Approximate Semi-explicit and Explicit Hybrid Model Predictive Control via Simplicial Partitioning

Danylo Malyuta\textsuperscript{1} and Behçet Açıkmeşe\textsuperscript{2}

Abstract—For multiparametric mixed-integer convex programs, we propose a partitioning algorithm that generates semi-explicit and explicit guaranteed real-time implementations to within a given suboptimality tolerance. The algorithm is particularly relevant for hybrid model predictive control, where real-time on-line implementation for fast systems is hampered by the worst-case exponential runtime complexity of mixed-integer programming solvers. The output of our algorithm is a binary tree simplicial partition which defines a static map from the current parameter to a suboptimal solution. Convergence of our algorithm is proved by defining a new optimal cost overlap metric, which is linked to the convergence speed. A spacecraft position control example shows that our approach is faster than implicit MPC by up to two orders of magnitude, allowing a guaranteed real-time on-line implementation.

I. INTRODUCTION

This paper develops a provably convergent algorithm for generating semi-explicit and explicit real-time implementations of multiparametric convex mixed-integer nonlinear programs (MINLPs) to within an arbitrarily small suboptimality tolerance. The algorithm is particularly well suited for model predictive control (MPC). Two important types of MPC are hybrid and robust MPC [1]. The former handles systems with discrete switches or piecewise affinely-approximated nonlinearities like chemical powerplants, pipelines and aerospace vehicles [2], [3]. The latter handles uncertain systems [4], [5]. When combined, the two formulations require the solution of a MINLP. If the system dynamics are fast, real-time implementation is hampered by the worst-case exponential runtime complexity of mixed-integer solvers [2].

Several approaches have been proposed to nevertheless attain real-time performance. By leveraging the polynomial runtime complexity of convex solvers, the technique of successive convexification is able to solve nonlinear programs in real-time [6], [7]. Recently, the method was extended to handle binary decision making via state-triggered constraints [8]. Hence, at least some MINLPs may be solved in real-time via successive convexification. However, this is a local method which may not always converge to a feasible solution.

A more traditional method of ensuring real-time MPC performance while guaranteeing convergence and global optimality has been to make the optimal solution “explicit” by moving the optimization off-line. Various explicit MPC methodologies have been proposed [9], [10], [11]. For MPC laws more complicated than linear or quadratic programs, exact explicit solutions are generally not possible due to non-convexity of common active constraint sets [12]. Instead, approximate solutions have been proposed via local linearization [13], [14] or via optimal cost bounding by affine functions over simplices [12], [15], [16] or hyperrectangles [17]. An approximate explicit solution to mixed-integer quadratic programs has been proposed based on difference-of-convex programming [18] and for MINLPs based on local linearization and primal/master subproblems [19], [20].

Our primary contribution is a novel algorithm for approximate semi-explicit and explicit implementation of multiparametric convex MINLPs. The former queries a binary tree for an integer solution and solves a single convex program on-line. The latter queries a binary tree for a solution directly, without on-line optimization. To the best of our knowledge, we are also the first to prove convergence of the partitioning scheme by using a novel definition of an optimal cost “overlap” which is a fundamental determinant of convergence speed and partition complexity.

The paper is organized as follows. Section II defines the class of MPC laws that our algorithm can handle. Section III then presents the algorithm, followed by its convergence and complexity properties in Section IV. The algorithm is extended to an explicit implementation in Section V. The validity of the approach is demonstrated through a spacecraft robust position control example in Section VI. Section VII concludes with future research directions.

Notation: \( A = \{0, 1\} \) is the binary set and \( B = \{x : \|x\|_2 \leq 1\} \) is the unit ball. Matrices are uppercase (e.g. \( A \)), scalars, vectors and functions are lowercase (e.g. \( x \)), and sets are calligraphic uppercase (e.g. \( S \)). The scalar \( \ell \) is a placeholder. \( \cos, S^c, \partial S, \text{vol}(S) \) and \( V(R) \) denote respectively the convex hull, complement, boundary, volume and vertices of \( S \). The cardinality of a countable set \( S \) is \( |S| \). Given \( A \subseteq \mathbb{R}^n \), \( s \in \mathbb{R} \) and \( b \in \mathbb{R}^n \), \( A + b = \{a + b : a \in A\} \) and \( sA = \{sa : a \in A\} \).

II. PROBLEM FORMULATION

Our algorithm can handle the following multiparametric conic MINLPs, previously defined in [21]:

\[
V^*(\theta) = \min_{x, \delta} \ f(\theta, x, \delta) \\
\text{s.t. } g(\theta, x, \delta) = 0, \\
h(\theta, x, \delta) \in \mathcal{K}, \\
\delta \in \mathbb{I}^m,
\]

where \( \theta \in \mathbb{R}^p \) is a parameter, \( x \in \mathbb{R}^n \) is a decision vector and \( \delta \in \mathbb{I}^m \) is a binary vector called the commutation. The
cost function $f : \mathbb{R}^p \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is jointly convex and the constraint functions $g : \mathbb{R}^p \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ and $h : \mathbb{R}^p \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^d$ are affine in their first two arguments. The functions can be nonlinear in the last argument. The convex cone $\mathcal{K} = C_1 \times \cdots \times C_q \subset \mathbb{R}^d$ is a Cartesian product of $q$ convex cones. Examples include the positive orthant, the second-order cone and the positive semidefinite cone. Let us also define a fixed-commutation multiparametric conic NLP:

$$V^*_{\delta}(\theta) = \min_{\theta, x, \delta} f(\theta, x, \delta)$$

s.t. $g(\theta, x, \delta) = 0,$

$h(\theta, x, \delta) \in \mathcal{K},$ (P$_{\theta}^0$

which corresponds to (P$_{\theta}^0$) where $\delta$ has been assigned a specific value. Let $\Theta^* \subseteq \mathbb{R}^p$ and $\Theta^*_\delta \subseteq \Theta^*$ denote respectively the parameter sets for which (P$_{\theta}^0$) and (P$_{\theta}^0$) are feasible. Define the following three maps similarly to [12], [21].

Definition 1. The optimal map $f^*_\delta : \Theta^* \to \mathbb{R}^m$ associates $\theta \in \Theta^*$ to any optimal commutation of (P$_{\theta}^0$), that is any $\delta \in \{ \delta \in \mathbb{R}^m : V^*(\theta) = V^*_{\delta}(\theta) \}.$

Definition 2. The feasible map $f_{\theta} : \Theta^* \to \mathbb{R}^m$ associates $\theta \in \Theta^*$ to a commutation such that (P$_{\theta}^0$) is feasible.

Definition 3. The suboptimal map $f^*_{\theta} : \Theta^* \to \mathbb{R}^m$ associates $\theta \in \Theta^*$ to an $\epsilon$-suboptimal commutation $\delta$ such that

$$V^*_{\delta}(\theta) - V^*(\theta) \leq \epsilon \in \{ \epsilon \in \mathbb{R}^m : V^*(\theta) = V^*_{\delta}(\theta) \}.$$

where $\epsilon$ and $\epsilon_\delta$ are the absolute and relative errors.

This paper presents an algorithm for computing $f^*_{\theta}$ over a subset $\Theta \subseteq \Theta^*$. It is assumed that $\Theta$ is a convex and finite-dimensional polytope in vertex representation. We refer to [21, Section IV-C] for how one might choose $\Theta$.

### III. Suboptimal Map Computation

This section presents an algorithm for computing $f^*_{\theta}$. We target a semi-explicit MPC implementation where $f^*_{\theta}$ yields a $\delta$ that is used to solve the convex problem (P$_{\theta}^0$) on-line.

Recent research proposed an algorithm [21, Algorithm 2] for computing $f_{\theta}$ as a coarse simplicial partition $\mathcal{R} = \{(R_i, \delta_i)\}_{i=1}^{\mathcal{R}}$, that is stored as a binary tree where $(R_i, \delta_i)$ are the leaves. We have $\mathcal{R}_i \subseteq \Theta^*_\delta$ and $\Theta = \bigcup_{i=1}^{\mathcal{R}} \mathcal{R}_i$. To compute $f^*_{\theta}$, we shall refine this partition until $\delta_i$ becomes $\epsilon$-suboptimal within $\mathcal{R}_i$. $f^*_{\theta}$ can then be recovered using:

$$f^*_{\theta}(\theta) = \delta_i$$

Let $(\mathcal{R}, \delta)$ be some partition cell. The algorithm must determine whether $\delta$ is $\epsilon$-suboptimal in $\mathcal{R}$. This is encoded in the following absolute and relative error bilevel MINLPs:

$$e_{\delta}^a(\mathcal{R}) = \max_{\theta \in \mathcal{R}, \delta'} \frac{V^*_{\delta}(\theta) - V^*_{\delta'}(\theta)}{V^*_{\delta}(\theta)},$$

$$e_{\epsilon}^a(\mathcal{R}) = \max_{\theta \in \mathcal{R}, \delta'} \frac{V^*_{\delta}(\theta) - V^*_{\delta'}(\theta)}{V^*_{\delta}(\theta)}.$$

If $e_{\delta}^a(\mathcal{R}) \leq \epsilon_a$ or $e_{\epsilon}^a(\mathcal{R}) \leq \epsilon_a$ then $\delta$ is $\epsilon$-suboptimal in $\mathcal{R}$. Both problems above, however, are non-convex so their optimal solutions are not readily computable. As a remedy, we formulate tractable upper bounds that are sufficient to guarantee $\epsilon$-suboptimality. We focus first on $e_{\epsilon}^a(\mathcal{R})$.

**Definition 4.** Let $\nu_i \in V(\mathcal{R})$ be the $i$-th vertex of $\mathcal{R}$ and let $\theta = \sum_{i=1}^{\mathcal{R}} \alpha_i \nu_i$ with $\alpha_i \in \mathcal{R}$. The affine over-approximator of $V^*_{\delta}(\theta)$ over $\mathcal{R}$ is:

$$\bar{V}_{\delta}(\theta) \triangleq \sum_{i=1}^{\mathcal{R}} \alpha_i V^*_{\delta}(\nu_i).$$

**Proposition 1.** The affine over-approximator (3) satisfies $V^*_{\delta}(\theta) \leq \bar{V}_{\delta}(\theta) \forall \theta \in \mathcal{R}$ and with equality at the vertices.

**Proof.** Since $V^*_{\delta}$ is convex [21, Lemma 1], Jensen’s inequality implies $V^*_{\delta}(\theta) \leq \bar{V}_{\delta}(\theta)$. 

Consider now the following convex MINLP:

$$\bar{e}_a(\mathcal{R}) = \max_{\theta \in \mathcal{R}, \delta' \in \mathbb{R}^m} \bar{V}_{\delta}(\theta) - V^*_{\delta}(\theta).$$

**Lemma 1.** The absolute error satisfies $e_{\epsilon}^a(\mathcal{R}) \leq \bar{e}_a(\mathcal{R})$.

**Proof.** Let $\theta \in \mathcal{R}$ and $\delta' \in \mathbb{R}^m \setminus \delta$. By Proposition 1 $V^*_{\delta}(\theta) \leq \bar{V}_{\delta}(\theta) \Rightarrow V^*_{\delta}(\theta) - V^*_{\delta'}(\theta) \leq \bar{V}_{\delta}(\theta) - V^*_{\delta'}(\theta)$. The result follows directly.

**Theorem 1.** $\bar{e}_a(\mathcal{R}) \leq \epsilon_a$ implies $e_{\epsilon}^a(\mathcal{R}) \leq \epsilon_a$.

**Proof.** By Lemma 1 $e_{\epsilon}^a(\mathcal{R}) \leq \bar{e}_a(\mathcal{R}) \leq \epsilon_a$.

A tractable alternative to (E$_R^\mathcal{A}$) was given in [15]:

$$\bar{e}_\epsilon(\mathcal{R}) = \frac{\min_{\theta \in \mathcal{R}, \delta' \in \mathbb{R}^m} \bar{V}_{\delta'}(\theta) - V^*_{\delta'}(\theta)}{\min_{\theta \in \mathcal{R}, \delta' \in \mathbb{R}^m} \bar{V}_{\delta'}(\theta)}.$$

**Lemma 2.** The relative error satisfies $e_{\epsilon}^a(\mathcal{R}) \leq \bar{e}_\epsilon(\mathcal{R})$.

**Proof.** The result follows from Lemma 1 and the following sequence of (conservative) over-approximations:

$$e_{\epsilon}^a(\mathcal{R}) = \max_{\theta \in \mathcal{R}, \delta' \in \mathbb{R}^m} \frac{V^*_{\delta}(\theta) - V^*_{\delta'}(\theta)}{V^*_{\delta}(\theta)} \leq \max_{\theta \in \mathcal{R}, \delta' \in \mathbb{R}^m} \frac{V^*_{\delta}(\theta) - V^*_{\delta'}(\theta)}{\min_{\theta' \in \mathcal{R}, \delta'' \in \mathbb{R}^m} V^*_{\delta''}(\theta)} \leq \frac{\bar{e}_a(\mathcal{R})}{\min_{\theta \in \mathcal{R}, \delta' \in \mathbb{R}^m} \bar{V}_{\delta'}(\theta)} = \bar{e}_\epsilon(\mathcal{R}).$$

**Theorem 2.** $\bar{e}_\epsilon(\mathcal{R}) \leq \epsilon$ implies $e_{\epsilon}^a(\mathcal{R}) \leq \epsilon$.

**Proof.** By Lemma 2 $e_{\epsilon}^a(\mathcal{R}) \leq \bar{e}_\epsilon(\mathcal{R}) \leq \epsilon$.

Hence the computationally tractable (E$_R^\mathcal{A}$) and (E$_R^\mathcal{R}$) can be used to guarantee $\epsilon$-suboptimality. When both $\bar{e}_a(\mathcal{R}) > \epsilon_a$ and $\bar{e}_\epsilon(\mathcal{R}) > \epsilon_a$, however, (E$_R^\mathcal{A}$) and (E$_R^\mathcal{R}$) are indecisive due to Theorems 1 and 2 being merely sufficient. Section IV shows that a sound remedy is to further subdivide $\mathcal{R}$.

Algorithm 1 computes $f^*_{\theta}$ as follows. First, all leaves of the tree output by [21, Algorithm 2] are re-opened for further
Algorithm 1 Computation of $f_\delta^*$

1: Open all leaves of the tree output by [21, Algorithm 2]  
2: while any open leaf exists do  
3: \( (\mathcal{R}, \delta) \leftarrow \) the most recent open leaf  
4: \( \bar{e}_a(\mathcal{R}) \leftarrow \) solve \( \{ \mathcal{E}_\delta \} \)  
5: \( \bar{e}_t(\mathcal{R}) \leftarrow \) solve \( \{ \mathcal{E}_\delta \} \)  
6: infeasible \( \in \{ \mathcal{E}_\delta \} \) is infeasible  
7: if \( \epsilon \)-suboptimal \( \bar{e}_a(\mathcal{R}) \leq \epsilon_\delta \) or \( \bar{e}_t(\mathcal{R}) \leq \epsilon_\tau \)  
8: if infeasible or \( \epsilon \)-suboptimal then  
9: Close leaf  
10: else  
11: \( \delta^* \leftarrow \) solve \( \{ \mathcal{D}_\delta^* \} \)  
12: if \( \epsilon \)-feasible then  
13: if \( \delta^* \) holds for \( \Theta_\delta = \mathcal{R} \) then  
14: Change open leaf to \( \mathcal{R}, \delta^* \)  
15: else  
16: \( \mathcal{S}_1, \mathcal{S}_2 \leftarrow \) SPLIT(\( \mathcal{R} \))  
17: Add child open leaves \( \mathcal{S}_1, \delta^* \) and \( \mathcal{S}_2, \delta^* \)  
18: else  
19: \( \mathcal{S}_1, \mathcal{S}_2 \leftarrow \) SPLIT(\( \mathcal{R} \))  
20: Add child open leaves \( \mathcal{S}_1, \delta \) and \( \mathcal{S}_2, \delta \)  
21: function SPLIT(\( \mathcal{R} \))  
22: \( \bar{v}, \bar{v}' \leftarrow (\arg \max_v, v' \in V(\mathcal{R})) ||v - v'||_2 \)  
23: \( v_{mid} \leftarrow (\bar{v} + \bar{v}')/2 \)  
24: \( \mathcal{S}_1 \leftarrow \) co\( \{ (V(\mathcal{R}) \setminus \{ \bar{v} \}) \cup \{ v_{mid} \} \} \)  
25: \( \mathcal{S}_2 \leftarrow \) co\( \{ (V(\mathcal{R}) \setminus \{ \bar{v}' \}) \cup \{ v_{mid} \} \} \)  
26: return \( \mathcal{S}_1, \mathcal{S}_2 \)

Lines 4-20 carry out the main work of subdividing the cell \( (\mathcal{R}, \delta) \). First, Theorems 1 and 2 are used to check \( \epsilon \)-suboptimality on lines 4-9. If \( \epsilon \)-suboptimality is verified, the leaf is closed on line 9. Note that if \( \{ \mathcal{D}_\delta^R \} \) fails then \( \delta \) is the only feasible commutation in \( \mathcal{R} \) and is certainly optimal. If, however, \( \epsilon \)-suboptimality cannot be guaranteed then we update \( (\mathcal{R}, \delta) \) on lines 11-20. This either splits \( \mathcal{R} \) and/or improves the choice of \( \delta \). The following convex MINLP attempts to find a more optimal commutation:

\[
\delta^* = \arg \max_{\theta, \delta, \epsilon} \epsilon \\
\text{s.t.} \quad V_\delta(\theta) - \epsilon \geq V_*^\theta(\theta), \quad \epsilon \geq \max \{ \epsilon_\delta, \epsilon_\tau \} \min \{ V_*^\theta \} \\
\quad \theta \in \mathcal{R}, \quad \delta \in \{ \delta' \in \mathcal{D}_\delta \setminus \{ \delta \} : R \subseteq \Theta_{\delta'} \}. \quad (D_\delta^R)
\]

The first two constraints in \( \{ \mathcal{D}_\delta^R \} \) search for a \( \delta^* \) with respect to which \( \delta \) is “the most” suboptimal. Notwithstanding the use of \( V_\delta \) to retain convexity, we shall see in Lemma 5 that once \( \mathcal{R} \) is small enough, \( \delta^* \) is guaranteed to be better than \( \delta \). The third constraint of \( \{ \mathcal{D}_\delta^R \} \) prevents a “greedy” choice of \( \delta^* \) by requiring it to be feasible everywhere in the simplex. Thanks to [21, Lemma 2], this constraint is simple to implement by requiring \( \{ \mathcal{P}_\theta \} \) to be feasible \( \forall \theta \in V(\mathcal{R}) \).

To avoid any unnecessary partitioning, we check on line 13 if \( V_*^\delta \) varies over \( \mathcal{R} \) by less than the \( \epsilon \)-suboptimality threshold. If so, the proof of Theorem 5 assures that \( \mathcal{R} \) need not be subdivided further as far as \( \delta \) is concerned and we can simply change \( \delta \) to \( \delta^* \). If \( \{ \mathcal{D}_\delta^R \} \) does not hold, however, then to guarantee convergence \( \mathcal{R} \) must be subdivided. On lines 22-26 this is done by splitting \( \mathcal{R} \) into two simplices at the midpoint of its longest edge.

If \( \{ \mathcal{D}_\delta^R \} \) is infeasible then \( \mathcal{R} \) must be too big for there to exist a single \( \epsilon \)-suboptimal \( \delta \) within it. In the formal terms of Section IV-A, \( \mathcal{R} \) is larger than the optimal cost “overlap”. Again, the remedy is to split \( \mathcal{R} \) in half on lines 19 and 20.

IV. PROPERTIES

A. Convergence

This section proves in Theorem 5 that Algorithm 1 converges if Assumption 1 holds. Without loss of generality, let us restrict the discussion to feasible commutations in \( \Theta \):

\[
\Delta \triangleq \{ \delta \in \mathcal{I}^m : \Theta^*_\delta \cap \Theta \neq \emptyset \}. \quad (4)
\]

Definition 5. The overlap is the largest \( \gamma \in \mathbb{R}_+ \) such that \( \forall \theta \in \mathcal{I} \exists \delta \in \Delta \) which is \( \epsilon \)-suboptimal in \( (\gamma B + \theta) \setminus \Theta^* \).

Assumption 1. The overlap is positive, i.e. \( \gamma > 0 \).

Previous research [21, Definition 4] considered an overlap \( \kappa \in \mathbb{R}_+ \) between \( \Theta^* \setminus \Theta^*_\delta \) sets. Because we require [21, Algorithm 2] to converge, we need \( \kappa > 0 \) [21, Theorem 1]. This implies that \( \forall \theta \in \Theta \exists \delta \in \Delta \) feasible, so we know \( \gamma \in \mathbb{R}_+ \) exists.

The overlap considered in this work is between sets where a certain commutation is \( \epsilon \)-suboptimal. This is a non-trivial intrinsic property of \( \{ \mathcal{P}_\theta \} \). Assumption 1 is not readily verifiable, but the convergence of Algorithm 1 implies that \( \gamma > 0 \) thanks to Theorem 5 below. Figure 1 illustrates just three overlap possibilities. In the simplest case, local continuity gives positive local overlap as shown in Figure 1a and formalized in the following proposition.

Proposition 2. If \( V_* \) is continuous at \( \theta \in \Theta \setminus \partial \Theta \) and \( \exists \delta \in \Delta \) optimal at \( \theta \) such that \( V_*^\delta \) is continuous at \( \theta \), then the overlap at \( \theta \) is positive.
Proof. Consider the following function based on (1):

\[ f : \Theta^* \triangleq \{ \theta^* \in R^p : \| \theta - \theta^* \|_2 < r \} \rightarrow R, \]

\[ \theta^* \mapsto V^*_\delta (\theta^*) - V^*(\theta^*) - \max \{ \epsilon_a, \epsilon_r V^*(\theta^*) \}. \]

Because of local continuity of \( V^* \) and \( V^*_\delta \), \( \exists r > 0 \) such that \( f \) is continuous. We also have \( f(\theta^*) \in U \triangleq \{ z \in R : z < 0 \} \). Since \( U \) is an open set, \( f^{-1}(U) \subseteq \Theta^* \) is an open set [22]. Hence \( \exists \gamma > 0 \) such that \( f(\theta') \subseteq U \) for all \( \theta' \in \gamma B + \theta \). Therefore, \( \delta \) is \( \epsilon \)-suboptimal in \( \gamma B + \theta \). [Q.E.D.]

More complicated cases of positive overlap exist. For example, Figures [10] and [12] illustrate that if the discontinuity in \( V^* \) is small enough, a positive overlap may still exist. We now define a “variability” metric for \( V^*_\delta \), which influences convergence speed.

**Definition 6.** The variability for a given \( \delta \in \Delta \) is the largest constant \( \sigma_{\delta} \in \mathbb{R}^+ \) such that

\[ \max_{\theta \in \Theta_{\sigma}} V^*_\sigma (\theta) - \min_{\theta \in \Theta_{\sigma}} V^*_\sigma (\theta) < \min_{\theta \in \Theta_{\sigma}} \{ \epsilon_a, \epsilon_r V^*(\theta) \}, \quad (5) \]

where \( \Theta_{\sigma} \triangleq \{ \sigma_{\delta} B + \theta \} \ \forall \ \theta \in \Theta_{\delta} \).

If \( V^*_\sigma \) changes little, \( \sigma_\delta \) will be relatively large. On the other hand, a \( V^*_\sigma \) with high gradients will yield a small \( \sigma_\delta \). It shall be useful to define a \( \{ P_{\delta} \} \) “condition number”:

\[ \psi \triangleq \min_{\delta \in \Delta} \{ \gamma, \min_{\delta} (\sigma_{\delta}) \}. \quad (6) \]

In proving convergence, we want to be sure that Algorithm [1] does not “oscillate” forever between \( \delta \) choices. This is guaranteed by the following lemma.

**Lemma 3.** Consider the cell \( \{ R, \delta \} \). Suppose that \( R \subseteq (\psi B + \theta_{\sigma}) \) for some \( \theta_{\sigma} \in \Theta \). If a more optimal \( \delta_{\sigma} \) is found on line 7 of Algorithm [7], then \( R \) will never again be associated with \( \delta \), i.e., the cell \( \{ R, \delta \} \) will not re-appear.

**Proof.** We begin by showing that

\[ \min_{\delta \in \Delta} V^*_\delta (\theta) < \min_{\theta \in \Theta} V^*_\delta (\theta). \quad (7) \]

Due to the first two constraints of (D8), \( \exists \tilde{\delta} \in R \) such that

\[ V_\delta (\theta) - \max_{\theta \in \Theta} \{ \epsilon_a, \epsilon_r V^*(\theta) \} \leq V^*_\delta (\tilde{\theta}), \quad (8) \]

Since \( R \subseteq (\sigma_{\delta} B + \theta_{\sigma}) \ \forall \ \theta \in \Theta_{\delta} \), we have the following set of inequalities thanks to (5):

\[ \tilde{V}_\delta (\theta) \leq \max_{\delta \in \Delta} V^*_\delta (\theta) \leq \max_{\theta \in \Theta_{\sigma}} V^*_\delta (\theta) \]

\[ < \min_{\theta \in \Theta_{\sigma}} \{ \epsilon_a, \epsilon_r \min_{\delta \in \Delta} V^*(\theta) \} \]

\[ \leq \min_{\theta \in \Theta_{\sigma}} \{ \epsilon_a, \epsilon_r \min_{\theta \in \Theta_{\sigma}} V^*(\theta) \} \quad (9) \]

Using (9) in (8), we have

\[ \min_{\theta \in \Theta_{\sigma}} V^*_\delta (\theta) > V^*_\delta (\tilde{\theta}), \]

hence (7) must hold. Since \( R \in (\psi B + \theta_{\sigma}) \), Algorithm [1] can only ever finish the iteration on line 9 or 14 i.e. \( R \) is not subdivided. Instead, future iterations may associate it with more optimal commutations. Consider now some future iteration where the cell is \( \{ R, \delta' \} \). By recursively applying (7), it is clear that:

\[ \min_{\delta \in \Delta} V^*_\delta (\theta) < \cdots < \min_{\delta \in \Delta} V^*_\delta (\theta) < \min_{\delta \in \Delta} V^*_\delta (\tilde{\theta}). \quad (10) \]

By contradiction, suppose that \( \delta \) is chosen on line 11 which means that \( \exists \tilde{\theta} \in R \) such that

\[ V_{\delta'} (\tilde{\theta}) - \max_{\theta \in \Theta_{\delta'}} \{ \epsilon_a, \epsilon_r \min_{\theta \in \Theta_{\delta'}} V^*(\theta) \} \geq V^*_\delta (\tilde{\theta}). \quad (11) \]

In the same way that we obtained (9), we have:

\[ \tilde{V}_{\delta'} (\tilde{\theta}) < \min_{\theta \in \Theta_{\delta'}} V^*_\delta (\theta) + \max_{\theta \in \Theta_{\delta'}} \{ \epsilon_a, \epsilon_r \min_{\theta \in \Theta_{\delta'}} V^*(\theta) \} \quad (12) \]

Using (12) in (11), we have

\[ \min_{\delta \in \Delta} V^*_\delta (\theta) > V^*_\delta (\tilde{\theta}) \Rightarrow \min_{\delta \in \Delta} V^*_\delta (\theta) > \min_{\delta \in \Delta} V^*_\delta (\theta), \]

which contradicts (10), hence \( \delta \) cannot be more optimal than \( \delta' \) and thus cannot be re-associated with \( R \). [Q.E.D.]

The next lemma assures that Algorithm [1] decreases the partition cell size until Lemma 5 applies. This shall be instrumental to the convergence proof in Theorem 3.

**Lemma 4.** Let \( \{ R_k, \delta_k \} \) be the leaf chosen at the \( k \)-th call of line 2 of Algorithm [7]. If Assumption [7] holds and the algorithm does not terminate sooner, there will be a \( k \) large enough such that \( R_k \subseteq (\psi B + \theta_{\sigma}) \) for some \( \theta_{\sigma} \in \Theta \).

**Proof.** An iteration can finish in one of four ways: on line 9 [14] or 17 [20]. The first way removes a leaf from consideration, so need not be considered. In the latter two cases, \( R_k \) is split along its longest edge. It can be shown that this halves its volume such that, as long as some iterations finish on lines [17] or [20] \( \exists k \) large enough such that \( R_k \subseteq (\psi B + \theta_{\sigma}) \). Hence, if a large enough \( k \) does not exist, it must be because iterations persistently finish on line 14. Without loss of generality, suppose that \( R_k \) is the last open simplex left. The sequence of open leaves seen by the algorithm is then \( \{ R_k, \delta_k \}, \{ R_{k+1}, \delta_{k+1} \}, \{ R_{k+2}, \delta_{k+2} \}, \ldots \) where \( R_j \subseteq (\sigma_{\delta_j} B + \theta_{\sigma_j}) \) \( \forall j \geq k \) and \( R_k = R_j \forall j \geq k \). However, since \( |\Delta| \) is finite, after a finite number of iterations this implies that \( R_k \subseteq (\min_{\delta \in \Delta} (\sigma_{\delta} B + \theta_{\sigma})) \). Furthermore, generally \( R_k \subseteq (\gamma B + \theta_{\sigma}) \) since otherwise \( \{ D_{\delta_k} \} \) could fail and an iteration would finish on line 20. Hence \( R_k \subseteq (\psi B + \theta_{\sigma}) \). [Q.E.D.]

**Theorem 3.** Algorithm [1] converges if and only if Assumption 7 holds.

**Proof.** Let \( \{ R_k, \delta_k \} \) be the leaf chosen at the \( k \)-th call of line 3 and suppose that Assumption [7] holds. By Lemma 4 if the algorithm did not already terminate then there will be a \( k \) large enough such that \( R_k \subseteq (\psi B + \theta_{\sigma}) \) for some \( \theta_{\sigma} \in \Theta \). By Definition 5 \( \exists \delta \in \Delta \) that is \( \epsilon \)-suboptimal in \( R_k \). By Lemma 3 if \( \delta_k \) is not \( \epsilon \)-suboptimal then it will be replaced with a more optimal commutation and never again be re-associated with \( R_k \). Furthermore, we will have that \( \{ D_{\delta_k} \} \) is
feasible and that (5) holds. Hence, every iteration will exit through line 9 or 14. Since |Δ| is finite, eventually only a single commutation will be left to consider, which must then be the ϵ-suboptimal one. We refer to this as “monotonic” convergence to the ϵ-suboptimal commutation.

If Assumption 1 does not hold, ∃θ ∈ Θ such that if θ ∈ R_k then ∃δ-ϵ-suboptimal in R_k if and only if R_k = {θ}. Since this occurs for k = ∞, the algorithm does not converge.

Theorem 3 suggests that if ψ is large, simplices need to be partitioned relatively little until their small size assures monotonic convergence to an ϵ-suboptimal δ. We note that high cost function gradients decrease ψ while coarser tolerances (i.e. higher ϵ_a and ϵ_r) increase ψ. While the former are a property of (P_o), the latter are adjustable by the user.

B. Complexity

The main result of this section is Corollary 1, which states the on-line MPC implementation has polynomial runtime complexity. We assume that Assumption 1 holds.

Theorem 4. The depth τ of the tree output by Algorithm 7 is O(p^2 log(ψ^{-1})).

Proof. Consider an initial cell (R, δ) output by [21, Algorithm 2] and let R_k ⊂ R be a child simplex of R after k subdivisions. In the worst case, the iteration history of Algorithm 7 can be viewed in two stages. By Lemma 3, the first stage reduces the size of R_k until R_k ⊂ (ψ_B + θ_o) for some θ_o ∈ Θ. By Lemma 2, the second stage iterates through all possible δ ∈ Δ until the last remaining commutation value, which must be ϵ-suboptimal. Only the first stage increases the tree depth because once R_k ⊂ (ψ_B + θ_o), R_k is feasible and (5) holds, hence all subsequent iterations exit through lines 9 or 14. The tree depth is hence governed by the condition number ψ. It was shown in [21, Theorem 2] that, assuming l_0 is the longest edge length of R, the approximate required number of subdivisions is

\[ \tau = \left[ \frac{p(p + 1) \log_2(l_0/\psi)}{2} \right], \]

which yields τ = O(p^2 log(ψ^{-1})).

Theorem 5. The on-line evaluation complexity of f(S) via 2 is O(p^3).

Proof. The result was proved in [21, Theorem 3].

Corollary 1. The semi-explicit implementation of (P_o) has runtime complexity O(p^3 + n^f).

Proof. The semi-explicit implementation evaluates f(S) via 2 and solves one convex program (P_o). The latter operation is polynomial time O(n^f) [23] which, together with Theorem 5, yields an overall complexity O(p^3 + n^f).

Corollary 4 stands in contrast to implementing (P_o) directly with a mixed-integer solver, which has an exponential runtime O(n^f 2^m). Note that typically p ≪ m, so p^3 ≪ 2^m.

V. Extension to Explicit MPC

This section modifies Algorithm 1 for an explicit implementation where no optimization happens on-line. First, we recognize that a convex combination of vertex optimal decision vectors lower bounds ˆV_δ.

Theorem 6. Let (R, δ) be a cell of the f(S) partition. Let v_i ∈ V(R) be the i-th vertex and x_i ∈ R^n the associated optimal decision vector for (P_i) where x = v_i. Consider θ ∈ R such that θ = \sum_{i=1}^{V(R)} α_i v_i with α_i ∈ R. Let

\[ x^* \triangleq \sum_{i=1}^{V(R)} α_i x_i \] (14)

and ˆV_δ(θ) ≜ f(θ, x^*, δ). Then x is feasible for (P_o) and

\[ ˆV_δ(θ) \leq \bar{V}_δ(θ). \] (15)

Proof. Since g(·, ·) and h(·, ·) are affine,

\[ g(θ, x^*, δ) = \sum_{i=1}^{V(R)} α_i g(v_i, x_i, δ) = 0, \]

\[ h(θ, x^*, δ) = \sum_{i=1}^{V(R)} α_i h(v_i, x_i, δ) \in K, \]

so x is feasible for (P_o). Next, since f(·, ·) is convex,

\[ V_δ(θ) \leq \sum_{i=1}^{V(R)} α_i f(v_i, x_i, δ) = \sum_{i=1}^{V(R)} α_i V^*_o(v_i) = \bar{V}_δ(θ). \]

Theorem 4 gives a direct expression [14] for the ϵ-suboptimal decision vector, which can be evaluated as long as the x_i’s are known. To this end, let Algorithm 1 store the closed leaves with this information, i.e. (R, δ, {x_i}_{i=1}^{V(R)}).

It remains to ensure that x^* is ϵ-suboptimal. By definition, this requires \bar{V}_δ(θ) - V^*_o(θ) ≦ max(\epsilon_a, \epsilon_r V^*_o(θ)) where V^* considers all δ ∈ R^m. To this end, the tractable absolute and relative error over-approximators are updated to also check for ϵ-suboptimality with respect to δ itself:

\[ \hat{e}_a(R) = \max_{θ ∈ R, δ ∈ R^m} \bar{V}_δ(θ) - V_o^∗(θ), \] (E_a)\]

\[ \hat{e}_r(R) = \hat{e}_a(R)/\epsilon_r \] (E_r)\]

We then use (E_a) and (E_r) instead of (E_a) and (E_r) in Algorithm 1. The following theorem guarantees that ˆV_δ(θ), and hence x^* in (14), is ϵ-suboptimal.

Theorem 7. ˆV_δ(θ) obtained using (E_a) and (E_r) in Algorithm 1 is ϵ-suboptimal.

Proof. Using (15), the absolute error component of (1) holds:

\[ \bar{V}_δ(θ) - V^*_o(θ) \leq \hat{V}_δ(θ) - V^*_o(θ) \leq \epsilon_a(R) \leq \epsilon_a. \]

The relative error component of (1) follows from:

\[ \hat{e}_a(R) = \hat{e}_r(R) \min_{θ ∈ R} V^*_o(θ) \leq \epsilon_r V^*_o(θ). \]
Given a parameter $\theta$, the partition can be used to obtain the $\epsilon$-suboptimal decision vector directly via a tree query:
\[
x^* = \sum_{\delta_i \in V(R)} \alpha_i x_i \quad \text{such that } \theta = \sum_{\delta_i \in V(R)} \alpha_i v_i \in V(R),
\]
(16)

VI. ILLUSTRATIVE EXAMPLE

We apply Algorithm 1 to minimum-fuel position control of a spacecraft. A full description of the problem and of the MPC law is given in [5]. Here we provide a brief summary. Consider Clohessy-Wiltshire out-of-orbital plane dynamics:
\[
\ddot{z} = -\omega_0^2 z + u + w,
\]
(17)
where $\omega_0$ rad/s is the orbital rate. Let the state $x = (z, \dot{z}) \in \mathbb{R}^2$, input $u \in \mathbb{R}$ and disturbance $w \in \mathbb{R}$. (17) is discretized using a reaction control system (RCS) input that instantaneously induce a $\Delta v$ velocity increment every $T_s = 100$ s, i.e. $u(t) = \Delta v(kT_s)\delta_0(t-kT_s)$ where $k \in \mathbb{Z}_+$ and $\delta_0$ is the Dirac delta. The disturbance models atmospheric drag, input-dependent input error and state-dependent state estimate error. We use $\Theta = [-10, 10] \times [-1, 1] \text{ cm/s}$ which is robust controlled invariant for (17). Thruster impulse-bit imposes a non-convex lower bound on the $\Delta v$ magnitude,
\[
\Delta v \in \{ \Delta v \in \mathbb{R} : \Delta v \leq |\Delta v| \leq \bar{\Delta v} \} \cup \{0\},
\]
which is handled via mixed-integer programming by constraining the input to be in one of three convex sets: $[-\bar{\Delta v}, 0]$, $[0, \bar{\Delta v}]$ or $\{0\}$. Altogether, (P) is a mixed-integer second-order cone program (SOCP). Using a 3-step prediction horizon, the problem has parameter dimension $p = 2$ (the state) and commutation dimension $m = 9$.

We use four increasingly coarse $\epsilon$-suboptimality settings. The relative error is set to $\epsilon_r = 0.05, 0.1, 1, 2$. For the absolute error, we use the maximum optimal cost at the vertices of a scaled $\Theta$, i.e.
\[
\epsilon_a = \max_{\theta \in V(\Theta)} V^*(\theta),
\]
where $s = 0.03, 0.1, 0.25, 0.5$. We use Python 3.6.7 with CVXPY 1.0.14 [24] and Gurobi 8.1 [25] in Ubuntu 18.04.1 with a 3.60 GHz Intel i7-6850K CPU and 64 GB of RAM.

Figure 2 shows convergence plots in terms of the closed leaf count and volume fractions. A linear convergence is preferred since it is more easily extrapolated to predict total runtime. The more refined partitions favorably exhibit a more linear convergence. Since the curves in Figure 2 stay mostly below the linear convergence reference, the user can expect faster runtimes than predicted by a linear fit.

Table I summarizes the results. The data confirms that $\tau = \mathcal{O}(\log(\psi^{-1}))$ using the proxy $\psi \approx \epsilon_a/\epsilon_a + \epsilon_r/\epsilon_r$ where $\epsilon_a$ and $\epsilon_r$ are the largest of the tested values. The leaf count is exponential in $\psi$, with the semi-explicit partitions being generally coarser since (P$^\text{R}$) and (P$^\text{E}$) are easier to satisfy than (P$^\text{R}$) and (P$^\text{E}$). It follows that $T_{\text{solve}}$ and $M$ are also exponential in $\psi$.

Table II and Figure 3A show statistics for the on-line control input computation time $T_{\text{query}}$. For implicit MPC, $T_{\text{query}}$ is the solution time of (P). For semi-explicit MPC, it is the time to evaluate (2) and solve (P$^\text{E}$). For explicit MPC, it is the evaluation time of (16). As expected, explicit is the fastest and implicit is the slowest. $T_{\text{query}}$ is more variable for semi-explicit than implicit MPC due to a non-trivial dependence on the number of iterations required to solve the SOCP (P$^\text{E}$). Even with the most refined partition, the explicit implementation is up to two orders of magnitude faster than implicit MPC. More importantly, real-time execution of both the explicit and semi-explicit implementations is guaranteed.

Meanwhile, if we take $T_{\text{socp}} = 1$ ms as the approximate (P$^\text{E}$) solution time, then the worst-case runtime of implicit MPC is $\approx 27T_{\text{socp}} = 27$ ms where $27 = 3^3$ is the number of input subset choices over the 3-step prediction horizon.

Figure 3B characterizes the extent of fuel consumption suboptimality with respect to implicit MPC. We see that the finer partitions induce a smaller fuel penalty, with down to $< 20\%$ fuel over-consumption. Figure 4 shows the temporal response for each implementation, comparing the coarsest to the most refined partition. The input is actuated a lot more for the coarsest partition, hence the higher fuel consumption.

A. Improving the Storage Memory Requirement

Our binary tree implementation is not memory-optimized, so the $M$ values in Table II are far greater than necessary. Two more efficient storage models are possible.

The first model stores the unique simplex vertices and each “left” child node references its $p + 1$ vertices via integer indices. For explicit MPC, “right” child leaf nodes also reference their vertices. Lastly, all leaves store their $\epsilon$-suboptimal commutation in the semi-explicit case, or the $\epsilon$-suboptimal decision vectors at each vertex in the explicit case. Assuming a perfect binary tree and that $\Theta$ is a simplex, the semi-explicit and explicit storage memory requirements
TABLE I: Numerical results for several \( \epsilon \)-suboptimality settings. \( \tau \) is the tree depth, \( \lambda \) is the leaf count, \( T_{\text{solve}} \) is the Algorithm runtime, \( T_{\text{query}} \) is the median control input evaluation time and \( M \) is the storage memory requirement.

| Implementation | \( \epsilon_a \) | \( \epsilon_r \) | \( \tau \) | \( \lambda \) | \( T_{\text{solve}} \) [min] | \( T_{\text{query}} \) [\( \mu \)s] | \( M \) [MB] |
|----------------|----------------|----------------|---------|------------|-----------------|----------------|--------|
| Semi-explicit  | \( 4.31 \cdot 10^{-2} \) | \( 2.00 \) | 12      | 85         | 1               | 611             | 0.06    |
| Explicit       | \( 4.31 \cdot 10^{-2} \) | \( 2.00 \) | 12      | 85         | 1               | 106             | 0.06    |
| Semi-explicit  | \( 1.05 \cdot 10^{-2} \) | \( 1.00 \) | 14      | 532        | 15              | 630             | 0.39    |
| Explici         | \( 1.05 \cdot 10^{-2} \) | 1.00 | 14      | 532        | 14              | 133             | 0.39    |
| Semi-explicit  | \( 1.70 \cdot 10^{-3} \) | 0.10 | 17      | 6844       | 211             | 695             | 5.10    |
| Explicit       | \( 1.70 \cdot 10^{-3} \) | 0.10 | 17      | 7726       | 219             | 188             | 5.66    |
| Semi-explicit  | \( 1.76 \cdot 10^{-4} \) | 0.05 | 22      | 56189      | 1878            | 491             | 42.38   |
| Explicit       | \( 1.76 \cdot 10^{-4} \) | 0.05 | 22      | 60593      | 1852            | 244             | 45.26   |

(a) MPC on-line evaluation time. Bars show the median while error bars show the minimum and maximum values.

Fig. 3: Comparison of the proposed semi-explicit and explicit implementations to implicit MPC in terms of (a) on-line control input computation time and (b) total fuel consumption over 3 orbits.

(b) Relative over consumption of fuel with respect to implicit MPC due to \( \epsilon \)-suboptimality. Implicit MPC uses \( \approx 1.56 \) mm of “fuel”.

| Number of orbits | Input [\( \mu \)m/s] | Position [mm] | Velocity [mm/s] |
|------------------|-----------------------|---------------|-----------------|
| Semi-explicit    | implicit              | 0.0           | 0.0             |
|                  | \( \epsilon_a = 4.31 \cdot 10^{-3} \), \( \epsilon_r = 5.00 \cdot 10^{-2} \) |
| Explicit         |                       | 0.0           | 0.0             |
|                  | \( \epsilon_a = 1.76 \cdot 10^{-1} \), \( \epsilon_r = 2.00 \) |

Fig. 4: State and input responses for a coarse (blue) and a refined (red) partition for semi-explicit and explicit implementations. Implicit MPC response is in gray.

are, respectively:

\[
M^e_1 \approx \lambda(\mu_0 p + (p + 1) + \mu_0 m) + \mu_0 p^2 ,
\]

\[
M^e_2 \approx \lambda(\mu_0 p + (p + 1)(\frac{3}{2}n + \mu_1 n)) + \mu_1 p^2 ,
\]

where \( \mu_1 \), \( \mu_0 \) and \( \mu_0 \) are the floating point, integer and Boolean sizes, and \( \hat{n} \leq n \) is the decision vector part that is necessary for control input computation (e.g. the first control input for MPC). In the case of the last two rows of Table I we have \( M_{1c}^e \approx 1.98 \text{ MB} \ll M \) and \( M_{2c}^e \approx 3.35 \text{ MB} \ll M \). Using \texttt{float64}, \texttt{uint32} and \texttt{char} types. The memory requirement for the actual binary trees differs by \(< 0.01 \text{ MB} \) from these theoretical values.

This storage model, however, has the downside of requiring to invert a \( \mathbb{R}^{p \times p} \) matrix on-line in order to evaluate in \( \theta \in \mathbb{R} \) in (12) or (16). The matrix inverse, however, can be computed off-line and stored instead of the cell vertices. Again assuming a perfect binary tree and that \( \Theta \) is a simplex, the storage memory requirements for this second model are:

\[
M^e_1 \approx \frac{3}{2} \lambda(\mu_0 p + (p + 1) + \lambda \mu_1 n) \hat{n} ,
\]

Again for the last two rows of Table I we have \( M_{1c}^e < M_{1c}^e \approx 3.05 \text{ MB} \ll M \) and \( M_{2c}^e < M_{2c}^e \approx 5.55 \text{ MB} \ll M \). While greater economy is possible by eliminating further redundancy in the stored vertices and optimal decision vectors, these two storage models show that the memory required by the resulting partition need not be prohibitive. For example, the partition can be stored on a NOR external memory device [26] and interfaced to a microprocessor.

VII. CONCLUSION AND FUTURE WORK

This paper presented a partitioning algorithm for semi-explicit and explicit MPC that is applicable to multiparametric convex MINLPs. The algorithm is guaranteed to converge given positivity of a novel cost function “overlap” metric. The resulting MPC implementation is guaranteed real-time. Future work involves proving stability of the resulting control law along the lines of [15], [16], and massively parallelizing the tree exploration via the map parallel pattern [27].
VIII. ACKNOWLEDGMENT

The authors would like to thank Martin Cacan, David S. Bayard, Daniel P. Scharf, Jack Aldrich and Carl Seubert of the NASA Jet Propulsion Laboratory, California Institute of Technology, for their helpful insights and discussions.

REFERENCES

[1] D. Q. Mayne, “Model predictive control: Recent developments and future promise,” Automatica, vol. 50, pp. 2967–2986, dec 2014.
[2] A. Bemporad and M. Morari, “Control of systems integrating logic, dynamics, and constraints,” Automatica, vol. 35, pp. 407–427, mar 1999.
[3] L. Blackmore, B. Açıkmeşe, and J. M. Carson III, “Lossless convexification of control constraints for a class of nonlinear optimal control problems,” Systems & Control Letters, vol. 61, pp. 863–870, aug 2012.
[4] A. Bemporad and M. Morari, “Robust model predictive control: A survey,” in Robustness in identification and control, pp. 207–226, Springer London, 2007.
[5] D. Malyuta, B. Açıkmeşe, and M. Cacan, “Robust model predictive control for linear systems with state and input dependent uncertainties,” in 2019 American Control Conference (accepted), p. arXiv:1902.10984, IEEE, jul 2019.
[6] Y. Mao, D. Dueri, M. Szmuk, and B. Açıkmeşe, “Successive convexification of non-convex optimal control problems with state constraints,” IFAC-PapersOnLine, vol. 50, pp. 4063–4069, jul 2017.
[7] R. Bonalli, A. Canlìgi, A. Bylard, and M. Pavone, “GuSTO: Guaranteed sequential trajectory optimization via sequential convex programming,” in 2019 IEEE International Conference on Robotics and Automation (ICRA), IEEE, may 2019.
[8] M. Szmuk, T. Reynolds, B. Açıkmeşe, M. Mesbahi, and J. M. Carson III, “A tutorial on successive convexification for real-time rocket landing guidance with state-triggered constraints,” in AIAA Scitech 2019 Forum, American Institute of Aeronautics and Astronautics, jan 2019.
[9] A. Alessio and A. Bemporad, “A survey on explicit model predictive control,” in Nonlinear Model Predictive Control, pp. 345–369, Springer Berlin Heidelberg, 2009.
[10] A. Bemporad, “Model predictive control design: New trends and tools,” in Proceedings of the 45th IEEE Conference on Decision and Control, IEEE, 2006.
[11] E. N. Pistikopoulos, L. Dominguez, C. Panos, K. Kouramas, and A. Chinchuluun, “Theoretical and algorithmic advances in multi-parametric programming and control,” Computational Management Science, vol. 9, pp. 183–203, apr 2012.
[12] A. Bemporad and C. Filippi, “An algorithm for approximate multi-parametric convex programming,” Computational Optimization and Applications, vol. 35, pp. 87–108, mar 2006.
[13] E. N. Pistikopoulos, M. C. Georgiadis, and V. Dua, eds., MultiParametric Programming: Volume 1: Theory, Algorithms, and Applications, Volume 1, vol. 1. Wiley-VCH Verlag GmbH & Co. KGaA, feb 2007.
[14] R. H. Overdieck, Theoretical and algorithmic advances in multi-parametric programming and control. Dissertation (Ph.D.), Imperial College London, 2017.
[15] M. de la Peña, A. Bemporad, and C. Filippi, “Robust explicit MPC based on approximate multi-parametric convex programming,” in 2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601), IEEE, 2004.
[16] D. M. D. L. Peña, A. Bemporad, and C. Filippi, “Robust explicit MPC based on approximate multiparametric convex programming,” IEEE Transactions on Automatic Control, vol. 51, pp. 1399–1405, aug 2006.
[17] T. A. Johansen, “Approximate explicit receding horizon control of constrained nonlinear systems,” Automatica, vol. 40, pp. 293–300, feb 2004.
[18] A. Alessio and A. Bemporad, “Feasible mode enumeration and cost comparison for explicit quadratic model predictive control of hybrid systems,” IFAC Proceedings Volumes, vol. 39, no. 5, pp. 302–308, 2006.
[19] V. Dua and E. Pistikopoulos, “An outer-approximation algorithm for the solution of multiparametric MINLP problems,” Computers & Chemical Engineering, vol. 22, pp. S955–S958, mar 1998.
[20] C. Rowe and J. Maciejowski, “An algorithm for multiparametric mixed integer semidefinite optimization,” in 42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475), IEEE, 2003.
[21] D. Malyuta, B. Açıkmeşe, M. Cacan, and D. S. Bayard, “Partition-based feasible integer solution pre-computation for hybrid model predictive control,” in 2019 European Control Conference (accepted), p. arXiv:1902.10989, IFAC, jun 2019.
[22] H. Royden, Real Analysis. Pearson, 3 ed., 1988.
[23] D. Dueri, B. Açıkmeşe, D. P. Scharf, and M. W. Harris, “Customized real-time interior-point methods for onboard powered-descent guidance,” Journal of Guidance, Control, and Dynamics, vol. 40, pp. 197–212, feb 2017.
[24] S. Diamond and S. Boyd, “CVXPY: A Python-embedded modeling language for convex optimization,” Journal of Machine Learning Research, vol. 17, no. 83, pp. 1–5, 2016.
[25] L. Gurobi Optimization, “Gurobi optimizer reference manual,” 2018.
[26] Micron Technology, Inc., 1Gb, 3V , Multiple I/O Serial NOR Flash Memory, 2011. Rev. M 03/14 EN.
[27] M. McCool, A. D. Robison, and J. Reinders, Structured Parallel Programming: Patterns for Efficient Computation. Elsevier, 2012.