This is an author produced version of a paper published in *Control Engineering Practice*.

White Rose Research Online URL for this paper: [http://eprints.whiterose.ac.uk/43194](http://eprints.whiterose.ac.uk/43194)

**Published paper**

[http://dx.doi.org/10.1016/j.conengprac.2011.04.001](http://dx.doi.org/10.1016/j.conengprac.2011.04.001)
Efficient suboptimal parametric solutions to predictive control for PLC applications

G. Valencia-Palomo\textsuperscript{a,}\textsuperscript{*}, J.A. Rossiter\textsuperscript{a,1}

\textsuperscript{a}Department of Automatic Control and Systems Engineering, University of Sheffield, South Yorkshire, U.K. S1 3JD.

Abstract

The prime aim of this paper is to embed a predictive control (MPC) algorithm with constraint handling capabilities into a Programmable Logic Controller (PLC). In order to achieve it, this paper develops parametric approaches to MPC but differs from more conventional approaches in that it pre-defines the complexity of the solution rather than the allowable suboptimality. The paper proposes a novel parameterisation of the parametric regions which allows efficiency of definition, effective spanning of feasible region and also highly efficient search algorithms. Despite the suboptimality, the algorithm retains guaranteed stability, in the nominal case. A laboratory test was carried out to demonstrate the code on real hardware and the effectiveness of the solution.

Keywords: Predictive control, multi-parametric quadratic programming, PLC.

1. Introduction

Parametric solutions to predictive control (MPC) (Bemporad et al., 2002a,b; Pistikopoulos et al., 2002) have the key advantages of (i) giving transparency to the control law which may have advantages in safety critical or highly regulated environments and (ii) having the potential to significantly reduce the on line computational load/complexity. This paper concerns itself primarily with the second of these points because often the potential to reduce complexity is not realised; for instance, especially with high order systems, the optimal parametric solution may be very complex so that implementation is more difficult or slower than including an online quadratic programme (QP). The aim is to achieve a reduction of complexity and data storage to the point the controller can be used in a Programmable Logic Controller (PLC) as this hardware represents the standard computer in industry.

Several authors have tried to tackle this issue with various research directions. Some authors have looked at optimising the efficiency of storage of the parametric solution combined with fast algorithms for implementation of the solution (e.g. Borrelli et al., 2001; Tondel et al., 2003; Christophersen et al., 2007). Other authors have considered beginning from a suboptimal parametric solution in the hope that such a solution may be far simpler, but with a small loss in performance only (e.g. Johansen, 2003; Johansen et al., 2002; Johansen and Grancharova, 2003;...
Grieder et al., 2004; Bemporad and Filippi, 2003, 2006). One approach in the literature has looked at using orthogonal spaces to speed up search times and sub-divides the parametric space into small enough regions to quantify the sub-optimality as small enough, either from a performance perspective or in terms of feasibility (Johansen, 2003). Another alternative is to look at making the regions larger (Bemporad and Filippi, 2003) by allowing some relaxation of the optimality, while ensuring feasibility. However, critically for the motivation here, none of the works above are able to give any strict bounds on the resulting complexity of the solution which thus may still be worse than desired.

A less explored avenue is to base the parametric solution on points rather than regions (Canale et al., 2009, 2010a,b). In simple terms one predefines the optimum strategy for a number of possible initial conditions and then online select from these the one which is closest to the actual initial condition. However, once again the main failing of this approach is that it is hard to get bounds on the complexity of the solution because the focus of the work is on ensuring that the suboptimality meets some guaranteed requirement which thus can be conservative. The examples in those papers use numbers such as $10^4 - 10^6$ vertices. An even less explored avenue is the potential to use interpolation (Rossiter and Grieder, 2005) to give a convex blend from nearby points, thus reducing the number of points/regions required while ensuring feasibility.

Here the intention is to take a different viewpoint from those which either start from the explicit optimal and define efficient searches or look for ways of trading complexity with performance. Instead this paper proposes to predefine the complexity of the solution and then ask whether one is able to get sufficient performance and guarantees of feasibility. The argument taken is that any result which is based on sub-division until the difference from the optimal is small will, in general, lead to a large number of regions. In general, a convergence and feasibility guarantee with a pre-defined complexity are wanted; and then to ask what level of performance it can get from that, accepting that it will be suboptimal in comparison to a more complex solution. The advantage of predefining the complexity is the possibility, a priori, of giving strict limits on data storage and sample time requirements and thus being much cheaper and simpler to implement on systems which have available only low computational power like a PLC but may still require fast sample times (Valencia-Palomo and Rossiter, 2010).

Hence the key contribution of this paper is a proposed approach which reduces complexity of the parametric solution and does not give large growth in complexity with the state dimension. Moreover, the online coding requirements are trivial as compared to the online implementation of a QP solver. In line with some concepts adopted by earlier authors (Johansen and Grancharova, 2003; Grieder et al., 2004; Bemporad and Filippi, 2006) this paper intends to make use of regular shapes as this enables very efficient search methods and simple polytope definitions with predefined complexity.

The paper is organised as follows: Section 2 will give a brief background on a standard MPC algorithm and explicit solutions; Section 3 discusses about complexity of polytope representations; Section 4 then introduces the proposed suboptimal parametric solution with proofs of feasibility and convergence; Section 5 presents some Monte-Carlo numerical illustrations; Section 6 presents the hardware overview and implementation of the algorithm; Section 7 presents an experimental example; the paper finishes with the conclusions in Section 8.
2. Background on predictive control

2.1. Model and constraints

This paper assumes a standard state space model of the form

$$x_{k+1} = Ax_k + Bu_k; \quad y_k = Cx_k;$$

(1)

where $x_k \in \mathbb{R}^{n_x}$, $u_k \in \mathbb{R}^{n_u}$, $y_k \in \mathbb{R}^{n_y}$ are the states, inputs and outputs at sample $k$ respectively. It is assumed that these are subject to polytopic constraints at every sample instant, for example:

$$A_u u_k \preceq b_u; \quad A_\Delta u_\Delta u_k \preceq b_\Delta u; \quad A_y y_k \preceq b_y;$$

(2)

where $\Delta u_k = u_k - u_{k-1}$.

In the context of predictive control, it is common to take the following quadratic performance index as the objective to be minimised at each sample

$$J = \sum_{i=1}^{\infty} \{ x_{k+i}^T Q x_{k+i} + u_{k+i-1}^T R u_{k+i-1} \},$$

(3)

with $Q \in \mathbb{R}^{n_x \times n_x}$, $R \in \mathbb{R}^{n_u \times n_u}$ positive definite state and input cost weighting matrices; as, under some mild conditions, this allows a straightforward stability guarantee in the nominal case. In the unconstrained case, the control law is given as $u_k = -K x_k$, where the optimal feedback gain $K$ is obtained via the solution of the corresponding Linear Quadratic Regulator (LQR) problem. However, it is noted here that this paper omits the fine details associated to integral action and offset free tracking to simplify the presentation.

2.2. Optimal MPC (OMPC)

The key idea in Scokaert and Rawlings (1998); Rossiter et al. (1998), is to embed into the predictions the unconstrained optimal behaviour and handle constraints using perturbations about this. Assuming that $K$ is the optimal feedback, the input predictions are defined as follows:

$$u_{k+i} = \begin{cases} -K x_{k+i} + c_{k+i} & i = 0, \ldots, n_c - 1 \\ -K x_{k+i} & i \geq n_c \end{cases}.$$  

(4)

where $n_c$ is the prediction horizon for the perturbations $c_k$. It is known that, the input predictions (4) and associated state predictions for model (1) satisfy constraints (2) if

$$M x_k + N c_k \leq 1; \quad c_k = [c_k^T, c_{k+1}^T, \ldots, c_{k+n_c-1}^T]^T;$$

(5)

for suitable $M$, $N$ with a finite number of rows (Gilbert and Tan, 1991). Details of how to compute these matrices are omitted as by know well known in the literature (Gilbert and Tan, 1991; Scokaert and Rawlings, 1998; Rossiter, 2003).

Definition 2.1 (MAS). Define the Maximum Admissible Set (MAS) as the region in the state space for which the unconstrained feedback $u_k = -K x_k$ satisfies constraints, i.e. \{ $x_k \in \mathbb{R}^{n_x} | M x_k \leq 1$ \}.
Definition 2.2 (MCAS). Define the Maximum Controllable Admissible Set (MCAS) as the region in the state space whereby it is possible to find a $\mathbf{c}_k$ such that the future trajectories satisfy constraints; i.e. \( \{ x_k \in \mathbb{R}^{n} | \exists \mathbf{c}_k \in \mathbb{R}^{n_{u}} \text{ s.t. } Mx_k + N\mathbf{c}_k \leq 1 \} \)

The MCAS gets larger as $n_c$ increases, but usually up to a finite limit if the state constraints give a closed region (Gilbert and Tan, 1991). It is not the purpose of this paper to consider nuances in that discussion topic. $n_c$ is taken to be large here and in fact it is known from work in parametric solutions that often a relatively small finite value is enough to capture the maximum MCAS.

Algorithm 2.1 (OMPC). The OMPC algorithm is

\[
\mathbf{c}_k^* = \arg\min_{\mathbf{c}_k} \mathbf{c}_k^T S \mathbf{c}_k \text{ s.t. } (5)
\]

Use the first element of $\mathbf{c}_k^*$ in the control law of (4), with $K$. This algorithm will find the global optimal, with respect to (3), whenever that is feasible and has guaranteed convergence/recursive feasibility in the nominal case.

Definition 2.3. For initial states $x_k = v_j$, define the corresponding optimal control sequences as $\mathbf{c}_k = \mathbf{c}_{j,k}$. By definition therefore the recursive use of the this sequence of $\mathbf{c}_{k+1}$ values in (4) will give input/state trajectories that satisfy constraints and converge to the origin.

Definition 2.3 explicitly associates the optimal trajectory $\mathbf{c}_{j,k}$ with the initial condition $v_j$ this will be useful in the next sections where the vertices ($v_j$'s) of the feasible regions are important elements of the proposed algorithm.

2.3. Parametric solutions

The solution of problem (6) has a parametric solution (mp-QP) (Bemporad et al., 2002b) of the form

\[
x_k \in \mathcal{R}_r \Rightarrow \mathbf{c}_k^* = -K_r x_k + t_r; \quad \mathcal{R}_r = \{ x_k : M_r x_k \leq d_r \};
\]

for suitable $K_r$, $t_r$, $M_r$, $d_r$ where the interiors of the polytopes $\mathcal{R}_r$ do not overlap and the union gives the MCAS. The main weakness of parametric solutions is that the number of regions $r$ can grow very quickly both with state dimension and indeed $n_c$. This paper seeks alternative but suboptimal parametric solutions which require far fewer regions.

3. Facet or vertex representations of polytopes and parametric solutions

The main assumption of this paper is that an efficient parametric realisation requires an assumption of regular polytopes, such as $n_c$-dimensional cubes, as opposed to the more general shapes possible for $\mathcal{R}_r$. This is because such an assumption carries several simple benefits: (i) the
number of facets may be small; (ii) the number of vertices may be small and equispaced to some extent; (iii) the shape, facets and vertices are regular and hence easy to handle and define. This allows for very efficient search algorithms. In fact, the assumption in this paper is slightly different from a cube, although the proposed algorithm is seeded by a cube.

This section will first present a discussion about the complexity of facet and vertex based representations of polytopes (Section 3.1 and 3.2). It then goes to present convexity arguments for MPC (Section 3.3), the definition of the particular shape used (Section 3.4) and the algorithm to solve the point location problem (Section 3.5). The section finalises with a brief summary of the results (Section 3.6).

3.1. Facet based parametric solutions

A major concern is related to the efficiency of search algorithms. It is recognised (Borrelli et al., 2001; Rossiter and Grieder, 2005) that one can define very efficient algorithms for finding an active facet [Definition 3.1] and thus an MPC algorithm which utilises this mechanism (there is an explicit link between the active facet and the control law) has the potential to be very efficient; it has also been shown that efficient mp-QP parameterisations, can use active facet computations to infer the set membership computations in (7).

**Definition 3.1.** Consider a closed polytope, containing the origin, given as \( \mathcal{R} = \{ \mathbf{x}_k : \mathbf{Rx}_k \leq 1 \} \), \( \mathbf{R} \in \mathbb{R}^{q \times n_x} \) with \( q \) (non-redundant) inequalities and facets \( \{ \mathcal{R}_1^f, \ldots, \mathcal{R}_q^f \} \). Then for any state \( \mathbf{x}_k \), the active facet is \( \mathcal{R}_a^f \) with

\[
    a = \arg \min_{j \in \{1, \ldots, q\}} e_j^T \mathbf{Rx}_k,
\]

where \( e_j \) is the \( j \)th standard basis vector.

Thus, the search algorithm efficiency depends only on \( q \), the number of inequalities defining \( \mathcal{R} \). However, this is where the goal of a highly efficient search algorithm may break down because there may be many different optimal solutions contributing to a single facet; moreover there may be too many inequalities. Thus this paper considers to what extent a suboptimal solution which enforces a single solution on each facet and/or keeps the number of facets small is amenable to guarantees of feasibility and convergence.

3.2. Facet vs vertex representations of polytopes

A key issue for the user is to ask which representation is more efficient, one based on the vertices or one based on facets. For an arbitrary region such as the MAS one cannot give a simple answer to this. It is difficult to form a systematic link between the number of vertices and facets except for a few cases such as the \( n_x \)-dimensional cube. For a general \( n_x \)-dimensional polytope, it is possible that some facets have far more than \( n_x \) vertices and equally some vertices contribute to many more than \( n_x \) facets.

In summary, if one wants efficient or consistent relationships between facets and vertices then one is steered towards using regular polytopes.
3.3. Convexity

Assume that some global optimum and feasible control/state sequences
\( \mathbf{c}_{i,k} = \{c_k, c_{k+1}, \ldots\} \),
\( \mathbf{x}_{i,k} = \{x_k, x_{k+1}, \ldots\} \), \( \mathbf{M} \mathbf{v}_i + \mathbf{N} \mathbf{c}_{i,k} \leq \mathbf{d} \), are known for a set of initial points \( \mathbf{x}_k = \mathbf{v}_i \), \( i = 1, 2, \ldots, m \).

Then one can use convexity arguments to show that the following sequence \( \mathbf{c}_{k} \) is also feasible.

\[
\begin{align*}
\mathbf{x}_k &= \sum_{i=1}^{m} \lambda_i \mathbf{v}_i \\
\mathbf{c}_k &= \sum_{i=1}^{m} \lambda_i \mathbf{c}_{i,k} \\
\mathbf{x}_k &= \sum_{i=1}^{m} \lambda_i \mathbf{x}_{i,k} \\
\lambda_i &\geq 0, \quad \sum_{i=1}^{m} \lambda_i = 1
\end{align*}
\]

(9)

Moreover, by definition of (4), the use of \( \mathbf{c}_k \) will give rise to convergent state trajectories.

Lemma 3.1. Assuming \( n_x \) states, then given \( n_x+1 \) affinely independent vertices \( \mathbf{v}_i \), \( i = 1, \ldots, n_x+1 \) and the corresponding optimal control sequences \( \mathbf{c}_{i,k} \), a feasible convergent sequence for the initial state \( \mathbf{x}_k \) is given from:

\[
\mathbf{c}_k = \left[ \begin{array}{c} \mathbf{c}_{1,k} \\
\mathbf{c}_{2,k} \\
\vdots \\
\mathbf{c}_{n_x+1,k} \end{array} \right] \\
\lambda = \left[ \begin{array}{c} \lambda_1 \\
\lambda_2 \\
\vdots \\
\lambda_{n_x+1} \end{array} \right]
\]

(10)

if \( \lambda_i \geq 0 \), \( \forall i \); the condition \( \sum_{i=1}^{n_x+1} \lambda_i = 1 \) is given by definition of \( \lambda \) in eqn. (10).

Proof: This is obvious from (9). \( \square \)

Definition 3.2 (Simplex). Define a simplex \( \mathcal{S} \subset \mathbb{R}^{n_x} \) as the convex hull of \( n_x+1 \) affinely independent vertices: one vertex is the origin and the remaining \( n_x \) vertices define a facet not intersecting with the origin.

Corollary 3.1. The computation of \( \lambda_i \) in (10) will be such that \( \sum_{i=1}^{n_x+1} \lambda_i = 1 \) and \( \lambda_i \geq 0 \) if \( \mathbf{x}_k \) is inside the simplex \( \mathcal{S} \) made from these vertices. This is obvious from trivial vector algebra.

Theorem 3.1. If \( \mathbf{x}_k \) lies inside the simplex defined by \( n_x \) vertices and the origin, then a feasible convergent trajectory can be determined from (10) using those vertices.

Proof: The proof follows directly from the Lemma and Corollary above. \( \square \)

The key insight offered in this section is that one can use convexity arguments very easily to form convergent and feasible trajectories for an arbitrary state \( \mathbf{x}_k \) if one has knowledge of feasible and convergent trajectories \( \mathbf{c}_{i,k}, \mathbf{x}_{i,k} \) for a large enough number of possible initial points \( \mathbf{v}_i \) (here denoted vertices) spanning the space and, if \( \mathbf{x}_k \) lies inside the polytope defined by those vertices and the origin. Nevertheless, it is necessary to select the right \( n_x \) vertices, not only to ensure
\( \lambda_i \geq 0, \sum_{i=1}^{n_x+1} \lambda_i = 1 \) but also in general it will be non-trivial to identify the best \( n_x \) vertices from \( m \) possible, \( m \gg n_x \). For instance, Definition 3.1 may suggest a facet which has many more than \( n_x \) vertices in which case several different combinations of \( n_x \) vertices from these may give a simplex in which \( x_k \) lies and satisfying the requirement on \( \lambda_i \).

### 3.4. Building a regular polytope for which each facet has only \( n_x \) vertices

If a facet has only \( n_x \) vertices, then one can use convexity arguments to argue that, for any \( x_k \) for which that facet is active, there is a unique affine linear combination of the vertices as in (9) which will give a feasible convergent trajectory. This section outlines one such set of polytopes which are easy to define and has only \( n_x \) vertices per facet. In essence, the proposed shape will be based from a cube and a cross-polytope.

**Definition 3.3.** Define a polytope \( P \subset \mathbb{R}^{n_x} \) as the following convex hull:

\[
P = \text{hull}\{C, X\}; \tag{11}
\]

where \( C \) is a \( n_x \)D cube of volume \( 2^{n_x} \) centered in the origin, i.e. the convex hull of all sign permutations of the vertices \([\pm 1, \pm 1, \ldots, \pm 1]^T\); and \( X \) is a \( n_x \)D cross-polytope of volume \( (2\mu)^{n_x}/n_x! \) (with \( \mu = \sqrt{n_x} \)) centered in the origin, i.e. the convex hull of the vertices \( \pm e_i \mu, \ i \in \{1, \ldots, n_x\} \).

**Remark 3.1.** The reader will note that \( P \) is a full-dimensional polytope by construction and defines the corners of an \( n_x \)D cube and the centre of the facets of this cube stretched to the same modulus (\( e_i \) being the \( i \)th standard basis vector). The total number of vertices of this polytope is given as \( 2^{n_x} + 2n_x \). The total number of facets, however, grow quicker with \( n_x \), i.e. \( n_x 2^{n_x} \).

Figure 1 demonstrates what \( P \) might look like in the 3D case and the reader can clearly see that each facet has 3 vertices, but vertices contribute to 4 or 6 facets each.

---

2The quality of the vertices is measured by the cost (3) of the resulting trajectory obtained from those vertices. In general, the smaller simplex the better.
3.5. Locating the active facet/vertices on proposed simple shape

Having defined the shape in Definition 3.3, one wants to know how easily one can identify the active facet for an arbitrary initial point \( x_k \), that is to find which are the nearest \( n_x \) vertices such that \( x_k \) lies inside the simplex formed by those vertices and the origin, thus satisfying (9). This section shows that for vertices defined as in Definition 3.3, one can find these vertices with a trivial search, simpler even than that used in Definition 3.1. The idea is simply to find the nearest \( n_x \) vertices to a point; these will always form a simplex, with the origin, enclosing the point. Further note that \( n_x - 2 \) vertices can be given by inspection due to the regular definitions of the vertices.

Algorithm 3.1 (Define active vertices). Assume an \( n_x \)D space, an initial point \( x = [x_1, \ldots, x_{n_x}]^T \) and define the \( j \) vertices of \( P \) as \( t_j \). The active vertices \( t_{ij}^* \) which form a simplex with the origin containing \( x \) in its interior can be found with the following algorithm.

1. Create a vector \( p = [p_1, \ldots, p_{n_x}] \), so that its elements \( p_i \)’s are the indices of the elements of \( x \) sorted from the largest to the smallest absolute values.
2. The active vertices include standard basis vectors corresponding to \( p_1, \ldots, p_{n_x-2} \), with the choice being \( t_i^* = \mu e_{p_i} \text{sgn}(x_{p_i}), \ i = 1, \ldots, n_x - 2 \).
3. The vertex \( t_{n_x-1}^* \) is calculated with \( t_{n_x-1}^* = \sum_{i=1}^{n_x} e_{p_i} \text{sgn}(x_{p_i}) \). The remaining vertex \( t_{n_x}^* \) is taken from the selection of one of the vertices \( v_1, v_2 \): where

\[
\begin{align*}
v_1 &= \mu e_{p_{n_x-1}} \text{sgn}(x_{p_{n_x-1}}); \\
v_2 &= \sum_{i=1}^{n_x-1} e_{p_i} \text{sgn}(x_{p_i}) - e_{p_{n_x}} \text{sgn}(x_{p_{n_x}}).
\end{align*}
\]

The choice is taken by selecting the vertex that is nearer to \( x \) according to the Euclidean distance.
4. Given \([t_1^*, \ldots, t_{n_x}^*]\) and the origin, define the corresponding input trajectory using eqn. (10).

Remark 3.2. Note that only a fixed number of control laws, given by \( 2^{n_x} + 2n_x \), needs to be stored.

Theorem 3.2. If \( x \in P \), Algorithm 3.1 always defines vertices which in turn define a simplex with \( x_k \) inside the simplex. Consequently, the vertices can be used to find a feasible convergent trajectory using (10).

Proof: Note that \( P \) can be tessellated by \( j \) simplices \( S_j \), each of them formed by the convex hull of the origin and the \( n_x \) vertices of the \( j \)th facet, with \( j \in \{1, \ldots, n_x 2^{n_x}\} \). Then it is obvious that \( x \) will lie in one of the \( S_j \)’s, specifically in the simplex dubbed \( S_x \). Algorithm 3.1 constructs the simplex \( S_x \) using the vertices \([t_1, \ldots, t_{n_x-1}]\) calculated in Step 2 and 3, completes it with one more vertex: \( v_1 \) or \( v_2 \) calculated in Step 3, the choice depends in the dimension \( n_x \) of the simplex. Therefore, Algorithm 3.1 can always select \( S_x \) (which is probably a suboptimal choice) if \( x \in P \).

Finally, it is noted that some vertices of \( P \) not included in \( S_x \) could also allow a smaller (valid) simplex for \( x \). The most likely vertex to swap is the vertex \( v_1 \) or \( v_2 \) discarded to form \( S_x \) reducing the size of the simplex and therefore validating if \( S_x \) is or not the best option; Step 3 determines which vertex gives the smallest simplex on the basis that this gives best feasibility although in fact both alternatives would be feasible in general!
3.5.1. Illustration of algorithm to find active vertices

Consider the initial points \( x_1 = [0.2, 0.3, 1]^T \), \( x_2 = [0.5, 0.8, 1]^T \). It is shown that the algorithm gives different choices for the last active vertices; the first \( n_x - 1 \) are the same.

1. The largest component of \( x_1 \) is the 3rd, so include \( \mu e_3 = [0, 0, \mu]^T \) as \( t_1^* \). As the next highest component is the 2nd, and the 3rd component is positive, therefore, include \( t_2^* = e_3 + e_2 + e_1 \). Finally, the last vertex \( t_3^* \) is chosen from \( e_3 + e_2 - e_1 = [-1, 1, 1]^T \) and \( \mu e_2 \), whichever is closer; in this case \( t_3^* = [-1, 1, 1]^T \). Including the origin to complete the simplex, eqn. (10) then gives \( \Lambda = [0.4, 0.25, 0.05, 0.3]^T \) which are clearly all positive and \( \sum \lambda_i = 1 \).

2. In the case of \( x_2 \), clearly one should include \( t_1^* = \mu e_3 \) and \( t_2^* = [1, 1, 1]^T \) and then one from \( [-1, 1, 1]^T \), \( [0, \mu, 0]^T \). In this case the \( t_3^* = [0, \mu, 0]^T \) is closest to \( x_2 \). Including the origin to complete the simplex, eqn. (10) then gives \( \Lambda = [0.29, 0.5, 0.17, 0.04]^T \) which are clearly all positive and \( \sum \lambda_i = 1 \).

3.5.2. Comments on algorithm for finding active vertices

The reader will note that Algorithm 3.1 identifies the active vertices (and implicitly a facet with only \( n_x \) vertices) without at any point needing to define the facet representation for the implied shape. This is key later when the shape will be distorted and hence the vertex combinations making up the facets cannot be determined simply; as will be seen these do not need to be known and the paper uses only the vertices throughout. Also, the number of facets grows faster than the number of vertices.

It is noted that eqn. (10) requires a matrix inversion. However, the matrix to be inverted comprises columns which are mostly scaled standard basis vectors and one or two columns of the form \([\pm 1, \pm 1, \ldots]^T\). Consequently the inversion is trivial in general and one could easily generate highly efficient code for this. Having identified \( \Lambda \), the predicted control is computed from (10) and thus the overall algorithm is highly efficient.

3.6. Summary

This section has outlined the key background information required to develop an efficient parametric solution.

1. Definition of a regular polytope for which all facets have \( n_x \) vertices.
2. A link between optimal trajectories at \( n_x \) predefined initial values and possible feasible trajectories at other points inside the corresponding simplex.
3. A fixed number of vertices \( 2^n + 2n_x \), with the associated control laws, needs to be stored.
4. A highly efficient algorithm for identifying the active vertices for an arbitrary initial point.

It remains to modify this polytope to more general shapes or feasible regions, but as will be shown none of the efficiency benefits or key attributes are lost in doing so.

4. A suboptimal parametric MPC algorithm

This section first develops the vertex based polytopes to encompass a region close to the MCAS and therefore applicable to OMPC. Then, it proposes an efficient suboptimal parametric predictive control law.
4.1. Definition of the vertices of the feasible region

The first objective is to find vertices corresponding to the polytope of Definition 3.3 which lie on the boundary of the MCAS as these are the points furthest from the origin, in those directions, for which a feasible control trajectory is known.

Algorithm 4.1. Given the vertices \( \mathbf{t}_j \) of \( \mathcal{P} \), find points in the same direction which lie on the MCAS by performing the following optimisations

\[
\beta_j^* = \arg \max_{\beta_j, \mathbf{c}_{j,k}} \beta_j \quad \text{s.t.} \quad M\beta_j + N \mathbf{c}_{j,k} \leq 1, \quad \beta_j > 0; \quad (12)
\]

and hence define the vertices \( \mathbf{V}_j = \beta_j^* \mathbf{t}_j \) and corresponding optimal sequences \( \mathbf{c}_{j,k} \) [Capital V is used to distinguish between MCAS vertices and lower case \( \mathbf{t} \) for the regular shape]. Define the convex hull \( \mathcal{P} = \text{hull} \{ \mathbf{V}_1, \mathbf{V}_2, \ldots \} \).

The vertices \( \mathbf{V}_j \) will describe a polytope \( \mathcal{P} \) which may be close to the MCAS, but clearly smaller in general due to the predefined assumptions on the directions of the vertices\(^3\) and restrictions to the complexity. The argument that will be made is that the loss in the volume of \( \mathcal{P} \) as compared to the MCAS is countered by the huge gains in the simplicity of definition as the vertices have regular directions as given in Definition 3.3 and are small in number. An example is given in Figure 2. It is clear that there is a huge gain as compared to the MAS and the feasible region for \( \mathcal{P} \) is quite close to the MCAS given the simplicity of the assumed shape.

\(^3\)It can also be considered an axis rotation of \( \mathcal{P} \) in order to match the semi-axis of the largest ellipsoid contained in the MCAS. This will make \( \mathcal{P} \) even closer to the MCAS. This has not been done here.
Remark 4.1. As noted earlier, the controller partition obtained with mp-QP \( R \) may have a very large number of facets and it is not obvious, a priori, which vertices will make up those facets (it is not necessarily the same as for the shape in Figure 1). In this paper, no attempt is made to compute the facet representation as this is not needed.

4.2. Locating the active vertices from polytope \( P \)

The advantage of building \( P \) from a regular polytope is that the directions of the vertices are the same and therefore locating the simplex in which a current point lies can be based on the same logic as Algorithm 3.1.

Algorithm 4.2 (Define active vertices). Assume an \( n_x \times D \) space, an initial point \( \mathbf{x} = [x_1, \ldots, x_{n_x}]^T \) and use the vertices \( V_j \) produced in Algorithm 4.1. The active vertices \( V_j^* \) which form a simplex with the origin containing \( \mathbf{x} \) in its interior can be found with the following algorithm.

1. Define a correspondence \( t_i \equiv V_i \), in that the values differ only by a positive scaling factor \( \beta^*_i \).
2. Follow the steps of Algorithm 3.1 (note that \( v_1, v_2 \) of step 3 needs to be scaled to the corresponding \( V_j \) term in order to select the nearest to \( \mathbf{x} \)).
3. The final choices \( V_i^* \)'s will be the corresponding \( V_i \)'s from \( t^*_j \) obtained from Algorithm 3.1.

4.3. Proposed suboptimal control law

This section defines the proposed MPC algorithm assuming that the only information available to the user is:

- The vertices \( V_j \) of \( P \) on the boundary of the MCAS.
- The optimal sequences \( \mathbf{c}_{j,k} \) corresponding to each vertex.
- The constraint inequalities (2).
- The tail of the optimum sequence taken at the previous sample; typically the tail is taken as \( \mathbf{c}_{\text{tail},k} = [c_{k-1}, c_{k+1|k-1}, \ldots, c_{k+n_c-2|k-1}, 0]^T \).

The proposed MPC algorithm is as follows.

Algorithm 4.3. At each sample perform the steps:

1. If the current state satisfies \( M_{x_k} \leq 1 \), the unconstrained control law \( u_k = -Kx_k \) is feasible and should be used, otherwise do steps 2-6.
2. Using the current state \( \mathbf{x}_k \), find the active vertices \( V_j^* \) (and corresponding \( \mathbf{c}_{j,k} \)'s) using Algorithm 4.2.
3. Find the feasible control sequence \( \mathbf{c}_k \) (using \( V_j^* \)'s, \( \mathbf{c}_{j,k} \)'s) with eqn. (10) \(^4 \) [Note, strictly this is guaranteed feasible iff \( \lambda_i > 0, \ i = 1, \ldots, n_x + 1 \)]. If \( \lambda_i < 0 \) implement the first element of \( \mathbf{c}_{\text{tail},k} \) in (4), update the sample instant and return to Step 1.
4. Define the optimum sequence as (\( \alpha \in \mathbb{R} \))

\[
\mathbf{c}_k^* = (1 - \alpha) \mathbf{c}_k + \alpha \mathbf{c}_{\text{tail},k}.
\]  

\(^4k + i|k \) means the prediction for sample \( k + i \) made at sample \( k \).
5. Perform the optimisation

\[
\begin{align*}
\min_\alpha J_c &= (1 - \alpha) c_{k} + \alpha c_{\text{tail},k} \\
\text{s.t.} \quad Mx + Nc_{k}^{*} \leq 1; \\
0 \leq \alpha \leq 1.
\end{align*}
\]

(14)

Use the optimum \( \alpha \) to compute \( c_{k}^{*} \) of eqn. (13).

6. Implement the first element of \( c_{k}^{*} \) in (4), update the sample instant and return to Step 1.

**Theorem 4.1.** In the nominal case, Algorithm 4.3 has a guarantee of convergence and recursive feasibility.

**Proof:** Convergence relies on the standard argument that the inclusion of \( c_{\text{tail},k} \) allows an upper bound on the cost \( J_c \) and moreover \( J_c(k) \leq J_c(k - 1) - c_{k-1}^{T}Sc_{k-1} \) and therefore \( J_c \) is Lyapunov; once \( J_c = 0 \) the system is inside the MAS and unconstrained control applies. Recursive feasibility also follows automatically from \( c_{\text{tail},k} \) being feasible by definition. \( \square \)

Ironically, there is no guarantee that the set \( \mathcal{P} \) is invariant and thus trajectories which begin in \( \mathcal{P} \) may go outside \( \mathcal{P} \) and thus the optimal \( c_{k} \) produced by Step 3 of Algorithm 4.3 may be infeasible (because \( \lambda_i < 0 \)). In this case one may rely more extensively on the tail until the state trajectory re-enters \( \mathcal{P} \). However, insisting on invariance of \( c_{k} \) as opposed to \( c_{k}^{*} \) each sample instant would require either more complex set definitions and/or more demanding optimisations and thus defeats the object of this paper.

Nevertheless, should this be desired, one could easily determine, offline, a separate region \( \mathcal{P}_2 \subset \mathcal{P} \) such that \( x_k \in \mathcal{P}_2 \Rightarrow x_{k+i} \in \mathcal{P}, \quad \forall i > 0 \), thus preserving the feasibility of both \( c_{\text{tail},k} \) and \( c_{k} \).

**Remark 4.2.** This paper has not looked at robustness issues but it is also worth repeating that the focus here is simplicity and in general algorithms with strong robustness results come at a price of increased complexity.

5. Numerical examples

This section will demonstrate several aspects of the proposed algorithm, but primarily the two main attributes: (i) despite the restriction to so few points, the performance is only slightly suboptimal and moreover recursive feasibility and convergence are achieved for all initial points in \( \mathcal{P} \) and (ii) the complexity is always low relative to the norms in the literature for a given state dimension. To this end, this section will give four examples, one with 2 states (slightly under damped mass-spring-damper), two with 3 states (one a simplified two-input-two-output model of electrical power generation) and one with 4 states. In each case \( n_c = 5 \) as much beyond this offered little benefit by way of feasibility. Details of the systems used for the examples can be found in the Appendix.
5.1. Performance and feasibility comparisons

Figures 3-6 show the efficacy of the proposed approach as compared to a standard OMPC algorithm in terms of performance and feasibility for two different choices of control weighting $R$. Several initial conditions are chosen randomly with Monte-Carlo method.

- The top plots show the normalised size (distance to boundary of $P$ from the origin) of the feasible region for a large number of different directions, as compared to the MCAS. For completeness the figure includes the corresponding distance for the MAS, that is when $n_c=0$. The directions are ordered by normalised magnitude of the distance from the origin to $P$.

- The lower plots show the normalised performance of Algorithm 4.3 as compared to OMPC, again for a large number of random initial points on the boundary of $P$; the same points as for the upper figures.

These figures demonstrate two clear conclusions. First, despite the very restricted shape of $P$, there can be significant feasibility improvements as compared to the MAS, although unsurprisingly it is smaller than the MCAS which may have a very complex shape. Second, the performance degradation from using a suboptimal strategy is often quite small (here only the 3rd example has serious performance degradation) and, unsurprisingly again, the loss in performance was dependent on the initial condition.

In order to further emphasise the feasibility improvements of $P$ over the MAS and the proximity of the boundary of $P$ to the MCAS, the unsigned volumes for these polytopic sets are shown in Table 1. In this table, it is evident that $P$ allows larger feasible sets than the MAS, specially in the cases when the optimal feedback gain $K$ is highly tuned. On the other hand, the volume of $P$ is obviously smaller than the MCAS due its restricted shape, and in fact, there may be cases when the approximation is unacceptable (e.g., a non-symmetric MCAS) where other strategies not covered in this paper may be needed.
Figure 4: Cost and feasibility comparisons for example 2.

Figure 5: Cost and feasibility comparisons for example 3.
Figure 6: Cost and feasibility comparisons for example 4.

Table 1: Unsigned volumes of the polytopic sets of the examples.

<table>
<thead>
<tr>
<th>Example 1</th>
<th>Example 2</th>
<th>Example 3</th>
<th>Example 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R=10$</td>
<td>$R=0.1$</td>
<td>$R=10$</td>
<td>$R=0.1$</td>
</tr>
<tr>
<td>MAS</td>
<td>4.99</td>
<td>50.48</td>
<td>2313</td>
</tr>
<tr>
<td>$P$</td>
<td>17.91</td>
<td>69.58</td>
<td>9607</td>
</tr>
<tr>
<td>MCAS</td>
<td>20.74</td>
<td>73.42</td>
<td>12983</td>
</tr>
</tbody>
</table>
5.2. Complexity comparisons

In this section complexity of the proposed strategy is compared with the complexity of the exact parametric solution to the optimisation problem (mp-QP) (Bemporad et al., 2002a) and to an algorithm to approximate the exact parametric solution (Bemporad and Filippi, 2006). It is worth to repeat here that the aim of methods proposed in this paper is not to approximate the parametric solution of the optimisation problem, is rather to provide a MPC algorithm with constraint handling capabilities for the implementation in hardware with very low processing power and memory resources (e.g. a PLC) achieving enough guarantees of stability and recursive feasibility.

The idea of Bemporad and Filippi (2006) is to partition the MCAS into simplices where the approximate solution inside each simplex is given by linear interpolation of the exact solution at the vertices; for each simplex, if the resulting absolute error in the objective exceeds a prescribed tolerance then it is divided into smaller simplices where it applies recursively. The initial set of simplices is obtained with a Delaunay tessellation (DT) (Yepremyan and Falk, 2005) of the MCAS. The resulting approximate solution is organised over a tree structure for efficiency of evaluation, a similar tree structure based on cubes was used in Johansen and Grancharova (2003).

The approach of Bemporad and Filippi (2006) gives substantial reduction in the number of regions compared to mp-QP (Bemporad et al., 2002a) and the resulting size of the feasible set for the approximate solution is the same as the MCAS. However, as argued in the introduction, any result which is based on sub-division until the difference from the optimal is small will, in general, lead to a large number of regions. Even if the error bound is generously relaxed the minimum number of simplices to describe the MCAS will depend on the shape and number of facets of the MCAS (for the algorithm that determines the initial tessellation) and in this case tree structure will not be very useful, all simplices will be at the first level of the tree (just after the root of the tree which is the set \( \mathbb{R}^n \)) and a conventional search will follow, i.e. the simplices have to be tested one by one until the simplex containing \( x \) is found. In contrast, Algorithm 3.1 will find the simplex containing \( x \) in a very small fixed number of arithmetic operations.

It is very important to mention that the original paper of Bemporad and Filippi (2006) does not discuss issues about implementing this parametric approximation in the receding horizon context, so stability and recursive feasibility cannot be guaranteed a priori and further conditions need to be considered.

Table 2 focuses on the complexity of the solution as compared to a standard parametric approach of Bemporad et al. (2002a) and the approximation of Bemporad and Filippi (2006). Specifically, the table compares (i) the number for regions for a standard mp-QP; (ii) the minimum number of simplices needed to describe the MCAS obtained from a DT which is the initialization algorithm from Bemporad and Filippi (2006) (with no error bounds to get the minimum number of simplices); and (iii) vertices for Algorithm 4.3. This is a fair statement of data storage requirements. However, it should be noted further, that the search efficiency for the proposed algorithm is far better, even if the number of regions/simplices/vertices were the same.

It is clear that the proposed algorithm allows significant reductions in complexity.

6. Implementation of the algorithm on a Programmable Logic Controller

The principal objective of Algorithm 4.3 is to be as efficient as possible to lower memory and processing requirements, thus improving the potential for the application on standard industrial hardware. This section introduces the PLC with its limitations and describes the implementation of Algorithm 4.3.
Table 2: Number of regions obtained using mp-QP, simplices from a Delaunay tessellation (DT) and vertices using Algorithm 4.3.

<table>
<thead>
<tr>
<th></th>
<th>Regions (mp-QP)</th>
<th>Simplices (DT)</th>
<th>Vertices (Alg. 4.3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R=10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R=0.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example 1</td>
<td>67 63</td>
<td>12 12</td>
<td>8 8</td>
</tr>
<tr>
<td>Example 2</td>
<td>77 77</td>
<td>31 30</td>
<td>14 14</td>
</tr>
<tr>
<td>Example 3</td>
<td>297 285</td>
<td>99 92</td>
<td>14 14</td>
</tr>
<tr>
<td>Example 4</td>
<td>727 251</td>
<td>137 94</td>
<td>24 24</td>
</tr>
</tbody>
</table>

Figure 7: Allen Bradley PLC – SCL500 processor family.

6.1. Allen Bradley – Rockwell Automation PLC

PLCs are by far the most accepted computers in industry which offer a reliable, safe and robust system. The arrangement and packaging of the PLC system is tailored for ease of integration into on-site control racks or cabinets with minimal effort. PLCs are also suited for the ease of implementation of standard wiring terminations. Each of these means that any on-site technician will have no difficulty or require any additional skills or tools when it comes to installing a controller. PLC systems also offer an added advantage that the program can be monitored online. The visual nature of the language means that the user can view in real-time the changing nature of bits, the value of counters or timers etc and how these relate to the overall program structure. Another advantage which PLC systems afford is that in the most cases, adjustments to programs can be made online without having to take the process or system under control offline. This is obviously an attractive property to industry as shutting down parts of a process can be a very costly affair indeed. However, it must be noted that this does present some safety issues which must be carefully addressed beforehand.

Nevertheless, normally their use is only to implement control sequences in open loop and/or different structures of PID controllers. For the purposes of this paper, the implementation is based on the family of SLC500 processors belonging to the Allen Bradley PLC systems, e.g. see Fig. 7.

The Allen Bradley set of PLC includes 64 Kbs of memory size with 0.37 $\mu$-sec of bit execution and the facilities to be programmed in 3 of 5 languages in agreement with the IEC 61131-3 stan-
standard using Control Logix 5000\textsuperscript{TM} software programming package. Each of these allows for any combination of programming languages to be used for a single project. These three languages are:

1. **Ladder Diagram** is a graphical language that uses a standard set of symbols to represent relay logic. The basic elements are coils and contacts which are connected by links. *Ladder logic* is thus a highly visual, easy to understand, program and diagnose as previously stated.

2. **Function Block Diagram** is a graphical language that corresponds to circuit diagrams. The elements used in this language appear as blocks wired together to form circuits. The wires can communicate binary and other types of data between *Function Block Diagram* elements (e.g. Real, Integers, etc.).

3. **Structured Text** is a general purpose, high-level programming language, similar to PASCAL or C. *Structured Text* is particularly useful for complex arithmetic calculations, allows to create boolean and arithmetic expressions as well a structured programming constructs such as conditional statements (IF, THEN, ELSE). Functions and function blocks can be invoked in this language.

### 6.2. PLC programming issues

There are some barriers and criteria required before an MPC algorithm can be coded effectively into the PLC; these are discussed next.

The SCL500 Control-Logix Controllers together with RS-Logix 5000 allows for the memory allocation of matrices (which it refers to as data arrays), for up to 3 dimensions. However, with the exception of one-dimensional, simple element by element arithmetic, it cannot be easily performed other matrix operations, notably: transposition, inversion and multiplication. To achieve such functions, it is thus necessary to code functions from scratch within the software to perform such operations. One could code an entire control algorithm in Structured Text, but for the ease of understanding by technicians it is strongly advisable to program most of the algorithm in a graphical language. In this way, the technical staff could view all the realtime data of the controller and debug the program if need it, in a more intuitive way. Finally, all the computations to calculate the next control movement should be done in a limited time dictated by the sampling period, so the computational load should be kept as low as possible to avoid accumulated errors during the tests. There is a need therefore to check and evaluate the algorithm timing after coding while noting that a bigger program inevitably requires more memory and therefore a more powerful PLC with the associated cost.

### 6.3. Implementation of Algorithm 4.3 on the target PLC

Algorithm 4.3 is programmed in the High Priority Periodic Execution Group (This periodicity is set up with the chosen sample time). The file structure of the program is shown in Figure 8, and the description of the routines is presented next:

- **MPC\_MAIN** (Ladder Logic Diagram). This is the main routine whose purpose is to control the program execution, calling routines in the correct order. Specifically, this routine first calls **Observer**; if $x_k$ satisfies $Mx_k \leq 1$, sets $c^*_{k} = 0$, $c_{tail,k+1} = 0$ and calls **Controller\_Output**; otherwise, calls the subroutines sequentially: **Vertex\_Id**, **Optimisation** and **Controller\_Output**.

- **Observer** (Structured text). This subroutine is used to reconstruct the state vector using a Kalman filter. Invokes the subroutine **Matrix\_Multiply** to complete the operations.

- **Vertex\_Id** (Structured text). This subroutine gets from **Data\_Vertex** the optimal sequence $c_k$ associated with the active vertices using Algorithm 4.2 and eqn. (10).

- **Data\_Vertex** (Structured text). This is not properly a program routine, is a data bank. Contains the information about the vertices $V_j$ of the polytope $P$ and associated optimal solutions $c_j$. 

18
Figure 8: File structure of Algorithm 4.3 in the target PLC.

- **Optimisation** (Structured text). This subroutine is used to calculate the optimum $c^*_{k}$ from equation (13) finding the optimal $\alpha$ in problem (14). Before ending, this routine stores $c_{\text{tail},k+1}$.
- **Controller_Output** (Structured text). This subroutine sends to the plant the calculated output using $u_k = -Kx_k + c_k$.
- **Matrix_Multiply** (Structured Text). The matrix dimensions are passed to this along with two matrices. The routine returns the resulting answer matrix of the multiplication.
- **Matrix_Inverse** (Structured Text). The matrix dimensions are passed to this along with one matrix. The routine returns the resulting inverted matrix using an augmented matrix with Gaussian elimination.
- **PLANT_SIMULATION** (Ladder Logic Diagram). This subroutine is for development purposes only. It is used to simulate the plant dynamics/response and thus off-line testing of the controller coding.

7. Experimental laboratory test

This section shows the experimental results from applying the MPC law via the PLC, the aim is to show the effectiveness of Algorithm 4.3 on real hardware. The process consists of a motor fitted with a speed sensor, the control objective is to regulate the speed of the motor by manipulation of the input voltage. The mathematical model of the system with a sampling time of 0.5 sec is:

$$
\begin{align*}
x_{k+1} &= \begin{bmatrix} 0.93 & -0.007 \\ 1 & 0 \end{bmatrix} x_k + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u_k; \\
y_k &= \begin{bmatrix} -0.1078 & 29.68 \end{bmatrix} x_k;
\end{align*}
$$

with constraints $-3.5V \leq u \leq 3.5V$, $-0.05 \leq \Delta u \leq 0.05V$. In order to get a closed polytope, the output is bounded to $-1100 \text{ RPM} \leq y \leq 1100 \text{ RPM}$. The experimental validation of the model is shown in Fig. 9.
In order to allow tracking of a time-varying reference, it is necessary to use an augmented state vector formulation:

$$\xi_k = \begin{bmatrix} x_k & d_k & u_{k-1} & r_k \end{bmatrix}^T,$$

(15)

where $x_k$ are the model states of the controlled plant, $d_k$ is the disturbance estimate, $u_{k-1}$ is the previous control input and $r_k$ is the reference signal. Therefore, the dynamics are formulated in $\Delta u$-form; in this framework, the system input at time $k$ is $\Delta u_k$ whereby $u_{k-1}$ is an additional state in the dynamical model, i.e. the system input can be obtained as $u_k = u_{k-1} + \Delta u_k$.

Remark 7.1. Note that the augmented vector $\xi_k$ replaces $x_k$ in all the algebra presented in Sections 2-6.

The tuning parameters for the controller are $n_c = 3$, $R = I$, $Q = I$. Figure 10 shows the conventional mp-QP partitions and the polytope $P$ for this problem in the case when $d_k = 0$, $u_{k-1} = 0$. In this case submpOMPC only needs to store information about 42 vertices and the associated control laws opposed to 97 regions from the mp-QP.

The program uses 15% of the available storage of the PLC including required memory for I/O, running cache and other necessary subroutines as it can be seen from the properties of the controller with the RSLogix® programming tool in Figure 11. This number indicates that one can even think in embed more than one controller in a single PLC as long as the scheduling is possible.

Two setpoint step changes are demanded; the results show that the proposed controller is tracking the setpoint accurately as can be seen in Figure 12. To complete the assessment of the implemented program, the diagnostics tool from the hardware (shown in Fig. 13) displays that the time for scanning the program each sample time which in the worst case is 9.85 ms while the elapsed time between triggers (sampling instants) for the speed process oscillates around $100.00 \pm 0.425$ ms. The significance of this is the potential to apply the algorithm on much faster processes.
Figure 10: Cuts on the mp-QP controller partition and polytope $\mathcal{P}$ for the speed process ($d_k = 0$, $\Delta u_{k-1} = 0$).

Figure 11: Algorithm 4.3 memory usage on the target PLC.
Figure 12: Experimental test for the speed process.

Figure 13: Execution time and sampling jittering of Algorithm 4.3.
8. Conclusions

This paper has demonstrated that a simple but very efficient constrained MPC algorithm can be effectively coded in a standard PLC unit. Thus, it is an industrial affordable alternative for replacing standard controllers with poor performance particularly in loops primarily controlled with PLC units. Along with this, the paper has made two important contributions.

First, it has evolved work on efficient parametric solutions to predictive control by removing the requirement for a specified bound on suboptimality and instead replaced that assumption with a specified bound on solution complexity. This is done without loss of recursive feasibility and stability and is a key advance when considering application to systems with low processor capability or highly complex optimal parametric solutions.

Secondly, it has proposed a novel choice of regions which allows for highly efficient search algorithms in conjunction with a simple definition which allows more shape flexibility than $n_xD$ cubes. Specifically, the choice of regions allows one to define the feasible region in terms of facets with at most $n_x$ vertices thus allowing simple convexity statements and thus simple computations; critically the facets do not need to be enumerated explicitly offline and are computed implicitly, with a trivial iteration, as required; hence the data storage requirements are lower.

While it would be impossible to give a generic statement that the proposed approach always gives at most a given degree of suboptimality or percentage loss in feasibility, the numerical examples show that these comparisons are easy to determine and in many cases the gain in simplicity and efficiency far outweighs any performance loss. Future work will look at the potential of using this approach recursively, that is to derive smaller additional polytopes near the boundary as is done typically by authors using a $n_xD$ cube assumption.

References


Appendix A. Systems for the numerical examples

Example 1

\[ A = \begin{bmatrix} -0.2 & -1 \\ 1 & 0 \end{bmatrix}; \quad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}; \]

\[ \bar{u} = 0.4 = -u; \quad \Delta u = 10 = -\Delta u; \quad y = 10 = -y. \]

Example 2

\[ A = \begin{bmatrix} 1.4 & -0.105 & -0.108 \\ 2 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}; \quad B = \begin{bmatrix} 0.2 \\ 0 \\ 0 \end{bmatrix}; \]

\[ C = \begin{bmatrix} 5 & 7.5 & 5 \end{bmatrix}; \]

\[ \bar{u} = 4 = -u; \quad \Delta u = 1 = -\Delta u; \quad y = 12 = -y. \]

Example 3

\[ A = \begin{bmatrix} 0.914 & 0 & 0.04 \\ 0.166 & 0.135 & 0.005 \\ 0 & 0 & 0.135 \end{bmatrix}; \quad B = \begin{bmatrix} 0.054 & -0.075 \\ 0.005 & 0.147 \\ 0.864 & 0 \end{bmatrix}; \]

\[ C = \begin{bmatrix} 1.799 & 13.216 & 0 \\ 0.823 & 0 & 0 \end{bmatrix}; \]

\[ \bar{u} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} = -u; \quad \Delta u = \begin{bmatrix} 0.25 \\ 0.5 \end{bmatrix} = -\Delta u; \quad y = \begin{bmatrix} 3 \\ 1 \end{bmatrix} = -y. \]

Example 4

\[ A = \begin{bmatrix} 0.9 & -0.105 & -0.108 & 0.2 \\ 0.6 & 0 & 0 & -0.1 \\ 0 & 0.8 & 0 & 0.3 \\ 0 & 0 & 0.8 & 0 \end{bmatrix}; \quad B = \begin{bmatrix} 2 \\ 0 \\ 0 \\ 0.5 \end{bmatrix}; \]

\[ C = \begin{bmatrix} 5 & 7.5 & 5 & 1 \end{bmatrix}; \]

\[ \bar{u} = 2 = -u; \quad \Delta u = 0.08 = -\Delta u; \quad y = 2.4 = -y. \]