Simplification of Explicit MPC Feedback Laws via Separation Functions
Michal Kvasnica ∗,1, Ivana Rauová ∗, and Miroslav Fikar*

Abstract: We consider the problem of reducing the memory footprint of explicit MPC feedback laws for linear systems. A controller defined by a continuous Piecewise Affine (PWA) function is simplified using separation functions. If a state resides in a saturated region, control value is given by the sign of the separator. Thus, instead of storing and evaluating all regions, only the unconstrained regions and the separator are needed. A large case study is provided to evidence that such separators can be constructed efficiently even for very complex explicit MPC solutions.

Keywords: model predictive control, constrained control, parametric optimization

1. INTRODUCTION

Real-time implementation of MPC in the Receding Horizon fashion (RHMPC) boils down to repetitively solving a given optimization problem for a given value of the initial condition x. However, implementing such a policy in real time requires significant computational resources, which might not be available. The problem becomes especially prominent when sampling times in the range of milliseconds are to be achieved. One way around, as shown in Bemporad et al. (2002), is to precompute the explicit RHMPC optimizer given optimization problem for a given value of the initial state. Such a function is then composed of numerous distinct affine feedbacks defined over a set of polytopic regions. Due to the combinatorial nature of the problem, the explicit RH MPC synthesis is usually only tractable for low state dimensions, say with 5 states or less. Computing u∗ on-line then reduces to a mere function evaluation, which can be performed very quickly even with limited computational resources. The main issue here is that the number of regions of κ(x), which is problem-dependent, tends to be large, easily exceeding the storage capacity of a selected implementation platform. Therefore it is important to keep the number of regions as low as possible, especially when considering typical industrial control platforms like programmable logic controllers or embedded microcontrollers. Such devices usually only provide few kilobytes of memory, while the memory footprint of typical explicit RH MPC solutions can easily exceed several megabytes.

One approach is to construct a sub-optimal replacement function ű(x) ≈ κ(x) of substantially lower complexity, see e.g. Bemporad and Filippi (2003); Johansen and Grancharova (2003); Cychowski and O’Mahony (2005); Ulbig et al. (2007); Jones and Morari (2009); Scibilia et al. (2009). Another line of research is concerned with finding the replacement ű(x), which is simpler than the original function, but maintains the equivalence κ(x) = ű(x) for all points x of interest. In Geyer et al. (2008) regions are merged if they share the same expression for the control law. If the PWA function κ(x) is convex (or if there exists a convex function q(x), defined over the same regions), then the method of Baotic et al. (2008) can be used to reduce the required memory storage. If κ(x) is non-convex, but continuous, its lattice representation (Wen et al., 2009) can be built, again decreasing the memory consumption.

In our previous work (Kvasnica and Fikar, 2010), the performance-lossless replacement ű(x) was constructed by only considering the regions of κ(x) where the control action is not saturated. This can considerably reduce the complexity as the number of unsaturated regions is usually significantly smaller compared to the total number of underlying polytopes over which κ(x) is defined. In this paper we improve our previous method and propose to use separation functions. At first, we divide the regions of κ(x) into three categories: unsaturated regions R UNSAT where umin < κ(x) < umax, regions R MAX where κ(x) = umax, and regions R MIN where κ(x) = umin. We then search for a function p(x) which separates the sets R MAX and R MIN. When found, the on-line implementation of u∗ = κ(x) can be substantially simplified by only requiring the storage of unsaturated regions. If x ∈ R MAX for a given x, the function p(x) is evaluated, and its sign then governs whether u∗ = umax or u∗ = umin. The problem then becomes to find a simple separator p(x), such that it is easy to evaluate on-line and requires small amount of memory for its storage. In this note we propose to opt for a polynomial type of p(x). The challenge of finding p(x) stems from the fact that the sets to be separated are in general non-convex. We show how to solve such a separation problem by solving linear optimization problems. In the case of polynomial separation, additional certification is needed, which can be implemented by finding the roots of a given polynomial. Existence of the separator p(x) then guarantees that the replacement feedback ű(x) will always consist of the unsaturated regions of κ(x) only. Such a direct guarantee cannot be

Corresponding author, e-mail: michal.kvasnica@stuba.sk
given for the clipping-based method of Kvasnica and Fikar (2010). A large case study is provided to confirm viability of the approach.

2. DEFINITIONS

A finite set of n elements $\mathcal{I} := \{I_1, \ldots, I_n\}$ will be denoted as $|\mathcal{I}| = n$ and its cardinality by $|\mathcal{I}|$. A polytope is the bounded convex intersection of c closed affine half-spaces, i.e. $\mathcal{R} := \{x \in \mathbb{R}^n_{c} \mid Fx \leq g\}$. We call the collection of polytopes $\{\mathcal{R}_i\}_{i=1}^N$ the partition of a polytope $\mathcal{R}$ if $\mathcal{R} = \bigcup_{i=1}^N \mathcal{R}_i$, and $\mathrm{int}(\mathcal{R}_i) \cap \mathrm{int}(\mathcal{R}_j) = \emptyset$ for all $i \neq j$. Each polytope $\mathcal{R}_i$ will be referred to as the region of the partition. Function $\kappa(x) : \mathbb{R}^n \mapsto \mathbb{R}^n_{c}$ with $x \in \mathcal{R} \subset \mathbb{R}^n_{c}$, $\mathcal{R}$ being a polytope, is called piecewise affine over polytopes if $\{\mathcal{R}_i\}_{i=1}^N$ is the partition of $\mathcal{R}$ and

$$\kappa(x) = K_i x + L_i \quad \forall x \in \mathcal{R}_i,$$

with $K_i \in \mathbb{R}^{n \times n_{c}}$, $L_i \in \mathbb{R}^{n}$, and $i = 1, \ldots, R$. PWA function $\kappa(x)$ is continuous if $K_i x + L_i = K_j x + L_j$ holds $\forall x \in \mathcal{R}_i \cap \mathcal{R}_j$, $i \neq j$.

3. EXPLICIT MODEL PREDICTIVE CONTROL

We consider the class of discrete-time, stabilizable linear time-invariant systems

$$x_{k+1} = Ax_k + Bu_k,$$

which are subject to polytopic constraints $x \in \mathcal{X} \subset \mathbb{R}^{n_{c}}$ and $u \in \mathcal{U} \subset \mathbb{R}^{n_{u}}$. Assume the following constrained finite-time optimal control problem:

$$\min_{U_N} \sum_{k=0}^{N-1} x_{k+1}^T Q x_{k+1} + u_{k}^T T Q u_k$$

s.t. $x_{k+1} = Ax_k + Bu_k$, $x_k \in \mathcal{X}$, $u_k \in \mathcal{U}$. (3b) where $x_k$ and $u_k$ denote, respectively, state and input predictions over a finite horizon $N$, given the initial condition $x_0$. It is assumed that $Q_k = Q_k^T \geq 0$, $Q_u = Q_u^T > 0$ in (3a), i.e. that (3) is a strictly convex QP. The receding horizon MPC feedback then becomes $u^*(x_0) = [1 \ 0 \ \cdots \ 0]U_N^T$, where the optimal vector $U_N^T := [u_0^T, \ldots, u_{N-1}^T]$ can be found by solving (3) as a QP for a given value of the initial condition $x_0$. For problems of modest size (typically for $n_x < 5$), it is also possible to characterize the optimal feedback $u^*(x_0)$ explicitly as a PWA function of $x_0$ (Bemporad et al., 2002) by solving (3) as a parametric quadratic program (pQP).

Theorem 3.1. (Bemporad et al. (2002)). The RH MPC feedback $u^*(x_0)$ for problem (3) is given by $u^*(x_0) = \kappa(x_0)$ where: (i) the set of feasible initial conditions $\Omega := \{x_0 \mid \exists 0 \leq k \leq N-1 \text{ s.t. } (3b) \text{ hold}\}$ is a polytope; (ii) $\kappa(x_0) : \Omega \mapsto \mathcal{U}$ is a continuous PWA function defined over $R$ regions $\mathcal{R}_i$, $i = 1, \ldots, R$; (iii) $\mathcal{R}_i$ are full-dimensional polytopes $\mathcal{R}_i = \{x \mid F_i x \leq g_i\}$; and (iv) $\{\mathcal{R}_i\}_{i=1}^R$ is a partition of $\Omega$.

The advantage of such an explicit representation is obvious; obtaining the optimal control action for a given $x_0$ reduces to a mere evaluation of the function $\kappa(x_0)$, which is henceforth denoted as the explicit RH MPC feedback law.

The crucial limitation, however, is that the number of regions tends to be large, often above the limits of typical control hardware implementation platforms.

In the next section we show how to replace the feedback law $u^*(x_0) = \kappa(x_0)$ by a different function $\tilde{\kappa}(x_0)$ which requires significantly less memory for its implementation in real-time arrangement and maintains the equivalence $\tilde{\kappa}(x_0) \equiv \kappa(x_0)$ $\forall x \in \Omega$. The procedure is applicable to generic PWA functions $\kappa(x)$ as long as they are continuous and all their regions $\mathcal{R}_i$ are full-dimensional polytopes. The scope of this work therefore extends to cases where 1- or $\infty$-norms are used in (3a), or when tracking of a non-zero reference is achieved by a suitable augmentation of the state vector.

4. COMPLEXITY REDUCTION VIA SEPARATION

By Theorem 3.1 we have that $\kappa(x)$ is a continuous PWA function defined over convex regions $\mathcal{R}_i$, union of which is the convex polytope $\Omega$. Denote by $\overline{\pi}$ and $\underline{\kappa}$ the maximal and minimal values which $\kappa(x)$ attains over its domain $\Omega$

$$\overline{\pi} = \max \{K_i x + L_i \mid x \in \mathcal{R}_i\}, \quad i = 1, \ldots, R \quad (4a)$$

$$\underline{\kappa} = \min \{K_i x + L_i \mid x \in \mathcal{R}_i\}, \quad i = 1, \ldots, R \quad (4b)$$

with $\overline{\pi} = \max(\overline{\pi}_1, \ldots, \overline{\pi}_R)$, $\underline{\kappa} = \min(\underline{\kappa}_1, \ldots, \underline{\kappa}_R)$. Then the regions of $\kappa(x)$ can be classified as follows.

(1) If $K_i = 0$ and $L_i = \overline{\pi}$, then region $\mathcal{R}_i$ is saturated at the maximum.

(2) If $K_i = 0$ and $L_i = \underline{\kappa}$, then region $\mathcal{R}_i$ is saturated at the minimum.

(3) otherwise the $i$-th region is unsaturated.

Denote by $I_{\max}$ and $I_{\min}$ the index lists of regions saturated at the maximum and minimum, respectively, and by $I_{\text{unsat}}$ the index list of unsaturated regions. With this classification, the RH MPC feedback $\kappa(x)$ can be written as

$$\kappa(x) = \begin{cases} K_i x + L_i & \text{if } x \in \mathcal{R}_{I_{\text{max}}} \times \overline{\pi} \\ \underline{\kappa} & \text{if } x \in \mathcal{R}_{I_{\text{min}}} \times \underline{\kappa}. \end{cases}$$

Evaluation of $\kappa(x)$ for any $x \in \Omega$ is therefore a two-stage process. First, the index $r$ of region $\mathcal{R}_r$, which contains $x$ needs to be identified. Then, the function value of $\kappa(x)$ is either computed by $K_r x + L_r$ if $r \in I_{\text{max}}$, or $\kappa(x) = \overline{\pi}$ (or $\kappa(x) = \underline{\kappa}$) if $r \in I_{\text{max}} \setminus I_{\text{min}}$. Identification of the index $r$ can either be done by searching through all regions $\mathcal{R}_r$, $i = 1, \ldots, R$ sequentially, or by traversing a corresponding binary search tree (Tøndel et al., 2003). In either case, the required memory storage is proportional to the total number of regions $R$.

If the number of saturated regions is non-zero, a simpler representation of $\kappa(x)$ in fact can be obtained. Notice that, since the regions $\mathcal{R}_r$ are non-overlapping due to Theorem 3.1, for any $x \in \Omega$, $x \notin I_{\text{unsat}}$, $\kappa(x)$ can only take two possible values; either $\kappa(x) = \overline{\pi}$, or $\kappa(x) = \underline{\kappa}$. This fact can be exploited to derive a new PWA function $\widetilde{\kappa}(x)$ which maintains the equivalence $\widetilde{\kappa}(x) = \kappa(x)$ for all $x \in \Omega$, and requires less memory for its description compared to the memory footprint of $\kappa(x)$.

Proposition 4.1. Let a function $\nu(x) : \mathbb{R}_{c} \mapsto \mathbb{R}$ which satisfies $\nu(x) > 0$ for all $x \in \mathcal{R}_{I_{\text{max}}}$ and $\nu(x) < 0$ for all $x \in \mathcal{R}_{I_{\text{min}}}$ be given. Define

$$\nu(x) = \begin{cases} \overline{\pi} & \text{if } x \in \mathcal{R}_{I_{\text{max}}} \times \overline{\pi} \\ \underline{\kappa} & \text{if } x \in \mathcal{R}_{I_{\text{min}}} \times \underline{\kappa}. \end{cases}$$

5384
Then, for all \( x \in \Omega \), \( \tilde{\kappa}(x) = \kappa(x) \).

**Proof.** Follows directly from (5) and the definition of \( \kappa(x) \).

Given \( p(x), u^* = \kappa(x) \) can be evaluated by only looking at the unsaturated regions \( R_{\text{unsat}} \). If \( x \in R_i, \ r \in I_{\text{unsat}}, \) then \( u^* = K_i x + L_i \). Otherwise, based on the sign of \( p(x) \), one either takes \( u^* = \pi \) or \( u^* = \kappa \).

If \( \kappa(x) \) is a continuous PWA function, then a possibly discontinuous separating function \( p(x) \) always exists. Under continuity, the convex regions \( R_i \) and \( R_k \) cannot be adjacent for any \( j \in I_{\text{max}}, \ k \in I_{\text{min}}, \) and therefore they can always be separated. As will be evidenced later, a typical explicit RHMPMC feedback laws \( \kappa(x) \) contains a significantly smaller number of unsaturated regions as compared to the number of saturated ones, i.e., \( |I_{\text{unsat}}| \ll |I_{\text{max}}| + |I_{\text{min}}| \). Therefore \( \tilde{\kappa}(x) \) will require significantly less memory than \( \kappa(x) \), and will be faster to evaluate too, if \( p(x) \) is a “simple” separator of the two sets \( R_{\text{max}} \) and \( R_{\text{min}} \). Various types of \( p(x) \) can be considered, either continuous (e.g. linear or polynomial), or discontinuous (e.g. piecewise linear or piecewise polynomial). In this work we have opted for the polynomial type of \( p(x) \) and the problem which we aim at solving is formally stated as follows.

**Problem 4.2.** Given a RHMPMC feedback law \( u^* = \kappa(x) \) with \( \kappa(x) \) as in (5), construct the replacement function (6) by finding the multivariate polynomial

\[
p(x) := \sum_{i_1 + \cdots + i_n \leq \delta} \alpha_{i_1, \ldots, i_n} x_1^{i_1} \cdots x_n^{i_n},
\]

of minimum degree \( \delta_{\text{min}} \) such that \( p(x) \) strictly separates the sets of regions \( R_{\text{max}} \) and \( R_{\text{min}} \), i.e. \( p(x) > 0 \) \( \forall x \in R_{\text{max}} \) and \( p(x) < 0 \) \( \forall x \in R_{\text{min}} \).

Solving Problem 4.2 is, however, nontrivial, since the unions of polytopes, i.e. \( R_{\text{max}} = \{ x \mid F_i x \leq g_i \} \) and \( R_{\text{min}} = \{ x \mid x \in \bigcup R_i, \ i \in I_{\text{min}} \} \), can be non-convex, in general. Even deciding whether they are convex or not is hard (Bemporad et al., 2001).

### 4.1 Polynomial separation

Given are the (non-convex) sets \( R_{\text{max}} \) and \( R_{\text{min}} \), each of which consists of a finite number of polytopes \( R_k \). Denote by \( V_k \) the vertices of \( R_k \) and fix some integer \( \delta \geq 1 \) in (7). Then the necessary condition for the existence of a polynomial \( p(x) \) which strictly separates \( R_{\text{max}} \) and \( R_{\text{min}} \) is that the following optimization problem is feasible:

\[
\epsilon^* = \max_{\epsilon, v_i} \epsilon \quad \text{s.t.} \quad p(v_i) \geq \epsilon, \quad \forall v_i \in V_{\text{max}}, \quad \text{(8a)}
\]

\[
p(v_j) \leq -\epsilon, \quad \forall v_j \in V_{\text{min}}, \quad \text{(8b)}
\]

\[
\epsilon \geq 0. \quad \text{(8c)}
\]

The optimal value \( \epsilon^* \) then denotes the maximal separation gap between the two sets of points \( V_{\text{max}} \) and \( V_{\text{min}} \). Important to notice is that (8) is a linear program (LP),

since, for some fixed argument \( x = v_k, \ v_k \in V_k, \ p(x) \) in (8b)–(8c) are linear functions of the coefficients \( \alpha_{i, j} \). If the LP (8) is infeasible, then no polynomial separator \( p(x) \) of the form of (7) exists for a given degree \( \delta \).

If \( \delta = 1 \) in (8) then having \( \epsilon^* > 0 \) is also sufficient for the linear function \( p(x) := \alpha_0 + \alpha_1 x \) to strictly separate the sets \( R_{\text{max}} \) and \( R_{\text{min}} \) (Boyd and Vandenberghe, 2004). Consider therefore \( \delta > 1 \). If (8) is feasible with \( \epsilon^* > 0 \), then one of the two possible scenarios can occur. In an ideal case, solving for \( p(x) \) from (8) by only considering separation of \( V_{\text{max}} \) and \( V_{\text{min}} \) will also provide a separator for the sets \( R_{\text{max}} \) and \( R_{\text{min}} \), as shown in Fig. 1(a). In a more general case, though, strict separation of vertices is not sufficient for \( p(x) \) to separate all points from the associated sets, cf. Fig 1(b).

An additional certification step therefore has to be performed. At this point we remind that all regions of \( R_{\text{max}} \) and \( R_{\text{min}} \) are polytopes described by \( R_k = \{ x \mid F_i x \leq g_i \} \). Consider the \( k \)-th facet of \( R_k \), i.e. \( \{ x \mid f_{i,k} x - g_{i,k} = 0 \} \) where \( f_{i,k} \) and \( g_{i,k} \) are the \( k \)-th rows of the respective matrices \( F_i \) and \( g_i \). Denote by \( \tilde{x}_{i,k} \) all (or some) solutions to the polynomial equation \( p(x) = f_{i,k} x - g_{i,k} \) on domain \( x \in R_k \):

\[
\tilde{x}_{i,k} = \{ x \mid p(x) - f_{i,k} x + g_{i,k} = 0, x \in R_k \}. \quad \text{(9)}
\]

Clearly, if \( \tilde{x}_{i,k} = \emptyset \) \( \forall i \in I_{\text{max}} \cup I_{\text{min}} \) and \( \forall k, \) then \( p(x) \) as a solution to (8) strictly separates \( R_{\text{max}} \) and \( R_{\text{min}} \) (cf. Figure 1(a)). On the other hand, the situation in Figure 1(b) corresponds to the case where there exist some points \( \tilde{x}_{i,k} \), for which the polynomial \( p(x) \) intersects the \( k \)-th facet of the \( i \)-th region, i.e. when \( \tilde{x}_{i,k} \neq \emptyset \) for some \( i \) and \( k \). In such a case, the existence of any such point \( \tilde{x}_{i,k} \) provides a certificate that \( p(x) \) does not separate \( R_{\text{max}} \) from \( R_{\text{min}} \).

When at least one offending point \( \tilde{x}_{i,k} \) exists, it can be added to the corresponding set of vertices in (8b)–(8c). I.e., if \( \tilde{x}_{i,k} \neq \emptyset \) for some \( i \in I_{\text{max}} \), then \( V_{\text{max}} = V_{\text{max}} \cup \tilde{x}_{i,k} \). Otherwise, if \( i \in I_{\text{min}} \) then \( V_{\text{min}} = V_{\text{min}} \cup \tilde{x}_{i,k} \). Resolving the LP (8) with the updated list of vertices will then give a new polynomial \( p(x) \) for which the certification is repeated, cf. Figure 1(c). If more offenders are found, they are added to the list of vertices and the procedure is repeated. Otherwise, an empty solution to (9) provides a certificate that \( p(x) \) strictly separates \( R_{\text{max}} \) from \( R_{\text{min}} \), whereupon the procedure terminates. The discussed mechanism can be formally stated as Algorithm 1, reported next.

**Remark 4.3.** Vertex enumeration in Step 1 of Algorithm 1 is considered a hard problem in general. However, for
Algorithm 1 Construction of a polynomial separator \( p(x) \)

**INPUT:** Sets \( R_{\text{I} \text{max}} \) and \( R_{\text{I} \text{min}} \), polynomial degree \( \delta \).

**OUTPUT:** Separating polynomial \( p(x) \) as in (7).

1: Get the lists of vertices \( V_{\text{max}} \) and \( V_{\text{min}} \).

2: repeat

3: Solve the LP (8) and obtain coefficients \( \alpha_i \).

4: if \( \alpha_\ast > 0 \) then

5: Compute the list of offending points \( x \) from (9).

6: Insert \( x \) to \( V_{\text{max}} \) or \( V_{\text{min}} \).

7: else

8: No strict separator \( p(x) \) of degree \( \delta \) exists, abort.

9: end if

10: until \( \tilde{x} \neq \emptyset \).

the type of small-dimensional problems considered here, enumerating \( V \) does not pose any significant technical difficulty and the vertices can be easily computed e.g. by CDD (Fukuda, 1997) in a matter of seconds.

**Remark 4.4.** There is no theoretical guarantee that the iterations between Steps 2–10 will terminate in finite time. However, for more than 400 random problems reported in Section 5.2, the number of iterations never exceeded 4.

**Remark 4.5.** The list of offending points \( x \) in Step 5 can be obtained by solving (9) in several ways. One option is to compute the real roots of the polynomial \( p(x) = f_{\ast, k} + g_{\ast, k} = 0 \) numerically e.g. by using the package of Zeng (2004). Since such a method does not allow to restrict the offenders to a particular domain, the roots which violate \( \tilde{x}_{\ast, k} \in \mathcal{R}_i \) need to be excluded. Another option is to consider (9) as a feasibility problem with a nonlinear constraint. Nonlinear optimization routines, such as fmincon of MATLAB, can then be used to find at least one such offender for each region \( \mathcal{R}_i \), provided it exists.

Solving Problem 4.2 involves finding a strict separator \( p(x) \) of the minimum degree \( \delta_{\text{min}} \). This can be achieved e.g. by using bisection, i.e. by running Algorithm 1 multiple times for various values of \( \delta \) until a feasible solution is obtained and \( \delta \) is minimized.

### 4.2 Multi-input case

So far we have considered replacing the RHMPC feedback law \( \kappa(x) \) by a different function \( \tilde{\kappa}(x) \) of the form of (6), which consists of the unsaturated regions of \( \mathfrak{k}(x) \) and the separator \( p(x) \). If \( \kappa(x) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u} \) is such that \( n_u > 1 \) in (2), then one can proceed by decomposing \( \kappa(x) \) into individual PWA functions, i.e. \( \kappa_i(x) := k_{i,j}x + l_{i,j} \) if \( x \in \mathcal{R}_i \), where \( k_{i,j}, l_{i,j} \) are the \( j \)-th rows of \( K_i \) and \( L_i \), respectively. Then a set of \( j = 1, \ldots, n_u \) polynomial separators \( p_{\ast,j}(x) \) can then be obtained by running Algorithm 1 \( n_u \) times for different polytopic sets \( \mathcal{R}_{\text{I} \text{max},j} \) and \( \mathcal{R}_{\text{I} \text{min},j} \). Here, the index sets \( \mathcal{I}_{\text{max},j} \) and \( \mathcal{I}_{\text{min},j} \) are obtained based on the scalarized version of (4), i.e.

\[
\mathcal{P}_{\ast,j} = \max \{ k_{i,j}x + l_{i,j} | x \in \mathcal{R}_i \}, \quad i = 1, \ldots, R, \quad (10a)
\]

\[
\mathcal{E}_{\ast,j} = \min \{ k_{i,j}x + l_{i,j} | x \in \mathcal{R}_i \}, \quad i = 1, \ldots, R, \quad (10b)
\]

with \( \mathcal{F}_{\ast,j} = \max \{ \mathcal{P}_{\ast,j} \}_{j=1}^R, \quad \mathcal{E}_{\ast,j} = \min \{ \mathcal{E}_{\ast,j} \}_{j=1}^R \). Naturally, different index sets of unsaturated regions, i.e. \( \mathcal{I}_{\text{unsat},j} \), will be obtained for different values of \( j \). Even though the total number of regions of \( \tilde{\kappa}(x) \) is then \( \sum_{j=1}^{R} \mathcal{I}_{\text{unsat},j} \), significant reduction of complexity can still be achieved if \( |\mathcal{I}_{\text{unsat},j}| \ll R \) for all \( j \in \{ 1, \ldots, n_u \} \).

### 4.3 Complexity analysis

Evaluation of \( \tilde{\kappa}(x) \) as in (6) for a given value of the vector \( x \) first requires to assess whether \( x \in \mathcal{R}_{\text{I} \text{max}} \). Searching through the regions \( \mathcal{R}_{\text{I} \text{max}} \) sequentially can answer this query in \( O(|\mathcal{I}_{\text{unsat}}|) \) time, while the binary search tree approach of Tøndel et al. (2003) can provide the answer in \( O(\log_2 |\mathcal{I}_{\text{unsat}}|) \) time. Both approaches require the storage of the unsaturated regions, hence their memory footprint is \( O(|\mathcal{I}_{\text{unsat}}|) \). If \( x \in \mathcal{R}_{\text{I} \text{max}} \), the index \( x \) of region \( \mathcal{R}_r \) is returned, whereupon the value of \( \tilde{\kappa}(x) \) is given by \( K_r x + L_r \). If \( x \notin \mathcal{R}_{\text{I} \text{max}} \), then the value of the separator \( p(x) \) is obtained and its sign is used in (6). The memory and computation requirements associated with storing and evaluating \( p(x) \) online is insignificant compared to the description of regions \( \mathcal{R}_{\text{I} \text{max}} \).

Other approaches can be used to derive the replacement function \( \tilde{\kappa}(x) \). The lattice representation (LR) of Wen et al. (2009) converts the original function \( \kappa(x) \) into a series of \( \min/\max \) operations over the functions \( K_i x + L_i \), eliminating the need to store the underlying regions \( \mathcal{R}_i \). Evaluation of such a lattice description requires \( \mathcal{O}(R_i \text{unique}) \) operations, where \( R_i \text{unique} \) is the number of regions where the feedback law is unique. The memory storage is also proportional to \( \mathcal{O}(R_i \text{unique}) \), however the constant term in the big-O formulation is small due to the fact that only the matrices \( K_i \) and \( L_i \) need to be stored. The clipping-based procedure (Kvasnica and Fixar, 2010) removes all saturated regions and replaces them by “extensions” of the unsaturated ones. In the best case, \( \tilde{\kappa}(x) \) is then defined over \( |\mathcal{I}_{\text{unsat}}| \) regions, while in the worst case the number of regions remains unchanged. On average, \( \tilde{\kappa}(x) \) consists of \( 1.3|\mathcal{I}_{\text{unsat}}| \) regions. The memory and runtime requirements of such a scheme are proportional to this figure.

### 5. EXAMPLES

#### 5.1 Illustrative example

Consider a 2-state 1-input system given by

\[
x^+ = \begin{bmatrix} 0.755 & 0.680 \\ 0.651 & -0.902 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0.825 \\ -0.139 \end{bmatrix} u,
\]

which is subject to constraints \( \mathcal{X} = \{ [x_1^1, x_1^2] \mid -10 \leq x_1^2 \leq 10 \} \) and \( \mathcal{U} = \{ u \in \mathbb{R} \mid -1 \leq u \leq 1 \} \). The MPC problem (3) was formulated with prediction horizon \( N = 10 \), \( Q_x = 1 \) and \( Q_u = 1 \) and solved as a parametric QP according to Theorem 3.1. Using the MPT Toolbox (Kvasnica et al., 2004), the explicit RHMPC feedback \( \kappa(x) \) was obtained in 4 seconds\(^2\) as a PWA function defined over 225 regions shown in Fig. 2. The partition of \( \kappa(x) \) consists of 29 unsaturated regions, 98 regions where \( \kappa(x) = 1 \), and 98 regions where \( \kappa(x) \) is saturated at \( -1 \).

As can be clearly see from the shape of the sets in Figure 2, no linear separation between \( \mathcal{R}_{\text{I} \text{max}} \) and \( \mathcal{R}_{\text{I} \text{min}} \) can be found. A polynomial separator \( p(x) = -x_1 - x_2 - 0.0011x_1^2 - 0.254x_2^2 \) of the minimal degree \( \delta_{\text{min}} = 2 \) on a 2.4 GHz CPU with 2GB of RAM using MATLAB 7.8 and MPT 2.6.3.
Step 1

3 was then found by applying bisection in conjunction with Algorithm 1. The algorithm converged within two iterations. The vertices in Step 1 were computed by CDD in 0.01 seconds. Coefficients of the polynomial were obtained by solving the LP (8), which only took 0.1 seconds using CPLEX. The subsequent certification check in Step 5 was implemented by solving (9) using fmincon, which took 1.1 seconds.

The total memory footprint of \( \kappa(x) \) (which consists of the regions \( R_i \) and the feedback laws \( K_i(x) + L_i \)) with 225 regions is 27 kilobytes. On the other hand, by devising the polynomial separator \( p(x) \), the storage requirements of \( \tilde{\kappa}(x) \) is a mere 3.5 kilobytes. Here, the unsaturated regions \( R_{\text{unsat}} \) contribute by 2.8 kB, the associated feedback laws by 0.7 kB, and the memory footprint is just 16 bytes. It follows that complexity of the on-line implementation of the RHMPC feedback law can be reduced by a factor of 7.7 when using the modified feedback \( \tilde{\kappa}(x) \) instead of the original function \( \kappa(x) \). Enhancing the prediction horizon to \( H = 15 \) leads \( \kappa(x) \) with 489 regions, 39 of which are unsaturated. For this larger case, the minimal degree of the separating polynomial \( p(x) \) is again \( \delta_{\text{min}} = 3 \) and its coefficients can be found by Algorithm 1 in 1.5 seconds. This time, the memory footprint of \( \kappa(x) \) is 58.2 kB, while \( \tilde{\kappa}(x) \) only occupies 4.8 kB, a reduction by factor of 12.

5.2 Large-scale analysis

Next, we have analyzed a large number of random RHMPC feedback laws \( \kappa(x) \) generated by solving problem (3) for randomly selected LTI systems with 2 to 3 states, and 1 to 2 inputs. 100 random cases were considered for each \( n_x/n_u \) category. For each PWA function \( \kappa(x) \) we have constructed the replacement \( \tilde{\kappa}(x) \) as in (6).

Purpose of such a large-scale analysis is to confirm the main two assertions behind this work. First, it verifies that the number of unsaturated regions is indeed considerably smaller compared to the number of saturated ones, i.e. that \( |I_{\text{unsat}}| \ll |I_{\text{max}}| + |I_{\text{min}}| \) often holds in practice. Second, it shows that low degrees of the polynomial separator \( p(x) \) are typically sufficient to obtain a strict separation. Moreover, the analysis also shows how Algorithm 1 scales with increasing problem size.

Based on the 400 random scenarios, Table 1 shows for how many cases a polynomial separator \( p(x) \) of a given minimal degree \( \delta_{\text{min}} \) could be found by Algorithm 1. Although only degrees \( \delta \leq 5 \) were considered due to practical reasons, the overall success was 99.5%. Important to notice is that a linear separator exists in a majority of cases. Such a \( p(x) \) can be found by solving the LP (8) without the need to further certify the separation in Step 5. We remark that for all cases where \( \delta = 1 \) was sufficient, it never took more than 10 seconds to compute \( p(x) \) by Algorithm 1, regardless of problem size, as reported in Table 2. CPLEX was used to solve the LPs (8).

Instances where higher-order polynomials \( p(x) \) were necessary are further elaborated in Table 3, which also shows how the computation scales with increasing number of regions. In addition, the reported results confirm the conclusions of Remark 4.4, i.e. that the number of iterations in Algorithm 1 is minor in practice. Majority of the runtime of Alg. 1 consists of the time spent in Step 5, which was implemented using fmincon.

Finally, Table 4 reports the minimal, maximal, and average values of the achievable complexity reduction ratio, which is defined as ratio between the total number of regions of \( \kappa(x) \) to the number of unsaturated regions, i.e. \( \Delta = \frac{|I_{\text{unsat}}|}{|I_{\text{min}}|} \). The results show that the number of unsaturated regions is indeed significantly smaller in practice. Therefore, the replacement function \( \tilde{\kappa}(x) \) (6), which only requires the storage of unsaturated regions, will typically be considerably simpler compared to the original RHMPC feedback law \( \kappa(x) \). We remark that the additional memory due to the storage of \( p(x) \) usually amounts to less than 100 bytes.

6. CONCLUSIONS

Given an explicit RHMPC feedback function \( \kappa(x) \), we have shown how to construct its simpler replacement \( \tilde{\kappa}(x) \) which maintains the equivalence \( \kappa(x) = \tilde{\kappa}(x) \) for all \( x \in \text{dom} \kappa(x) \). The mechanism was based on devising a function \( p(x) \), which separates the regions over which \( \kappa(x) \) attains a saturated value. The replacement \( \tilde{\kappa}(x) \) then requires only the storage of the unsaturated regions of \( \kappa(x) \), along with the separator \( p(x) \). We have shown how to build a polynomial separator by solving linear optimization problems, followed by a certification step which requires solution to a polynomial equation. By
Table 3. Complexity of construction of the separator $p(x)$ with $\delta_{\text{min}} > 1$.

<table>
<thead>
<tr>
<th>$n_x/n_u$</th>
<th>No. of regions</th>
<th>$\delta_{\text{min}}$</th>
<th>No. of iterations</th>
<th>Runtime of Alg. 1 [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/1</td>
<td>225</td>
<td>3</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>283</td>
<td>3</td>
<td>1</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>493</td>
<td>3</td>
<td>1</td>
<td>1.7</td>
</tr>
<tr>
<td></td>
<td>495</td>
<td>3</td>
<td>1</td>
<td>1.3</td>
</tr>
<tr>
<td>2/2</td>
<td>297</td>
<td>3</td>
<td>1</td>
<td>3.2</td>
</tr>
<tr>
<td></td>
<td>541</td>
<td>3</td>
<td>2</td>
<td>17.3</td>
</tr>
<tr>
<td></td>
<td>787</td>
<td>5</td>
<td>1</td>
<td>9.2</td>
</tr>
<tr>
<td></td>
<td>949</td>
<td>3</td>
<td>1</td>
<td>20.1</td>
</tr>
<tr>
<td>3/1</td>
<td>384</td>
<td>3</td>
<td>2</td>
<td>22.1</td>
</tr>
<tr>
<td></td>
<td>527</td>
<td>3</td>
<td>4</td>
<td>20.6</td>
</tr>
<tr>
<td></td>
<td>1275</td>
<td>3</td>
<td>1</td>
<td>14.4</td>
</tr>
<tr>
<td></td>
<td>2513</td>
<td>3</td>
<td>1</td>
<td>35.8</td>
</tr>
<tr>
<td>3/2</td>
<td>191</td>
<td>3</td>
<td>3</td>
<td>7.5</td>
</tr>
<tr>
<td></td>
<td>449</td>
<td>3</td>
<td>2</td>
<td>13.5</td>
</tr>
<tr>
<td></td>
<td>1396</td>
<td>3</td>
<td>4</td>
<td>73.3</td>
</tr>
<tr>
<td></td>
<td>3953</td>
<td>3</td>
<td>1</td>
<td>40.8</td>
</tr>
</tbody>
</table>

Table 4. Minimal, average, and maximal values of the complexity reduction ratio.

<table>
<thead>
<tr>
<th>$n_x/n_u$</th>
<th>$\Delta_{\text{min}}$</th>
<th>$\Delta_{\text{avg}}$</th>
<th>$\Delta_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/1</td>
<td>2.3</td>
<td>13.0</td>
<td>31.0</td>
</tr>
<tr>
<td>3/1</td>
<td>2.1</td>
<td>7.1</td>
<td>21.0</td>
</tr>
<tr>
<td>2/2</td>
<td>1.8</td>
<td>5.9</td>
<td>14.5</td>
</tr>
<tr>
<td>3/2</td>
<td>1.9</td>
<td>3.6</td>
<td>10.2</td>
</tr>
</tbody>
</table>

means of a large case study we have demonstrated that the procedure scales well with increasing problem size, and that significant reduction of complexity can be achieved in general.

ACKNOWLEDGMENT

The authors are pleased to acknowledge the financial support of the Scientific Grant Agency of the Slovak Republic under the grants 1/0095/11. This work was supported by the Slovak Research and Development Agency under the contracts No. VV-0029-07 and No. LPP-0092-07.

REFERENCES


