to the next until the region of interest is found. The algorithm requires minimal pre-computation, and achieves significant computational savings for many control problems.**

I. INTRODUCTION

Point-Location is a well studied problem in computational geometry, with applications to many branches of science and engineering. Given a set of regions and a point in n-dimensional space, the aim is to determine as efficiently as possible the region that contains this point. Beside the many manifestations of point location in a variety of different fields, we will concentrate in this paper on its application to explicit (or closed-form) Model Predictive Control (MPC). This will necessarily lead to a search with $n \gg 3$, whereas the most effective methods currently only work for $n = 2$ and $n = 3$.

In MPC an optimal sequence of inputs is chosen to minimise a given objective over a finite prediction horizon. Online computation amounts to solving an optimisation at every sampling instant, depending on the form of the cost. In recent years it has become well established [1] that the optimal input is a piecewise affine function defined over a polyhedral partition of all feasible states. This can be entirely pre-computed using multi-parametric optimisation [1] [2] [3], resulting in an explicit or closed-form solution to the MPC problem. The online calculation is therefore reduced to determining the region containing the current state - the so called *point location* or *set membership* problem.

Explicit MPC is not intended to replace traditional methods entirely but instead to expand its sphere of application. In particular, online evaluation times can be significantly reduced, leading to controller update intervals in the order of micro-seconds. The complexity of the closed-form solution is highly dependent on the parameters in the MPC problem formulation. Although the number of regions in a solution cannot be determined a-priori, in the worst case it is known to grow exponentially with horizon length, state/input dimension and the number of constraints [1]. The potential for highly complex problems with many thousands of subdivisions implies that an efficient way of solving the point location problem is required.

The simplest way to do this is by brute force. Each region is examined until the region containing the current state is found. The worst-case computational complexity of this approach is linear in the number of regions. In [4], the authors improve on both the search time and memory requirements of this basic method by exploiting various properties of the MPC value function. A more recent development [5] seeks to construct a binary search tree by dividing up the polyhedral partition using auxiliary hyperplanes. These will subdivide existing regions so that search time is logarithmic in the number of *subdivided* regions, which may be significantly more numerous. The offline pre-processing time required to implement this method is also prohibitive for large problems.

For MPC controllers involving 1 or ∞-norm costs, the objective of the optimisation problem is linear. In this case, [6] demonstrates that the polyhedral partition corresponds to an additively weighted Voronoi diagram. Using the approximate nearest neighbours algorithm [7], this geometric structure can be searched in time logarithmic in the number of regions, resulting in significant computational gains.

The structure of the closed-form solution arising from *non-linear* (and in particular, quadratic) penalty functions is currently not well understood, and does not contain any obvious structure we can exploit. Instead we will present a method that solves the point location problem, without regard for the underlying MPC formulation. The method requires almost no pre-computation, but is *heuristic* in the sense that the absolute worst case complexity is still linear in the total number of regions. We shall demonstrate however, that significant computational gains can often be achieved, and that the worst case can be determined offline given a specific problem. In [8] a similar method is used to speed up online solution of QP problems arising in MPC. The rest of the paper is structured as follows.

In section 2 the point location problem is formally stated, and the application to MPC discussed. Section 3 introduces subdivision walking, and Section 4 demonstrates how the worst-case online complexity can be calculated. Section 5

Y. Wang is supported by a Rambus Corporation Stanford Graduate Fellowship in the Department of Electrical Engineering, Stanford University, 350 Serra Mall, Stanford, CA 94305-9505, USA yw224@stanford.edu

C.N. Jones is with the Automatic Control Laboratory, Swiss Federal Institute of Technology, Physikstrasse 3, ETL K11, Zurich, Switzerland cjones@control.ee.ethz.ch

J.M. Maciejowski is with the Department of Engineering, Cambridge University, Trumpington Street, Cambridge, CB2 1PZ, United Kingdom jmm@eng.cam.ac.uk

provides some numerical examples and evaluates the performance of this algorithm on typical control problems.

PRELIMINARY DEFINITIONS

Definition [6] A *polyhedron* is the intersection of a finite number of halfspaces: $\mathcal{P} \triangleq \{x \in \mathbb{R}^n | Ax \leq b\}$. A *polytope* is a bounded polyhedron.

Definition [6] $\mathcal{F}$ is a *face* of the polyhedron $\mathcal{P} \subset \mathbb{R}^n$ if there exists a hyperplane $\{x \in \mathbb{R}^n | a^T x = b\}$, where $a \in \mathbb{R}^n$, $b \in \mathbb{R}$, such that $\mathcal{F} = \mathcal{P} \cap \{x \in \mathbb{R}^n | a^T x = b\}$ and $a^T x \leq b$ for all $x \in \mathcal{P}$.

Definition [6] A finite family $\mathcal{C}$ of polytopes in $\mathbb{R}^n$ is a *complex* if every face of a member of $\mathcal{C}$ is itself a member of $\mathcal{C}$ *and* the intersection of any two members of $\mathcal{C}$ is a face of each of them.

II. PROBLEM DEFINITION

A. Point Location Problem

Definition *Point Location Problem* [6]: Given a vector x and a set of non-intersecting polytopes $\{\mathcal{X}_1, \ldots, \mathcal{X}_R\}$, determine any integer $r(x) \in \{1, \ldots, R\}$ such that polytope $\mathcal{X}_{r(x)}$ contains x. If $x \notin \bigcup_{i=1}^{R} \mathcal{X}_i$, then $r(x) = 0$.

Throughout this work we will assume that the polytopes are stored in halfspace representation. This is because the typical geometric structure arising from optimal control has many more vertices than faces, so vertex representation should be avoided.

B. MPC Formulation and Solution

Consider the following discrete time linear time invariant state space representation, with state vector $x[k] \in \mathbb{R}^{n_x}$, manipulated variables $u[k] \in \mathbb{R}^m$, $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times m}$, and where the pair (A, B) is controllable.

$$x[k+1] = Ax[k] + Bu[k] \tag{1}$$

We consider an MPC problem where the goal is to minimise a quadratic cost over the future states and inputs (2), where $R = R^T \succ 0$ ($\succ$ denotes positive-definite), $P = P^T \succ 0$, $Q = Q^T \succeq 0$ ($\succeq$ denotes positive semi-definite), $x[k]$ is the current measured state, $x_0 \ldots x_N$ are the predicted states at time k, $u_0 \ldots u_{N-1}$ are the predicted inputs at time k and Ω is a polyhedral terminal set that contains the origin. For simplicity we assume that the control horizon H_u is equal to the prediction horizon H_p, $H_p = H_u = N$.

$$\begin{aligned} \text{minimise} \quad & J[k] = \|x_N\|_P^2 + \sum_{i=0}^{N-1} \left(\|x_i\|_Q^2 + \|u_i\|_R^2 \right) \\ \text{subject to} \quad & x_0 = x[k] \qquad x_N \in \Omega \\ & x_{i+1} = Ax_i + Bu_i \quad i \in \{0, \ldots, N-1\} \\ & C_i x_i + D_i u_i \leq b \end{aligned} \tag{2}$$

The optimisation variable is the input sequence $U = \left[u_0^T, \ldots, u_{N-1}^T\right]$. In MPC we solve the optimisation (2) to obtain $U^\star$, and apply the first element $u_0^\star$ to the plant. The process is repeated at the next sampling instant.

Definition The set of states $\mathcal{X}_f$, is the set of *all feasible states* for the MPC problem if: $x \in \mathcal{X}_f$ guarantees that a sequence of inputs can be found such that the constraints are satisfied at all points in the prediction horizon, *and* $x \notin \mathcal{X}_f$ guarantees that such a sequence does not exist.

Definition The set of polytopes $\{\mathcal{X}_1, \ldots, \mathcal{X}_R\}$ is a *partition* or *subdivision* of $\mathcal{X}_f$ if the polytopes are non-intersecting, $\mathcal{X}_i \cap \mathcal{X}_j$ is not full dimensional, $i \neq j$ *and* $\mathcal{X}_1 \cup \mathcal{X}_2 \cup \ldots \cup \mathcal{X}_R = \mathcal{X}_f$.

The optimisation problem (2) is easily rearranged into a standard QP in variable U (3), for suitable matrices $Y \in \mathbb{R}^{n_x \times n_x}$, $H \in \mathbb{R}^{Nm \times Nm}$, $F \in \mathbb{R}^{n_x \times Nm}$, $W \in \mathbb{R}^{q \times 1}$, $E \in \mathbb{R}^{q \times n_x}$, $G \in \mathbb{R}^{q \times Nm}$, q constraints, with $H = H^T \succ 0$.

$$\begin{aligned} \text{minimise} \quad & \frac{1}{2} x[k]^T Y x[k] + \frac{1}{2} U^T H U + x[k]^T F U \\ \text{subject to} \quad & GU \leq W + Ex[k] \end{aligned} \tag{3}$$

For conventional MPC, the controller estimates the state $x[k]$, and solves problem (3) for the optimal sequence $U^\star$. In explicit solutions we are interested in computing the above QP as an *explicit* function of the current state $U^\star = U^\star(x[k])$. This is known as a *multi-parametric* optimisation, with current state $x[k]$ as the parameter.

The nature of the explicit solution is characterised by Theorem II.1. For a rigorous proof, and treatment of degenerate cases, refer to [1].

Theorem II.1 *Consider the multi-parametric quadratic program (mpQP) (3) and let $H \succ 0$. Then the set of all feasible parameters $\mathcal{X}_f$ is convex, the optimiser $U^\star(x) : \mathcal{X}_f \to \mathbb{R}^{Nm}$ is continuous and piecewise affine over a polyhedral partition of $\mathcal{X}_f$, and the optimal solution $J^\star(x) : \mathcal{X}_f \to \mathbb{R}$ is continuous, convex and piecewise quadratic.*

Corollary II.2 *The MPC control law $u[k] = f(x)$, $f : \mathbb{R}^{n_x} \to \mathbb{R}^m$, defined by the optimisation problem (3) is continuous and piecewise affine.*

The closed-form solution results in the subdivision of the set of feasible states $\mathcal{X}_f$ into a partition $\{\mathcal{X}_1, \ldots, \mathcal{X}_R\}$, and within each region we apply the associated affine control law. Several algorithms exist for subdividing the state space and computing the explicit solution [1], [9], [3].

III. SUBDIVISION WALKING

Notice that what results from a multi-parametric quadratic program is a *partition*. This is very different from a *solution complex*, which can only arise from non-degenerate multi-parametric linear programs (mpLPs), or lex-perturbed degenerate mpLPs [10]. In a solution complex each face corresponds to only one neighbouring region, but for partitions multiple neighbours may exist [11]. The method we will present is capable of dealing with partitions, but there are extra complications. We therefore explain the operation of the method first for a solution complex, so that details of our algorithms are clear. The complications for handling partitions are briefly discussed later.

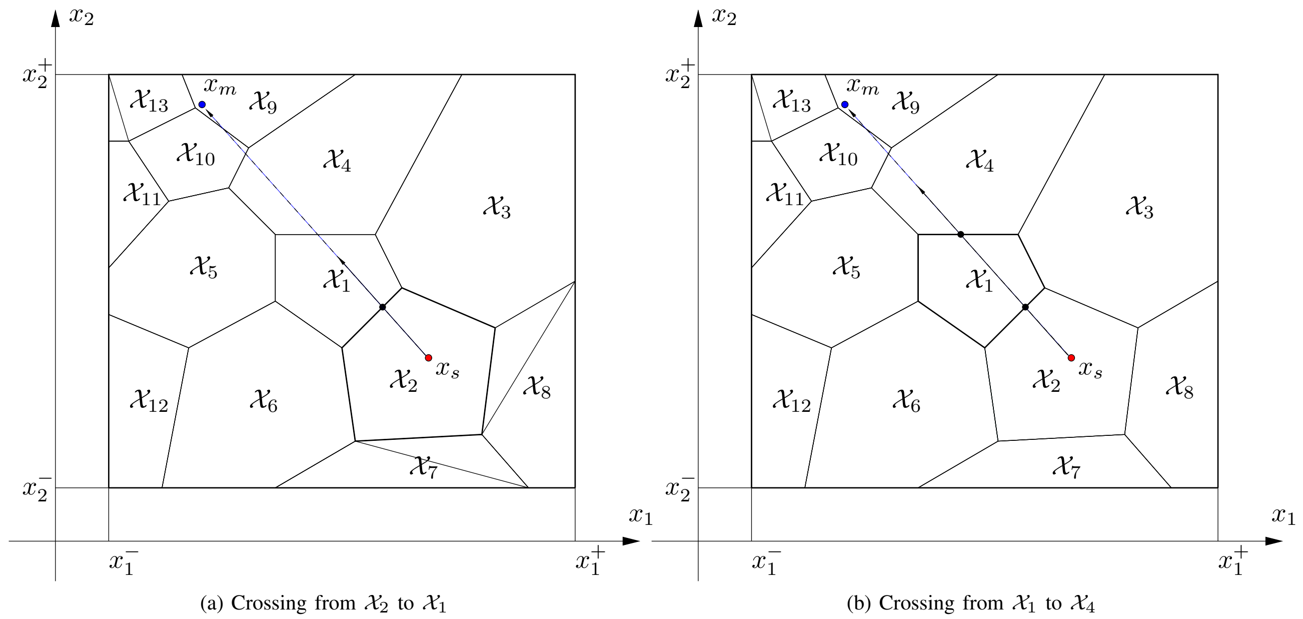

Fig. 1: Example of Subdivision Walking

The subdivision walking method is based on the concept of travelling through the state space in the direction of some point x_m contained in region $\mathcal{X}_r$, where $r(x)$ is the solution to the point location problem. The technique is easily demonstrated with an example solution complex, Fig. 1a. We start from a pre-defined seeded point x_s, which is associated with a known region $\mathcal{X}_s = \mathcal{X}_2$. To begin with a line is constructed between the seeded point and the point x_m, as shown in Fig. 1a. We start in region $\mathcal{X}_2$ containing the seed, and determine the facet that intersects with this line. In this way, it can be deduced that our line crosses into region $\mathcal{X}_1$. Once inside $\mathcal{X}_1$, the procedure is repeated and so we move into $\mathcal{X}_4$ (Fig. 1b). We continue in this fashion for successive regions until the region containing the point of interest is found. It is evident that *adjacency information*, which defines the neighbours of each region, is required to implement the algorithm.

To formalise this, we define the following

Definition Two polyhedra $\mathcal{X}_i$, and $\mathcal{X}_j$ are neighbours if they share a common facet.

Let $\mathcal{N}_i$ be the set of indices of the neighbours of $\mathcal{X}_i$, and let one element of $\mathcal{N}_i$, $\mathcal{N}_i^j$, be the index of the neighbouring region corresponding to the jth bounding hyperplane. Since we store all the polytopes in halfspace representation, $\mathcal{X}_i \triangleq \{x|H_i x \leq K_i\}$, so that the j^{th} bounding hyperplane has the representation $H_i^j x \leq K_i^j$, where H_i^j, K_i^j denote the j^{th} rows of H_i and K_i.

Given a measured state x_m and a seeded point x_s in region $\mathcal{X}_s$, define the ray $\rho = \{\lambda\nu + x_s|\lambda > 0, \nu = x_m - x_s\}$. Evaluating the crossing point of this ray with bounding hyperplane $H_s^j x = K_s^j$, we have

$$H_s^j(\lambda_j \nu + x_s) = K_s^j \tag{4}$$

$$\Rightarrow \lambda_j = \frac{K_s^j - H_s^j x_s}{H_s^j \nu},\ H_s^j \nu > 0 \tag{5}$$

We proceed to calculate the crossing λ_j for each $j \in \{1, \ldots, N_c^s\}$, where N_c^s is the total number of bounding hyperplanes of region $\mathcal{X}_s$. Clearly, since the controller partition is convex, the facet corresponding to the smallest *positive* λ_j, $\mathcal{F}_m$ is the one that intersects with the ray ρ. The region with the index $\mathcal{N}_i^m$ is therefore the region entered. Algorithm 1 summarises the procedure for point location search. If the minimum λ_j is not unique, it implies that the ray is crossing the intersection of two or more facets. In this situation, we arbitrarily select a region corresponding to either of the facets, we then generate a point in the interior of that region and continue as before.

Theorem III.1 *Algorithm 1 returns the index r of the region $\mathcal{X}_r$ containing the point x_m after at most R iterations, where R is the total number of regions.*

Proof Since the direction of search $\nu = x_m - x_s$ is predefined, and at every iteration $\lambda_j > 0$, every region is visited only once. There are only R regions in the subdivision, so m will be returned after at most R iterations. □

It is immediately obvious, that in order to employ subdivision walking, the adjacency information $\mathcal{N}_i$ must be calculated for each polytope in the partition. Adjacency computation is incorporated directly into most parametric solvers, and represents an insignificant computational overhead. Subdivision walking thus requires very little pre-computed data, and this is one of its major advantages.

Algorithm 1 Region Traversal

```
1:  procedure REGIONTRAVERSAL($x$)
2:      $r \leftarrow s$                                   ▷ Current region index
3:      $x_r \leftarrow x_s$                               ▷ Point in current region
4:      $\nu \leftarrow x - x_s$                           ▷ Direction of line
5:      while $1$ do
6:          if $H_r x \leq K_r$ then return $r$            ▷ Region found
7:          else
8:              for $j = 1$ to $N_c^r$ do                  ▷ For each facet
9:                  $\lambda_j = K_r^j - H_r^j x_r / H_r^j \nu$, $H_r^j \nu > 0$
10:             end for
11:             $m \leftarrow \arg\min_j \{\lambda_j > 0, j = 1, \ldots, N_c^r\}$
12:             $r \leftarrow \mathcal{N}_r^m$             ▷ New region
13:             $x_r = (\lambda_m + \epsilon)\nu + x_r$     ▷ New point, just
    inside the region. $\epsilon$ small, $\epsilon \geq 0$
14:         end if
15:     end while
16: end procedure
```

The choice of the seed point x_s clearly has an important effect on performance, and many different variations are possible. We could for example, have a seed point in each region. The online algorithm would first search for the closest seed as a nearest neighbours problem, which can be done in logarithmic time [7]. Alternatively, we can use the previous state measurement as the seed when solving the point location problem for the next state measurement. The rationale is that successive states tend to stay within regions that are close to each other, which would certainly be true after the state has been regulated within some terminal constraint set.

For the case of a partition, each facet can correspond to multiple neighbours. As a result, once the facet $\mathcal{F}_m$ is determined, we must further compute the neighbour that we are crossing into. Since experience shows that complexes occur in the majority of cases for quadratic programs, this would not represent an extra computational cost most of the time.

IV. WORST CASE COMPLEXITY

The difficulty with subdivision walking is putting an upper bound on the computation time. The worst case scenario is still linear in the total number of regions, but we shall present an algorithm that allows a bound to be calculated for a specific problem. In the following derivation, we will assume that only one seed point x_s is used.

Definition A set of polytopes $(\mathcal{X}_{i_1}, \ldots, \mathcal{X}_{i_m})$ is a path if $\mathcal{X}_{i_j}$ and $\mathcal{X}_{i_{j-1}}$ are adjacent for each $1 < j \leq m$. The *length* of such a path is m, the *distance* between polytope $\mathcal{X}_i$ and $\mathcal{X}_j$ is the length of the shortest path connecting them. The *diameter* of a subdivision is the maximum path length occurring between any pair of polytopes in the subdivision.

Definition A path $(\mathcal{X}_{i_1}, ..., \mathcal{X}_{i_m})$ is called a *linear path* between x_s and $\mathcal{X}_{i_m}$, $\mathcal{I}_{x_s \to \mathcal{X}_{i_m}}$, if there exists a ray $\rho = \{\lambda\nu + x_s | \lambda > 0, x_s \in \mathcal{X}_{i_1}\}$, such that $\rho \cap \mathcal{X}_{i_j} \cap \mathcal{X}_{i_{j+1}} \neq \emptyset, \forall j = 1, \ldots, m-1$.

Definition Given a linear path $\mathcal{I}_{x_s \to \mathcal{X}_{i_m}}$, define the associated *target set* $\mathcal{T}\left(\mathcal{I}_{x_s \to \mathcal{X}_{i_m}}\right) \subseteq \mathcal{X}_{i_m}$ as the set of *all* points $x_m \in \mathcal{X}_{i_m}$, for which there exists a ray $\rho = \{\lambda\nu + x_s | \lambda > 0, \nu = x_m - x_s, x_s \in \mathcal{X}_{i_1}\}$, such that $\rho \cap \mathcal{X}_{i_j} \cap \mathcal{X}_{i_{j+1}} \neq \emptyset, \forall j = 1, \ldots, m-1$.

That is, a linear path is a sequence of regions that a ray traverses to get from $x_s \in \mathcal{X}_{i_1}$ to some point in region $\mathcal{X}_{i_m}$. Notice that linear paths are not unique, there may be many different combinations of regions a ray can cross to get to $\mathcal{X}_{i_m}$. This concept is demonstrated for an arbitrary controller partition in Figs. 2a, 2b, 2c and 2d. To walk from x_s to $\mathcal{X}_{10}$ there are four different sequences of regions $(\mathcal{X}_2, \mathcal{X}_1, \mathcal{X}_4, \mathcal{X}_{10})$, $(\mathcal{X}_2, \mathcal{X}_1, \mathcal{X}_5, \mathcal{X}_4, \mathcal{X}_{10})$, $(\mathcal{X}_2, \mathcal{X}_1, \mathcal{X}_5, \mathcal{X}_{10})$, $(\mathcal{X}_2, \mathcal{X}_6, \mathcal{X}_1, \mathcal{X}_5, \mathcal{X}_{10})$. The associated target sets for each linear path are also highlighted.

In order to calculate the worst case computation time to search for a vector x_m, we must enumerate *all possible* linear paths $\mathcal{I}_{x_s \to \mathcal{X}_i}$, $\forall i \in \{1, \ldots, N_R\}$. The linear path with the largest number of elements therefore corresponds to the upper bound we wish to compute. This may seem like a tremendous task, but in fact it can be simplified greatly through the generation of a *crossing tree*. Algorithm 2 summarises this procedure, and we can prove the following results.

Theorem IV.1 *Every route from the root node Π_s of a crossing tree down to any node Π_i traces out a linear path between x_s and $\mathcal{X}_i$.*

Proof Each node Π_i is added based on whether the route from the root node Π_s down to Π_i traces out a sequence of regions that corresponds to a linear path between x_s and $\mathcal{X}_i$. If this is not the case: equivalently if the target set corresponding to this sequence of regions is not full dimensional, then the node is not considered. Every such route must therefore represent a linear path. □

Theorem IV.2 *The crossing tree contains every possible linear path from x_s to any region in our complex.*

Proof Suppose that this is not the case. This implies that we can find a linear path $(\mathcal{X}_{i_1}, \ldots, \mathcal{X}_{i_m})$ which cannot be traced out with a route from the root node. Assume, without loss of generality that it is the last node, Π_{i_m} that is missing. This implies $\Pi_{i_{m-1}}$ must be missing, since if $\Pi_{i_{m-1}}$ exists and by definition Π_{i_m} is a neighbour, then Π_{i_m} would have been added to the tree. By induction then, $\Pi_{i_{m-2}}$ is missing, and so on until we deduce that even the starting node $\Pi_{i_1} = \Pi_s$ was not added. This is clearly a contradiction. □

To better appreciate the workings of Algorithm 2, we develop a simple example. Before we begin we will require that each node Π_i in our crossing tree consists of the index i of the region $\mathcal{X}_i$ that the node corresponds to , as well as pointers to each of its descendants. Referring once more to Fig. 2a, we shall start tree generation in region $\mathcal{X}_2$ (root node), since it contains our seed x_s. First we list all the neighbours of $\mathcal{X}_s = \mathcal{X}_2$, $\{\mathcal{X}_6, \mathcal{X}_1, \mathcal{X}_3, \mathcal{X}_8, \mathcal{X}_7\}$. For each of the neighbours $\mathcal{N}_2^j$ we ask whether the linear path $(\mathcal{X}_2, \mathcal{X}_{\mathcal{N}_2^j})$ exists.

Clearly, since $\mathcal{X}_2$ contains the seed, we can always cross to every neighbour with a ray, so we add each of $\{\Pi_6, \Pi_1, \Pi_3, \Pi_8, \Pi_7\}$ as a descendant node of the crossing tree, passing to them the associated linear path $(\mathcal{X}_2, \mathcal{X}_{\mathcal{N}_2^j})$. Now we repeat the procedure for each of the descendant nodes. Consider the node Π_6 corresponding to $\mathcal{X}_6$. For each of the elements of $\mathcal{N}_6$ we ask whether $(\mathcal{X}_2, \mathcal{X}_6, \mathcal{X}_{\mathcal{N}_6^j})$ is a linear path. In this case it turns out by inspection, that all four sequences $(\mathcal{X}_2, \mathcal{X}_6, \mathcal{X}_7)$, $(\mathcal{X}_2, \mathcal{X}_6, \mathcal{X}_{12})$, $(\mathcal{X}_2, \mathcal{X}_6, \mathcal{X}_5)$ and $(\mathcal{X}_2, \mathcal{X}_6, \mathcal{X}_1)$ can be intersected by a single ray from x_s. As a result, all of the neighbours of $\mathcal{X}_6$ are added as children of Π_6. We pass down the appropriate linear paths to each of these children, and continue generating our tree recursively. Suppose instead we take Π_1 at the second level. For each element of $\mathcal{N}_1$, we therefore ask whether $(\mathcal{X}_2, \mathcal{X}_1, \mathcal{X}_{\mathcal{N}_1^j})$ is a crossing set. Clearly $(\mathcal{X}_2, \mathcal{X}_1, \mathcal{X}_4)$, $(\mathcal{X}_2, \mathcal{X}_1, \mathcal{X}_3)$, $(\mathcal{X}_2, \mathcal{X}_1, \mathcal{X}_5)$, satisfy the criteria, but $(\mathcal{X}_2, \mathcal{X}_1, \mathcal{X}_6)$ does not since it is impossible to draw a ray starting from x_s, going to a point in $\mathcal{X}_6$, that passes through all three regions in order. Region $\mathcal{X}_6$ will therefore not be added as a descendant of the second level node Π_1. The full crossing tree for the complex in Figs. 2a, 2b, 2c and 2d is shown in Fig. 3.

The final question we must therefore address, is how to determine whether a particular sequence of neighbouring regions forms a linear path from x_s. In Algorithm 2 we solve the equivalent problem of constructing the target set corresponding to the linear path, and then testing whether it is full dimensional. Suppose we are given a possible $\mathcal{I}_{x_s \to \mathcal{X}_{i_m}}$ to check. The task is to compute the subset of $\mathcal{X}_{i_m}$ that can be reached by a straight line from x_s crossing all the facets that separate successive regions in $\mathcal{I}_{x_s \to \mathcal{X}_{i_m}}$. For the example of Fig. 2a, we need to find the set of points that can be reached by a straight line that intersects the facet separating regions $\mathcal{X}_2$ and $\mathcal{X}_1$, the facet separating regions $\mathcal{X}_1$ and $\mathcal{X}_4$ as well as the facet separating regions $\mathcal{X}_4$ and $\mathcal{X}_{10}$.

If $\mathcal{X}_1$ is the jth neighbour of $\mathcal{X}_2$ so that $\mathcal{N}_2^j = 1$, then the facet that forms the intersection of $\mathcal{X}_1$ and $\mathcal{X}_2$ is given by

$$\mathcal{F}_{21} \triangleq \left\{x | H_2^j x = K_2^j, H_{2\setminus\{j\}} x \le K_{2\setminus\{j\}}\right\} \tag{6}$$

Notice however, that if $\mathcal{X}_2$ is the ith neighbour of $\mathcal{X}_1$ we could equivalently have

$$\mathcal{F}_{12} \triangleq \left\{x | H_1^i x = K_1^i, H_{1\setminus\{i\}} x \le K_{1\setminus\{i\}}\right\} \tag{7}$$

These are both valid representations, but we will always choose $\mathcal{F}_{ab}$ if we aim to traverse from region $\mathcal{X}_a$ into region $\mathcal{X}_b$. Sticking to this convention will ensure that the dot product between the normal of the hyperplane describing the facet, H_a^j (if $\mathcal{X}_b$ is the jth neighbour of $\mathcal{X}_a$) and the vector direction of traversal ν is *always positive* if the target set exists. This property will be important later.

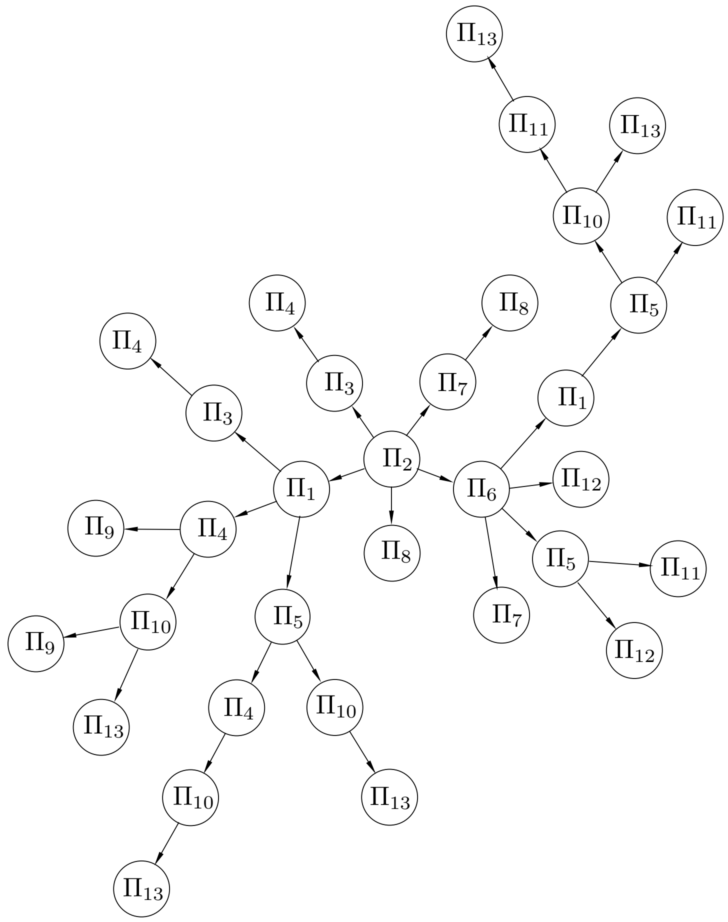

Fig. 3: Crossing Tree for the subdivision walking example

For ease of notation, denote these facets by $\mathcal{F}_i$, $i \in \{1, \ldots, N_f\}$, where $\mathcal{F}_i$ are chosen according to the above convention.

$$\mathcal{F}_i = \left\{x | a_i^T x = b_i, \digamma_i x \le y_i\right\} \tag{8}$$

Theorem IV.3 *Define*

$$Y_i = \begin{bmatrix} (\digamma_i x_s)\, a_i^T + \digamma_i \left(b_i - a_i^T x_s\right) - y_i a_i^T \\ -a_i^T \end{bmatrix} \tag{9}$$

Then the target set is equivalent to the following polyhedron,

$$\mathcal{T}\left(\mathcal{I}_{x_s \to \mathcal{X}_{i_m}}\right) \triangleq \left\{ x \left| \begin{bmatrix} H_{i_m} \\ Y_1 \\ \vdots \\ Y_{N_f} \end{bmatrix} x \le \begin{bmatrix} K_{i_m} \\ Y_1 x_s \\ \vdots \\ Y_{N_f} x_s \end{bmatrix} \right. \right\} \tag{10}$$

If this polyhedron is full dimensional, then the linear path $\mathcal{I}_{x_s \to \mathcal{X}_{i_m}}$ is valid.

Proof The target set can be written as follows,

$$\begin{aligned} \mathcal{T}\left(\mathcal{I}_{x_s \to \mathcal{X}_{i_m}}\right) \triangleq \{x | \exists \{\nu, \lambda_1, \ldots, \lambda_{N_f}\}, \\ (x_s + \lambda_i \nu) \in \mathcal{F}_i \ \forall i \in \{1, \ldots, N_f\}, \\ \nu = x - x_s, x \in \mathcal{X}_{i_m}\} \end{aligned} \tag{11}$$

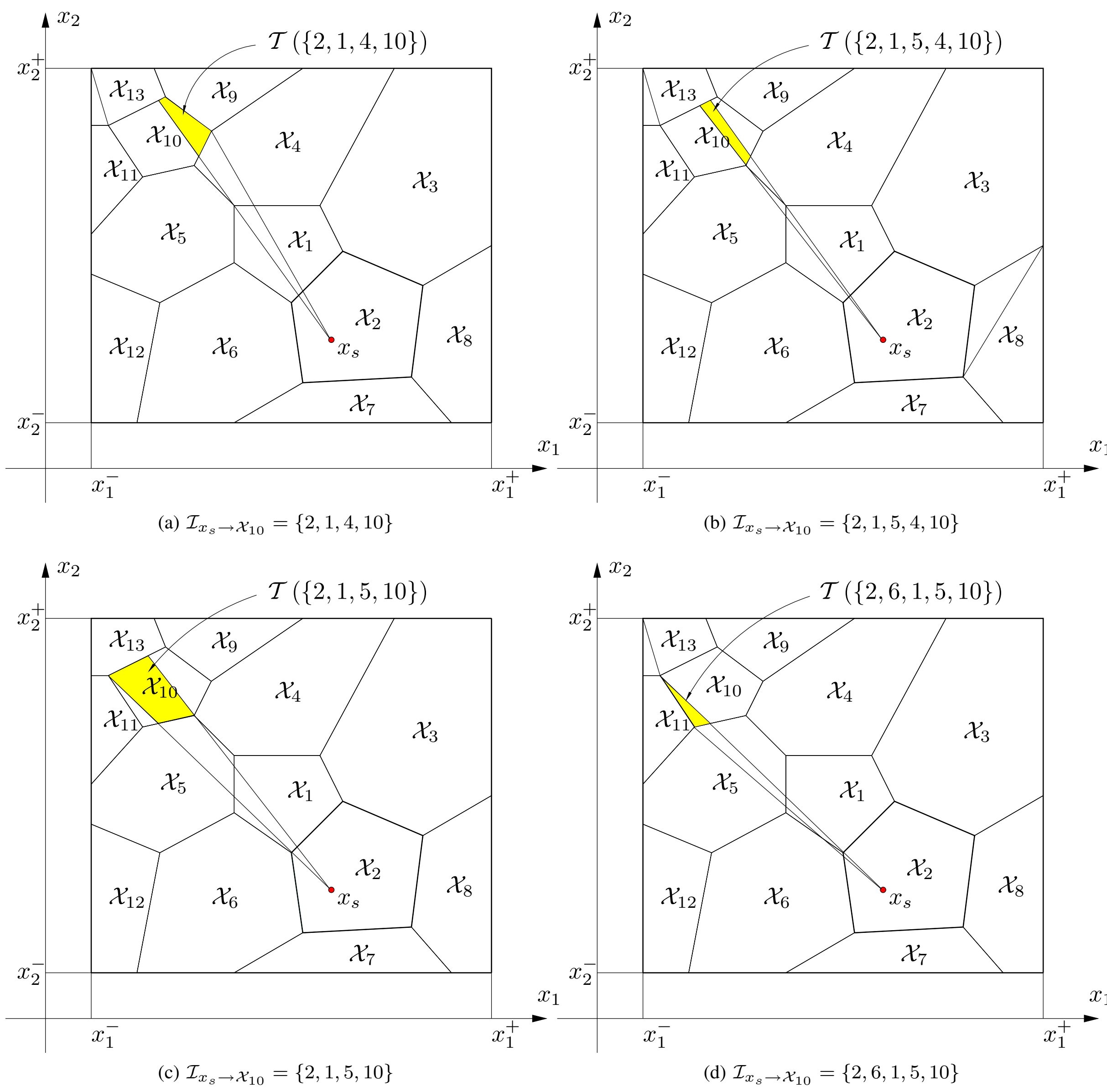
$\mathcal{T}(\{2,1,4,10\})$
$\mathcal{T}(\{2,1,5,4,10\})$
$\mathcal{T}(\{2,1,5,10\})$
$\mathcal{T}(\{2,6,1,5,10\})$
x_1
x_2
x_1^-
x_1^+
x_2^-
x_2^+
x_s
$\mathcal{X}_1$
$\mathcal{X}_2$
$\mathcal{X}_3$
$\mathcal{X}_4$
$\mathcal{X}_5$
$\mathcal{X}_6$
$\mathcal{X}_7$
$\mathcal{X}_8$
$\mathcal{X}_9$
$\mathcal{X}_{10}$
$\mathcal{X}_{11}$
$\mathcal{X}_{12}$
$\mathcal{X}_{13}$
(a) $\mathcal{I}_{x_s \to \mathcal{X}_{10}} = \{2,1,4,10\}$
(b) $\mathcal{I}_{x_s \to \mathcal{X}_{10}} = \{2,1,5,4,10\}$
(c) $\mathcal{I}_{x_s \to \mathcal{X}_{10}} = \{2,1,5,10\}$
(d) $\mathcal{I}_{x_s \to \mathcal{X}_{10}} = \{2,6,1,5,10\}$

Fig. 2: Linear path and Target set

Algorithm 2 Crossing Tree Generation

```
1: procedure CROSSTREE($\mathcal{I}_{x_s \to \mathcal{X}_m}$,$p$)    ▷ $p$ is the index of the parent node
2:     $\mathcal{N}_{m \setminus p} \leftarrow \mathcal{N}_m \setminus \mathcal{I}_{x_s \to \mathcal{X}_p}$    ▷ Remove the current linear path from neighbours
3:     currentregion= $m$
4:     $k \leftarrow 1$
5:     for $j = 1$ to $|\mathcal{N}_{m \setminus p}|$ do
6:         $\mathcal{T}_j = \mathcal{T}(\{\mathcal{I}_{x_s \to \mathcal{X}_m}, \mathcal{N}^j_{m \setminus p}\}) \leftarrow$ gettargetset($\mathcal{I}_{x_s \to \mathcal{X}_m}, \mathcal{N}^j_{m \setminus p}$)    ▷ Construct target set: method explained later
7:         if ($\mathcal{T}_j$) is full dimensional then
8:             childnode{$k$} $\leftarrow$ CrossTree($\{\mathcal{I}_{x_s \to \mathcal{X}_m}, \mathcal{N}^j_{m \setminus p}\}, m$)    ▷ Recursion
9:             $k \leftarrow k + 1$
10:        end if
11:    end for
12:    return node    ▷ node consists of 'currentregion' and 'childnode'
13: end procedure
```

Take the i^{th} facet we need to cross, $\mathcal{F}_i$. Then $(x_s + \lambda_i \nu) \in \mathcal{F}_i$ implies,

$$a_i^T (x_s + \lambda_i \nu) = b_i, \quad F_i (x_s + \lambda_i \nu) \le y_i \tag{12}$$

$$a_i^T x_s + \lambda_i a_i^T \nu = b_i \quad \Rightarrow \quad \lambda_i = \left(b_i - a_i^T x_s\right) / a_i^T \nu \tag{13}$$

Now we can eliminate λ_i, under the assumption that $a_i^T \nu$ is positive if the target set exists,

$$F_i \left(x_s + \left(\left(b_i - a_i^T x_s\right) / a_i^T \nu\right) \nu\right) \le y_i \quad \Rightarrow \tag{14}$$

$$(F_i x_s)\, a_i^T \nu + F_i \left(b_i - a_i^T x_s\right) \nu \le y_i a_i^T \nu, \quad a_i^T \nu \ge 0 \tag{15}$$

$$\left((F_i x_s)\, a_i^T + F_i \left(b_i - a_i^T x_s\right) - y_i a_i^T\right) \nu \le 0, \quad a_i^T \nu \ge 0 \tag{16}$$

Using the definition (9),

$$\mathcal{T}\left(\mathcal{I}_{x_s \to \mathcal{X}_{i_m}}\right) \triangleq \{x | \exists \nu, Y_1 \nu \le 0, Y_2 \nu \le 0, \ldots, Y_{N_f} \nu \le 0, \nu = x - x_s, x \in \mathcal{X}_{i_m}\} \tag{17}$$

With $\nu = x - x_s$,

$$\mathcal{T}\left(\mathcal{I}_{x_s \to \mathcal{X}_{i_m}}\right) \triangleq \{x | \exists \nu, Y_1 x \le Y_1 x_s, Y_2 x \le Y_2 x_s, \ldots, Y_{N_f} x \le Y_{N_f} x_s, H_{i_m} x \le K_{i_m}\} \tag{18}$$

Hence we have shown that the target set can be written in the form (10). From the definition of a linear path, there must exist a ray ρ, such that $\rho \cap \mathcal{X}_{i_m} \neq \emptyset$. Hence a linear path is valid if and only if the interior of the target set relative to its affine hull is nonempty. A sufficient condition is therefore that the polyhedron defined above is full dimensional. □

We can test whether a polyhedron is full dimensional by computing its Chebyshev ball, which requires the solution to one linear program.

V. Simulation Results

We apply the crossing tree method to evaluate the computational savings that subdivision walking achieves on two control problems. All of our simulations are done with the aid of the Multi-Parametric Toolbox for MATLAB [12]. The first is the standard two-dimensional double integrator model, with 513 regions in the explicit solution, as shown in Fig. 4. Defining the origin as the seed point, $x_s = [0, 0]^T$, the maximum crossing tree depth is 33, which means that only 33 regions will have to be searched in the worst case.

The second model is control of a linearized Cessna Citation Aircraft. This is a control problem with a five-dimensional state space, where the objective is to regulate the aircraft altitude. Details of the model and problem can be found in [13]. The closed form solution consists of 403 regions in five dimensions, and the crossing tree generated has a maximum depth of only 24.

VI. Conclusions

It is evident from our simulations that subdivision walking is an applicable, and highly advantageous method. Although it is still a heuristic, it is intuitive that the technique will offer computational savings for a variety of problems. Furthermore, unlike many point location solvers, subdivision walking requires almost no pre-computation. The only information required is the adjacencies of each region, and this can be easily incorporated into the multi-parametric solver itself. The major drawback is the necessity of generating a crossing tree in order to calculate the worst case online computation time, although this step is only performed if an exact bound is required. Tree generation is of course computationally expensive, but the problem is NP-hard and is therefore not likely to have better solutions.

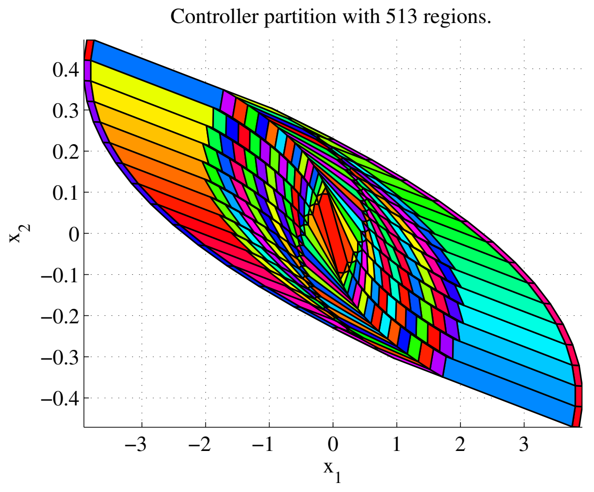

Fig. 4: Double integrator with 513 regions

References

[1] A. Bemporad, M. Morari, V. Dua, and E. Pistikopoulos, "The Explicit Linear Quadratic Regulator for Constrained Systems," *Automatica*, vol. 38, no. 1, pp. 3–20, Jan. 2002.

[2] F. Borrelli, *Constrained Optimal Control of Linear & Hybrid Systems*. Springer Verlag, 2003, vol. 290.

[3] P. Tondel, T. Johansen, and A. Bemporad, "An algorithm for multi-parametric quadratic programming and explicit MPC solutions," in *IEEE Conference on Decision and Control*, 2001, pp. 1199 – 1204.

[4] F. Borrelli, M. Baotic, A. Bemporad, and M. Morari, "Efficient On-Line Computation of Constrained Optimal Control," in *IEEE Conference on Decision and Control*, Orlando, Florida, Dec. 2001, pp. 1187–1192.

[5] P. Tondel, T. Johansen, and A. Bemporad, "Evaluation of piecewise affine control via binary search tree," *Automatica*, vol. 39, no. 5, pp. 945–950, 2003.

[6] C. Jones, P. Grieder, and S. Rakovic, "A Logarithmic-Time Solution to the Point Location Problem for Closed-Form Linear MPC," in *IFAC World Congress*, Prague, Czech Republic, Jul. 2005.

[7] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu, "An optimal algorithm for approximate nearest neighbor searching in fixed dimensions," *Journal of the ACM*, vol. 45, no. 6, pp. 891–923, 1998.

[8] H. J. Ferreau, H. G. Bock, and M. Diehl, "An online active set strategy for fast parametric quadratc programming in MPC applications," in *IFAC Workshop on Nonlinear Model Predictive Control for Fast Systems*, October 2006.

[9] M. Baotic, "An Efficient Algorithm for Multiparametric Quadratic Programming," Tech. Rep., Apr. 2002.

[10] C. Jones, "Polyhedral tools for control," Ph.D. dissertation, Cambridge University, 2006.

[11] J. Spjotvold, E. Kerrigan, C. Jones, P. Tondel, and T. Johansen, "On the facet-to-facet property of solutions to convex parametric quadratic programs," *Automatica*, vol. 42, no. 12, pp. 2209–2214, Dec. 2006.

[12] M. Kvasnica, P. Grieder, and M. Baotić, "Multi-Parametric Toolbox (MPT)," 2004. [Online]. Available: http://control.ee.ethz.ch/ mpt/

[13] J. Maciejowski, *Predictive Control with Constraints*. Prentice-Hall, 2002.