Finitely Parameterised Implementation of Receding Horizon Control For Constrained Linear Systems

María M. Serón  
marimar@eie.fceia.unr.edu.ar  
Departamento de Electrónica  
Universidad Nacional Rosario  
Riobamba 245 bis, 2000 Rosario  
Argentina

Graham C. Goodwin  
eegcg@ee.newcastle.edu.au  
Centre for Integrated Dynamics & Control  
The University of Newcastle  
Callaghan, NSW 2308  
Australia

José A. De Doná  
eejose@ee.newcastle.edu.au

Abstract

Recent work by the authors has provided a finitely parameterised characterisation of receding horizon control (RHC) for linear models with quadratic performance index and linear constraints. Using a closed-form solution to quadratic programming, we have expressed the RHC law via a "look-up table" consisting of a partition of the state space into regions in which the corresponding control law has an explicit form. This paper investigates numerical properties of the table look-up implementation of the RHC solution and compares its performance with traditional methods for on-line optimisation (such as active constraint methods for quadratic programming). Issues that we consider include computation times, code complexity and data storage.

1 Introduction

Receding horizon control (RHC) is a state feedback strategy that employs the first of a sequence of control laws corresponding to the solution of an open-loop optimal control problem. When the open-loop optimisation problem is constrained, then the RHC technique has traditionally used on-line optimisation to compute the current control for the current state rather than characterising the whole control law for all values of the state in (a region of) the state space. This on-line implementation of RHC is usually referred to as model predictive control (MPC) and has been a successful technique in industrial applications for decades (see e.g., [5]).

Recent work by the authors [6, 7] and other research teams [1, 2] has provided a finitely parameterised characterisation of the RHC control law for linear models with quadratic performance index and linear constraints. In particular, in [6, 7], the solution was obtained by transforming the underlying open-loop optimal control problem into an equivalent quadratic programme, and then geometric arguments were utilised to solve this derived problem. The resulting solution consists of a partition of the state space into a finite number of regions in which the corresponding control law is an affine function of the state. In this way, RHC is presented as a piece-wise affine switching strategy that can be pre-computed off line.

Other results pertaining to off-line solutions of RHC problems have been recently reported in [3, 8, 1, 4]. In [3], the authors have shown that the RHC problem has, in a non-trivial region of the state space, a closed-form solution that is identical to clipping the unconstrained solution. Independently, [8] obtained a closed-form solution for horizon $N = 1$. In [4] a suboptimal explicit RHC solution is developed that reduces the computational complexity of the optimal explicit solution for large horizons.

Based on these findings, it is relevant to examine whether it is best to compute on-line the current control move for the current state or to characterise the control law off-line for all values of the state and use table look-up for on-line control. A key issue then becomes a trade-off between on-line computation and off-line data storage and retrieval. In this paper we investigate numerical properties of the table look-up implementation of the RHC solution and compare its performance with active constraint methods for quadratic programming, typically used in MPC. We consider first a simple table look-up implementation that retrieves the RHC solution by searching all the regions of the partition in one step. This implementation can be coded using Matlab in just a few lines but performs more floating point operations than quadratic programming. We then implement the table look-up using a binary search tree which, at the expense of larger data storage, outperforms quadratic programming’s computation.
speed and number of floating points operations.

2 Closed-Form Solution to Quadratic Programming

We briefly review here the main result of [6, 7]. Although the notation in this section is fairly involved, the idea is quite straightforward and amounts to a systematic enumeration of all possible combinations of active and inactive constraints. We consider the quadratic programme (QP):

$$ u^{\text{OPT}} = \arg\min_{Lu \leq M} V(u). $$

(1)

where

$$ V(u) = u^T W u + 2u^T f, \quad u \in \mathbb{R}^{Nm} $$

(2)

$N, m$ are positive integers, $W \in \mathbb{R}^{Nm \times Nm}$, $W = W^T > 0$, $f \in \mathbb{R}^{Nm}$. For simplicity, the constraints

$$ Lu \leq M $$

(3)

are assumed of the form

$$ L = [I \quad -I]^T, \quad M = [\sigma \ldots \sigma]^T, $$

(4)

where $I$ denotes the identity matrix of dimension $Nm \times Nm$, and $\sigma$ is a positive number. Extensions to more general cases are given in [7].

We define the ordered set $\ell$ of $N$ indices, $1 \leq \ell \leq Nm$, selected from $\mathcal{I}_{Nm} = \{1, 2, \ldots, Nm\}$:

$$ \ell = \{\ell_1, \ell_2, \ldots, \ell_N\} $$

(5)

where $\ell_0 = 0$ and $\ell_k \in \{\ell_{k-1} + 1, \ldots, Nm - (N - k)\}$, $k = 1, \ldots, N$. The set $\ell$ identifies the constraints that are active in each region. Also, we define the set difference

$$ s = \{s_1, \ldots, s_{Nm-N} : s_k \in \mathcal{I}_{Nm} \text{ and } s_k \notin \ell\}. $$

(6)

For a matrix $A$, and sets of indices $\ell$ and $s$, the notation $A(\ell, s)$ identifies the submatrix of $A$ formed by selecting the rows of indices in $\ell$ and the columns of indices in $s$ (a colon indicates that all rows or columns are selected).

Consider the matrix $W$ in (2) and the sets (5) and (6). We introduce the matrices

$$ L_s = \begin{cases} 0 & \text{if } s = \emptyset, \\ [W(s,s)]^{-1}W(s,:) & \text{if } s \neq \emptyset. \end{cases} $$

(7)

$$ L_{s \cup \ell_k} = [W(s \cup \ell_k, s \cup \ell_k)]^{-1}W(s \cup \ell_k,:), $$

(8)

where $s \cup \ell_k$ is the union of set $s$ and element $\ell_k \in \ell$. We also define, for $k = 1, 2, \ldots, N$, the row vector

$$ L^k_s \equiv L_{s \cup \ell_k}(\ell_k - k + 1,:). $$

(9)

Let $V_\sigma(\ell)$ be the set of vertices in $\mathbb{R}^N$ of the hypercube $[-\sigma, \sigma]^N$. Then, given $\ell$ and $s$ defined in (5) and (6), respectively, and a vertex $v \in V_\sigma(\ell)$, we define the $(Nm \times 1)$ vectors

$$ \sigma_\ell^+ = \begin{bmatrix} I(\ell,:)^{-1}v \\ I(s,:)^{-1}v \end{bmatrix}, \quad \sigma_\ell^- = \begin{bmatrix} v \\ -v \end{bmatrix}. $$

(10)

The following theorem [7] gives the solution of (1).

Theorem 2.1 (Solution of QP (1)) Consider the matrix $W$ and the vector $f$ in (2), and $L_s$, $L^k_s$ defined in (7), (9), respectively. For $N = 1, 2, \ldots, Nm$, form the sets $\ell$ and $s$ as in (5) and (6), respectively; for each pair $(\ell, s)$ consider all vertices $v = [v_1, \ldots, v_k, \ldots, v_N]^T \in V(N)$; for each triple $(\ell, s, v)$ form $\sigma_\ell^+$ and $\sigma_\ell^-$ as in (10). The triple $(\ell, s, v)$ defines the region

$$ R_\ell^c : \begin{cases} L_s^k u \geq L^k_s \sigma_\ell^+ & \text{if } v_k > 0, \\ \leq L^k_s \sigma_\ell^- & \text{if } v_k < 0, \quad k = 1, \ldots, N, \\ L_s \sigma_\ell^+ \leq L_s^k u \leq L_s \sigma_\ell^- \end{cases}. $$

(11)

Then if $u^{\text{OPT}} = -W^{-1}f \in R_\ell^c$, the optimal solution $u^{\text{OPT}}$ in (1) takes the form

$$ u^{\text{OPT}} = v $$

If $N = Nm$:

If $N < Nm$:

$$ u^{\text{OPT}} = \left[I(\ell,:)^{-1}-[W(s,s)]^{-1}f(s)+W(s,:)v]\right] $$

(12)

The global solution of the QP (1) is then given: (i) by all $N_r = \sum_{s=1}^{Nm}Nm/N$ regions outside the constraint polyhedron defined by (11) with optimal control (12); and (ii) by

$$ u^{\text{OPT}} = -W^{-1}f, $$

(13)

if $-W^{-1}f$ is inside the constraint polyhedron, i.e., if it satisfies $-\sigma \leq -W^{-1}(k,:)f \leq \sigma, \quad k = 1, 2, \ldots, Nm$.

2.1 Application to Receding Horizon Control

We consider the following RHC problem. For the system model given by

$$ x(k+1) = Ax(k) + Bu(k), \quad k = 0, 1, 2, \ldots, $$

(14)
where \( x(k) \in \mathbb{R}^n \), and \( u(k) \in \mathbb{R}^m \), we pose the following finite-horizon open-loop optimal control problem: given the current state measurement \( x(k) = x \), find the \( N \)-move control sequence \( \mathcal{U} = \{ u(k), u(k + 1), \ldots, u(k + N - 1) \} \) that minimises the performance index:

\[
V_N(x, \mathcal{U}) = \sum_{t=k}^{k+N-1} [x^T(t)Qx(t) + u^T(t)Ru(t)] + x^T(k + N)Px(k + N). \quad (15)
\]

In (15), \( N \) is the prediction horizon, and \( Q \geq 0 \), \( R > 0 \) and \( P > 0 \) are weighting matrices. The minimisation of (15) is performed under magnitude constraints on the input of the form

\[
|u_i(k + j)| \leq \sigma, \quad \sigma > 0, \quad (16)
\]

for \( i = 1, 2, \ldots, m, \ j = 0, 1, \ldots, N - 1 \). For simplicity we assume equal constraints on each input but extension to unequal and/or asymmetric constraints is straightforward.

Let \( \mathcal{U}^{\text{opt}}(k; x) = \{ u_1^{\text{opt}}(k; x), u_2^{\text{opt}}(k + 1; x), \ldots, u_m^{\text{opt}}(k + N - 1; x) \} \) be the optimising sequence. The first control move \( u^{\text{opt}}(k; x) \) is applied to the system (14) at time \( k \), that is,

\[
u(k) = K_N(x) \triangleq u^{\text{opt}}(k; x), \quad (17)
\]

and then the whole procedure is repeated at time \( k + 1 \) with the new initial state \( x(k + 1) = x \).

It is easy to show that (14)-(16) can be expressed as a QP of the form (1). Indeed, we can express the performance index (15) as

\[
V_N(x, u) = \mathbf{V} + u^T W u + 2u^T F x. \quad (18)
\]

In (18), \( \mathbf{V} \) is independent of \( u \), and the matrices \( W, F \) are easily constructed using the problem data. Finally, constraints of the form (16), can be expressed as linear constraints on \( u \) of the form (3). Note that (18) depends on the state vector \( x \in \mathbb{R}^n \), so the associated QP falls into the class of multi-parametric quadratic programmes (i.e., programmes that depend on a “vector of parameters”, the state vector \( x \) in this case) [1].

We can readily apply Theorem 2.1 to the above problem and obtain a closed form for the RHC law (17). As \( x \) takes values in \( \mathbb{R}^n \), the vector \( u^{\text{opt}}(k; x) = -W^{-1} f - W^{-1} F x \) takes values in \( \mathbb{R}^m \) and will belong to a region \( R_L \) of the form (11). The RHC law can then be evaluated as the first \( m \) components of the optimal control (12). Specifically, we define the sets:

\[
X_L = \{ x \in \mathbb{R}^n \mid -W^{-1} F x \in R_L \}, \quad (19)
\]

with \( R_L \) as in (11). Then if \( x \in X_L \) the RHC law (17) is obtained from (12) as

\[
K_N(x) = [I_m \ 0 \ \cdots \ 0] u^{\text{opt}}. \quad (20)
\]

Thus, to retrieve the complete solution in the state space, we first compute the region partition as described in Theorem 2.1 and then we transform it into a partition of the state space using (19). If \( n \neq Nm \), then the transformation \( u^{\text{opt}} = -W^{-1} F x \) spans a lower dimensional subspace of \( \mathbb{R}^m \) and so some of the regions \( X_L \) in (19) will be empty. Hence, the partition has to be post-processed to eliminate redundant inequalities and empty regions. (Note that this is necessary in order to optimise table look up times.) If \( n \geq Nm \), then the computation of the region partition as described in Theorem 2.1 and the transformation into a partition of the state space using (19) directly gives the complete RHC partition with no need for further processing.

\[\text{3 Off-line Computations}\]

We have implemented an algorithm in Matlab 4 that computes the region partition of Theorem 2.1 and transforms it into a state space partition using (19). The computation of each polyhedral region \( X_L \) is performed using the Geometric Bounding Toolbox for Matlab 4 (by S. Veres and S. Hermansmeier), which gives a compact representation of the regions and detects the empty ones. The total number of regions that have to be computed (including the empty ones) is \( 1 + N_r = 3^{Nm} \), with \( N_r \) as in the statement of Theorem 2.1.

As an illustration of the method, consider the system (14) with

\[
A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}, \quad C = [1 \ 0], \quad (21)
\]

which is the zero-order hold discretisation, with a sampling period of 1, of the double integrator \( x_1(t) = x_2(t), x_2(t) = u(t), y(t) = x_1(t) \). The input constraint is taken as \( \bar{u} = 1 \). In the performance index (15) we take \( N = 5, \ Q = I_2, \ R = 0.01, \) and \( P \) is chosen as the solution of the algebraic Riccati equation \( P = A^T PA + Q - K^T RK \), where \( K = R^{-1} B^T PA, \ R = R + B^T PB \). The state-space partition for this case, computed from Theorem 2.1, is shown in Figure 1. The resulting RHC law, com-
puted from (20), is

\[ k_5(x) = \begin{cases} 
-0.9653 & \text{if } x \in X_0 \\
-0.6154 & \text{if } x \in X_1 \\
-0.4390 & \text{if } x \in X_2 \\
-0.3399 & \text{if } x \in X_3 \\
-0.2771 & \text{if } x \in X_4 \\
-1 & \text{if } x \in X_5 
\end{cases} \]

and similar expressions in the remaining unlabeled regions, which can be obtained by symmetry. Note that, since \( X_5 \) is non convex, the final look-up table of the controller implementation requires several entries (corresponding to the convex subregions, indicated in dotted lines in Figure 1, whose union form \( X_5 \)) to represent it.

4 On-line Implementation

The algorithm described in the previous section yields a region partition having \( N_0 \) regions in the area of interest (typically \( N_0 < N_r \)). For example, for the double integrator (21) with \( N = 10 \), the total number of nonempty regions in the area \(-20 \leq x_1 \leq 20, -8 \leq x_2 \leq 8 \) is \( N_0 = 165 \). The “look-up table” implementation of RHC checks, at each time instant, in which region the current state is and picks the corresponding control law from a stored table. A simple Matlab code that implements this search in one step is shown below:

```matlab
function u=mpclaw(x,n,m,N0,regions,... gains,indices)
check=regions(:,1:n)*x-regions(:,n+1);
I2=cumsum(indices(1,:));
I1=[1 I2(1:N0-1)+1];
sel=zeros(sum(indices(1,:)),N0);
for k=1:N0,
    sel(I1(k):I2(k),k)=ones(indices(1,k),1);
end
where=(sign(check)+1)'*sel;
hee=find(where==0);
u=gains(m*(hee-1)+1:m*(hee-1)+m,1:n)*x+...
gains(m*(hee-1)+1:m*(hee-1)+m,n+1);
```

The variable \( \text{regions} \) stores the hyperplanes that delimit the regions of the partition, the variable \( \text{gains} \) stores the RHC laws corresponding to each region, and the variable \( \text{indices} \) stores the number of hyperplanes that delimit each region. For the double integrator example, the “size” of these variable is:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Size</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>regions</td>
<td>632 by 3</td>
<td>15168</td>
</tr>
<tr>
<td>gains</td>
<td>165 by 3</td>
<td>3960</td>
</tr>
<tr>
<td>indices</td>
<td>1 by 165</td>
<td>1320</td>
</tr>
<tr>
<td>total</td>
<td></td>
<td>20448</td>
</tr>
</tbody>
</table>

Note that the above algorithm for table look-up consists of just a few lines of code as opposed to hundreds for Matlab’s QP algorithm. To compare the “on-line” performance of both algorithms, we gridded the state space and, for each point of the grid, we evaluated the controller using each method, measured computation time and number of floating point operations, and then averaged the latter two measures over the grid.

Figure 2 is a plot of computation times for the “look-up table” method (solid line) and QP (dashed dotted line) as a function of the horizon \( N \). Figure 3 shows the number of floating point operations, as a function of \( N \), for both methods.

We observe that the performance of both algorithms is comparable in relation with computation speed, although the number of floating point operations is much larger for the “look-up table” method. This is expected since, as the horizon increases, there are more regions in the state space partition that have to be checked at each instant.

This problem can be easily mitigated by splitting the regions into a binary search tree; this increases the total number of regions (as some of the original regions are divided into smaller subregions) but the number of checks at each sample time is smaller and so there are less floating point operations.

To accomplish this task we divide the area of interest into the smaller subregions of a grid defined by \( N_h \)
separating hyperplanes parallel to the axes. The original RHC partition is then intersected with the grid to form a new partition. The maximum number of inequalities to check at each sample time is \( N_h + 180 = 184 \). The stored data use 44480 bytes.

Next, we grid the area of interest using five horizontal lines and seven vertical lines (i.e., \( N_h = 12 \)), defining a grid with 48 subregions, as shown in Figure 5. The intersection of the original RHC partition with the grid yields a total of 672 new regions, but the maximum number of inequalities to check at each sample time is \( N_h + 96 = 108 \). The data stored in this case use 164352 bytes.

The inclusion of the binary search tree in the algorithm for table look-up does not add too much complexity to the Matlab code, which has approximately 30 lines. To compare the “on-line” performance of the new table look-up algorithm and QP, we proceed as before, averaging computation time and number of floating point operations over a state space grid of 3600 points. In all cases, the binary
search tree has $N_h = 12$ separating hyperplanes.

Figure 6 is a plot of computation times for the “binary search look-up table” method (solid) and QP (dashed-dotted) as a function of the horizon $N$. Figure 7 shows the number of floating point operations, as a function of $N$, for both methods.

![Figure 6: Computation time for binary search look-up table (solid) and QP (dashed-dotted)](image)

![Figure 7: Number of floating point operations (flops) for binary search look-up table (solid) and QP (dashed-dotted)](image)

We can see from these figures that the new algorithm for table look-up using binary search has much better performance than the standard QP algorithm. Also, both measures (computation time and number of floating point operations) stay approximately constant for all horizons.

5 Conclusions

We have compared numerical properties of the table look-up implementation of a finitely parametrised RHC solution with traditional (active constraint) quadratic programming (QP) for on-line optimisation. We have found through examples that the table look-up implementation performs much better than QP in terms of computation time and number of floating point operations. In addition, the table look-up algorithm can be very simply coded. The simplicity of the code, together with the availability of a complete off-line characterisation of the RHC law allows verification of the full spectrum of behaviour prior to implementation. This latter property of the table look-up implementation is of independent interest since it may enhance the acceptance of these schemes in industry by circumventing the vagaries inherent in the current reliance on on-line implementation.

References


