Parallel solution of model predictive control using the alternating direction multiplier method

Markus Kögel∗,**,1 Rolf Findeisen∗

* Laboratory for Systems Theory and Automatic Control, Otto-von-Guericke-University Magdeburg, Germany. e-mail: {markus.koegel, rolf.findeisen}@ovgu.de.
** Max Planck Institute for Dynamics of Complex Technical Systems Magdeburg, Germany.

Abstract: In this work a parallel solution method for model predictive control is presented based on the alternating direction multiplier method. Our approach solves the overall problem by decomposing it along the time-axis into smaller subproblem. The resulting subproblems can be solved efficiently in parallel and are coupled by the multiplier update. We specifically considers box constraints, since this allows to use tailored methods to solve the subproblems. We outline the applicability of the approach and the performance using a multi-core systems.

Keywords: Model predictive control, alternating direction method of multipliers.

1. INTRODUCTION

Model predictive control (MPC) uses as feedback the first part of the input sequence obtained by solving an optimal control problem. On the one hand this allows direct consideration of constraints and enables high control performance. One the other hand the optimal control problem needs to be solved in each sampling instance. Even tough many dedicated solution methods exists the computational burden of MPC for linear systems is still often demanding, especially for fast, complex problems. Therefore, we outline in this work an approach to utilize nowadays available parallel computation hardware to enable the parallel solution of the MPC problem. In particular we exploit the structure of the problem by splitting the problem along the time-axis into subproblems, which can be solved in parallel. Specifically we focus on MPC of linear, discrete-time systems with polytopic constraints and a quadratic performance index.

Different concepts for parallel, online solution of MPC problems have been studied. It is possible to parallelize the (linear) algebra of specific algorithms, which is often considered for implementations on FPGAs (field programmable gate arrays) or on GPUs (graphics processing units). Parallelizing the solutions of systems of linear equations appearing in interior-point methods is studied in Constantinides (2009); Gu et al. (2010); Jerez et al. (2011); Wills et al. (2011). The works Gu et al. (2009); Knagge et al. (2009) consider parallel solution approaches for active set methods.

Furthermore, it is possible to solve in parallel several optimization problems. One can evaluate in parallel different input parameterizations and select the one with the best performance, e.g. using parallel multiplexing schemes (Jerez et al. (2011)), channel-hopping (Ling et al. (2011)) or parallel move blocking (Longo et al. (2011)). The work Behrendt et al. (2011) studies a parallel solution using an active set method starting from multiple initial guesses.

Parallel/distributed optimization algorithms for the solution of large optimization problem exists, that solve iteratively in parallel smaller problems, compare Bertsekas and Tsitsiklis (1989). Such optimization techniques are often employed in so-called (cooperative) distributed MPC approaches, where the overall plant is basically decomposed into smaller coupled subplants and dedicated algorithms are used to allow a distributed solution and guarantee the equivalence of the resulting solution. Different optimization techniques have been studied in this framework, for example dual decomposition in (Giselsson and Rantzer (2010); Wakasa et al. (2008)) or Jacobi algorithms in (Stewart et al. (2010)). In our previous work, Kögel and Findeisen (2012), we used the alternating direction method of multipliers for that framework.

In this work we propose a different approach; instead of decomposing the problem based on the plant structure into smaller subplants, we decompose the problem along the time axis into smaller subproblems using the alternating direction method of multipliers. We specifically discuss parallel implementation issues and present a suitable method to solve the subproblems for box-constrained problems. A multi-core processor implementation outlines the effectiveness of the approach by simulation studies. However, the approach is not limited to multi-core processors, one might implement it also on e.g. multi-core microcontroller, FPGAs, GPUs, etc.

The structure of the remainder of the paper is as follows. First we state the problem in Section 2 and review the
alternating direction method of multipliers in Section 3. In
Section 4 we outline the proposed algorithm and discuss
implementation issues. Section 5 outlines the solution of
the subproblems for the case of box-constraints. In Section
6, we illustrate the performance of the algorithm.

All norms are Euclidean norms. In algorithms \(x(i)\) or \(x[i]\)
denotes \(x\) with respect to the iteration index \(i\).

2. PROBLEM STATEMENT

We consider plants of the form

\[
x_{k+1} = Ax_k + Bu_k + d_k, \tag{1}
\]

where \(x_k \in \mathbb{R}^n\) is the state, \(u_k \in \mathbb{R}^p\) the input and \(d_k\)
a known disturbance or exogenous input. The state and
input of the plant are required to satisfy the constraints

\[
C x_k + D u_k + f_k \leq 0, \tag{2}
\]

where \(\leq\) holds entry-wise and \(f_k \in \mathbb{R}^m\).

We consider model predictive control of the system (1)
subject to the constraints (2) and a quadratic cost crite-

\[
\begin{aligned}
J & = \sum_{k=0}^{N-1} x_k^T L x_k + \sum_{j=0}^{N} u_j^T F_j u_j + \sum_{i=1}^{N} \gamma_i \left( \sum_{j=0}^{N} \xi_{ij} y_j(t) - s_i \right)
\end{aligned}
\]

Clearly, \(x, \xi, \gamma\) need to satisfy the constraints (2):

\[
C x_j + D \xi_j + f_j \leq 0, \quad j = k, \ldots, k + N - 1. \tag{4}
\]

Additionally a consistent prediction requires that the op-
timization variables \(\xi, \gamma\) satisfy the system equations

\[
\begin{align*}
\pi_j &= \pi_{j+1} - A \pi_j + D \eta_j, & j = k, \ldots, k + N - 1 \tag{5a} \\
\pi_k &= x_k. \tag{5b}
\end{align*}
\]

With respect to the cost we consider a convex cost \(J_k\),
which consists of a stage costs \(J_j\) and the final cost \(J_f^k\):

\[
J_k(x, \xi) = J_f^k(\pi_{k+N}) + \sum_{j=k}^{k+N-1} J_j(\pi_j, \eta_j) \tag{6a}
\]

\[
J_f^k(\pi_{k+N}) = \frac{1}{2} \pi_{k+N}^T T \pi_{k+N} + \sum_{j=k}^{k+N-1} l_j^T \pi_j \tag{6b}
\]

\[
L = L^T = \begin{bmatrix} Q & S^T & R \end{bmatrix} > 0, \quad l_j = \begin{bmatrix} q_j \end{bmatrix} \tag{6c}
\]

Note that \(J_k\) contains linear and quadratic weightings.
Additionally, the terminal state is constrained by

\[
C^T \pi_{k+N} + f_{k+N}^T \leq 0, \tag{7}
\]

where \(f_{k+N}^T \in \mathbb{R}^m\).

It is easy to see, that the resulting optimal control problem
is a quadratic program (QP) given by

\[
\min_{x, \xi, \gamma} J_k \quad \text{subject to} \quad \pi_k = x_k, \tag{8}
\]

We present in Section 4 and 5 a parallel solution method
for this QP: splitting the problem along the time axis
into smaller subproblems using the alternating direction
method of multipliers, which we review next.

3. REVIEW: ALTERNATING DIRECTIONS
METHOD OF MULTIPLIERS

The proposed parallel solution scheme is based on a variant
of the alternating direction method of multipliers (ADM
method). We briefly review this variant for a general
problem, for more details see Bertsekas and Tsitsiklis
(1989); Boyd et al. (2010) and the references therein.

Consider a general optimization problem of the form

\[
\min_{y} \sum_{j=0}^{\alpha} F_j(y_j) = \min_{y} F(y) \tag{9a}
\]

subject to

\[
y_j \in P_j, \quad j = 1, \ldots, \alpha, \tag{9b}
\]

\[
\sum_{j=1}^{\alpha} e_{ij} y_j = s_i, \quad i = 1, \ldots, \beta, \tag{9c}
\]

where the vector of optimization variables \(y\) consist of \(\alpha
\)
sub-vectors \(y_j\)

\[
y = (y_1, y_2, \ldots, y_{\alpha}), \tag{10}
\]

and where the costs \(F_j\) are strictly convex, \(P_j\) are
closed, convex, nonempty polytopes and \(\sum_{j=1}^{\alpha}
\)
\(e_{ij} y_j = s_i\) are \(\beta\) equality constraints. Only the equality constraints
\(\sum_{j=1}^{\alpha} e_{ij} y_j = s_i\) couple different sub-vectors
\(y_j\) with each other. Since an equality might not depend on
every \(y_j\), we denote by \(\xi(i)\) the sub-vectors \(y_j\) in constraint
\(i\) and by \(\gamma_i\) the number of sub-vectors \(y_j\) appearing in
constraint \(i\) (the cardinal number of \(\xi(i)\)).

The ADM method for the optimization problem (9) is

\[
\tilde{y}_j(t + 1) = \arg \min_{y_j \in P_j} \tilde{F}_j(y_j) \tag{12a}
\]

\[
\tilde{F}_j(y_j) = F_j(y_j) + \sum_{i \in \xi(j)} \lambda_i(t) e_{ij} y_j \tag{12b}
\]

\[
\quad + \frac{c}{2} \sum_{i \in \xi(j)} (e_{ij}(y_j(t) - \tilde{y}_j) + w_i(t))^2 \tag{12c}
\]

\[
\lambda_i(t + 1) = \lambda_i(t) + c e_{ij}(y_j(t) - \tilde{y}_j) \tag{12d}
\]

where \(\lambda_i\) denotes multipliers, \(w_i\) normalized errors, \(c > 0\)
is the so-called penalty parameter and \(j = 1, \ldots, \alpha, i = 1, \ldots, \beta\).

The alternating direction multiplier method solves (9) iteratively using an initial guess \{\(\lambda_0(0)\), \(w_0(0)\}\).

Fortunately, the \(\alpha\) optimization problems in (12a)
are independent of each other, i.e. can be solved in parallel.
In this paper we refer to these problem as subproblems
and to (12c), (12d) as multiplier update.

Remark 1. (Convergence guarantee of ADM method)

If (9) is feasible, then \{\(\tilde{y}_j\)\} converges to the optimal
solution for any \(c > 0\) and each subproblem is feasible,
cf. Bertsekas and Tsitsiklis (1989); Boyd et al. (2010).

Although convergence is guaranteed for any value of \(c\),
the actual value influences the convergence rate and thus
the speed of the method. In addition, the choice of \(c\) also
influences the effort required for the minimizations (12a).

Remark 2. (Partial elimination of constraints)

It is not necessary to handle every equality constraint
(9c) by multipliers. One can use multipliers only for some constraints and handle the remaining constraints explicitly in step (12a) by including them into $P_j$.

4. PARALLEL MPC USING THE ADM METHOD

Now we apply the ADM method to the MPC optimization problem (8). Note that the state and input constraints (4), (7) as well as the cost (6) are separable. Thus we propose to use for every equality constraint due to the dynamics (1) a multiplier, i.e., a decomposition of problem (8) along the time axis into $N + 1$ subproblems.

Next we outline the ADM algorithm for this approach, discuss implementation issues and a possible variation, which allows to split the overall problem into less subproblems.

4.1 Resulting subproblems and multiplier update

To allow a simple presentation we divide the optimization variables $\mathbf{u}$ and $\mathbf{x}$ (3) into $N + 1$ sets $z_j$ given by

$$z_j = \frac{\mathbf{u}_{k+j}}{\mathbf{u}_{k+j}} , j = 0, \ldots, N - 1, \quad z_N = \mathbf{u}_{k+N} .$$

The ADM method applied to this setup consists of $N + 1$ subproblems $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_N$ and the multiplier update.

The subproblems $\mathcal{T}_j$, $j = 1, \ldots, N - 1$ are given by

$$\min \tilde{J}_{j+k}, \quad s.t. \ (C \ D) z_j + f_{j+k} \leq 0 ,$$

where $\tilde{J}_{j+k}$ is the augmented cost functions

$$\tilde{J}_{j+k} = J_{j+k} + \lambda_1(i)^T M_{j,j} z_j + \lambda_{j+1}(i)^T M_{j+1,j} z_j + \frac{c}{2} \| M_{j+1,j}(z_j - \hat{z}_j(i)) + w_{j+1}(i) \|^2$$

$$+ \frac{\hat{c}}{2} \| M_{j,j}(z_j - \hat{z}_j(i)) + w_j(i) \|^2 ,$$

where $M_{j,j} = (I \ 0)$, $M_{j+1,j} = -(A \ B)$, and $\hat{z}_j(i)$ is the minimizer of the previous iteration $i$, $c$ is the penalty parameter and $\lambda_1(i)$, $w_j(i)$ are the multiplier and the term corresponding to the (vector) equality constraint.\textsuperscript{3}

$$z_{j+k+1} - A z_j - B z_{j+k} - d_{j+k} = 0 . \quad (15)$$

The first subproblem $\mathcal{T}_0$ and the last subproblem $\mathcal{T}_N$ are slightly different; they can be found in the Appendix A. Note that all of the subproblems $\mathcal{T}_0, \ldots, \mathcal{T}_N$ are QPs. The multiplier update uses the optimal solution $\{ \hat{z}_j(i+1) \}$ of the subproblems $\mathcal{T}_j$ to update the multipliers $\lambda_i(i+1)$ and errors $\{ w_j(i+1) \}$. Assuming that no row of $(A \ B)$ contains only zeros, the update is given by

$$w_j(i+1) = \frac{1}{2} M_{j,j-1} \hat{z}_{j-1}(i+1) + d_{j+k-1}$$

$$- \frac{1}{2} M_{j,j} \hat{z}_j(i+1)$$

$$\lambda_{j+1}(i+1) = \lambda_j(i) + c w_j(i+1) ,$$

where $j = 1, \ldots, N$. The overall algorithm becomes:

\begin{algorithm}
\caption{Parallel MPC with the ADM method}
\begin{algorithmic}
\Require Initial guesses $\hat{z}(0), w(0), \lambda(0)$; penalty parameter $c$, maximum number of iterations $i^{\text{Max}}$, problem data $A, B, \ldots$.
\For{$i = 0, \ldots, i^{\text{Max}}$}
\State Solve subproblems $\mathcal{T}_0, \ldots, \mathcal{T}_N$ (A.1), (14), (A.3) in parallel using initial guess based on $\{ \hat{z}_j(i) \}$.
\State Update multipliers $\{ \lambda_j(i+1) \}$ and errors $\{ w_j(i+1) \}$ (16).
\EndFor
\State Return $\hat{z}(i^{\text{Max}}), w(i^{\text{Max}}), \lambda(i^{\text{Max}})$
\end{algorithmic}
\end{algorithm}

4.2 Implementation of the algorithm

We discuss now certain implementation details such as parallel implementation of the algorithm and warm-starting. Note that we do not focus on a special hardware structure.

Parallel implementation

Depending on the number $U$ of available computing entities and the horizon length $N$, there are different possibilities to assign the subproblems to the units.

First, if $U = N + 1$, then each entity $U_0, \ldots, U_N$ solves one subproblem; we assume that $U_j$ solves $\mathcal{T}_j$. The multiplier update can be parallelized as follows: entity $U_j$ sends $M_{j,j} z_j$ to $U_{j-1}$, $M_{j+1,j} z_j$ to $U_{j+1}$ and receives similar data from them. This allows that each $U_j$ can compute $w_j, w_{j+1}$ and $\lambda_j, \lambda_{j+1}$ on its own.\textsuperscript{4} Note that only local synchronization and communication, i.e., between neighboring computing entities are necessary.

If one has $U < N + 1$, then each (or only some) computation unit need to solve more than one subproblem. Assigning neighboring problems to the same subproblem helps to reduce the communication effort. Below (Section 4.3) we discuss for this case a variation of the algorithm to split (8) in less subproblems.

In the case $U > N + 1$, one can parallelize the solution of each/some subproblem. Also a combination of this time-axis decomposition approach with a decomposition into subsystems (as in distributed MPC) might be considered.

Warm-starting

One can warm start Algorithm 1 by using the solution of time step $k$ ($\hat{z}(i^{\text{Max}})k), w(i^{\text{Max}})k), \lambda(i^{\text{Max}})k)$ to obtain initial guesses for the next time step ($\hat{z}(0k+1), w(0k+1), \lambda(0k+1))$. A straightforward choice of initial guesses is to shift the previous solution and to append zeros at the end (also other methods are possible):

$$z_j(0k+1) = \hat{z}_{j+1}(i^{\text{Max}})k) \quad z_N(0k+1) = 0 \quad (17a)$$

$$w_j(0k+1) = w_{j+1}(i^{\text{Max}})k) \quad w_N(0k+1) = 0 \quad (17b)$$

$$\lambda_j(0k+1) = \lambda_{j+1}(i^{\text{Max}})k) \quad \lambda_N(0k+1) = 0 . \quad (17c)$$

This delivers often better performance/faster convergence than cold starting (initializing all variables with zeros).

Solution of the subproblems

The subproblems $\mathcal{T}_0, \ldots, \mathcal{T}_N$ are quadratic programs with inequality constraints, compare (A.2), (14), (A.3). Their cost functions are of the form

\textsuperscript{2} A detailed derivation of these subproblems (and also the multiplier update) is avoided here due to space limitations.

\textsuperscript{3} The vector form of the equalities (15) allows to rewrite the sums in (12) as scalar products and norms in $J_{j+k}$.

\textsuperscript{4} For $U_0$ only $\lambda_1, w_1$ are computed and $M_{1,0}z_0$ is send to $U_1$. Similar exceptions hold for $U_N$. 

371
\[ J_j(i) = \frac{1}{2} z_j^T H_j z_j + z_j^T g_j(i), \]  
(18)

where the Hessian matrix \( H_j \) is constant. For the subproblems \( T_1, \ldots, T_{N-1} \), \( H_j \) is

\[ H_j = L + \frac{c}{2} \left( I + A^T A + B^T B \right). \]  
(19)

The linear cost term \( g_j(i) \) depends on error terms \( w(i) \) and multipliers \( \lambda(i) \), the linear noise \( \epsilon_0 \) and the disturbance \( d_k \) (and for \( f_0 \) (A.2) on the measured state \( x_k \)).

Here exist many optimization methods to solve such problems. In Section 5 we discuss the solution using Nesterov’s method for the case of box-constraints.

4.3 Decomposing the problem in fewer subproblems

So far we handle all equality constraint by a multipliers, which directly results in \( N+1 \) subproblems. However often much less computation entities than subproblems are available \((N + 1 > U)\), therefore it might be advantageous to obtain fewer (e.g. exactly \( U \)), but more complex subproblems. Using multipliers only for some equality constraints results in fewer subproblems, see Remark 2.

For illustration consider that we have \( U = 2 \) computing entities and \( N = 3 \). Then using only multipliers for the equality constraint coupling \( z_1 \) with \( z_2 \) results in two subproblems \( P_0 \) and \( P_1 \). The first subproblem has as optimization variables \( y_0 = (z_0^T, z_1^T)^T \), contains the equality constraints (1) coupling \( z_0 \) with \( z_1 \). In a nutshell it replaces the subproblems \( T_0, T_1 \) and their coupling. The larger problem \( P_0 \) is given by

\[ \min_{y_0} \sum_{m=k}^{k+1} J_m + \langle 0, \tilde{\lambda}_1(i-1)^T \rangle \]  
(20)

\[ + \frac{c}{2} W_{1,0} W_{1,0} (y_0 - \tilde{y}_0 (i-1)) + \tilde{w}_1 (i-1)^2, \]

subject to

\[ C x_{k+m} + D \xi_{k+m} + \tilde{f}_{k+m} \leq 0, \quad m = 0, 1, \]
\[ \xi_{k+m+1} = A \xi_{k+m} + B \xi_{k+m} + d_{k+m}, \quad m = 0, 1, \]
\[ \tilde{x}_k = x_k. \]

where \( W_{1,0} = - (0 A B) \).

Solving this larger problem \( P_0 \) usually requires more computations than solving the smaller problems \( T_0, T_1 \), but the overall convergence of the multiplier method might improve. So this idea could improve the overall speed. Note that one can exploit the structure of the subproblems similar as in usual MPC problems.

Unfortunately, warm-starting the overall algorithm is no longer straightforward, since we cannot simply use (17). For the above example at step \( k \) we compute via the ADM method the multiplier corresponding to the coupling of \( \xi_{k+1}, \xi_{k+2} \) with \( \xi_{k+3} \), but in the next step we compute the multiplier corresponding to the coupling of \( \xi_{k+2}, \xi_{k+3} \) with \( \xi_{k+4} \), which we did not determine. Note that the optimization methods solving the subproblems might calculate multipliers, which could be used as initial guesses.

5. SPECIAL CASE: BOX CONSTRAINTS

In this section we consider the special case that the constraints are box constraints, i.e. \( 2 \) corresponds to

\[ -\infty \leq x_k^b \leq x_k \leq x_k^u \leq \infty \]
(21a)

\[ -\infty \leq u_k^b \leq u_k \leq u_k^u \leq \infty \]
(21b)

\[ -\infty \leq x_{k+N}^b \leq x_{k+N} \leq x_{k+N}^u \leq \infty. \]
(21c)

Note that not all states \( x \) inputs need to be bounded.

For this case there exists special optimization methods to solve the subproblems. Here we focus on Nesterov’s gradient method (Nesterov (2004)), since it delivers often good performance and is easy to implement.

5.1 Solution of subproblems by Nesterov’s gradient method

Nesterov’s gradient method solves a problem \( T_j(i) \) iteratively from an initial guess \( z_{i[0]} = z_{i[0]}(0) \) using the iteration

\[ z_{ij}(a + 1) \]  
(21a)

\[ z_{ij}(a + 1) = \max \left( z_{ij}^b, \min \left( z_{ij}^u, x_{ij}(a + 1) \right) \right), \]
(21b)

\[ z_{ij}(a + 1) = \max \left( z_{ij}^b, \min \left( z_{ij}^u, -x_{ij}(a + 1) \right) \right), \]
(21c)

and returns \( z_{ij}(i) = z_{ij}[M_{\text{Max}}] \) as (approximate) solution of \( T_j(i) \). In (22) \( a_{ij}[M_{\text{Max}}] \geq a > 0 \) denotes the iteration index of Nesterov’s method, \( g_j(i) \) is as in (18), \( \kappa_j \) is the maximum eigenvalue of \( H_j \), \( z_{ij}^b \) \( z_{ij}^u \) correspond to the constraints of \( T_j \) induced by (21). The constant \( \rho_j \) is \( \rho_j = \frac{\sqrt{\kappa_j^2 - \sqrt{\kappa_j^2 + 4 \rho_j^2}}}{\sqrt{\kappa_j^2 + 4 \rho_j^2}} \)

where \( \mu_j \) is the smallest eigenvalue of \( H_j \).

Although the bounds are extended real-values, the saturation (22b) is well defined and can be implemented. Note that Nesterov’s method is rather simple to implement. Warm-starting is possible using e.g. \( z_{ij}[0] = \tilde{z}_j(i-1) \).

Remark 3. (Convergence of Nesterov’s method)

The (guaranteed) convergence rate of Nesterov’s method depends on the condition number of the matrix \( H_j \) (19) \( \frac{\kappa}{\mu} \)

compare Nesterov (2004). Thus the value of \( c \) influences the convergence of Nesterov’s method, compare (19), and also the convergence of the ADM method. One needs to tune \( c \) to achieve a good overall performance.

5.2 Memory demand and computational demand

For this special case we discuss in the following the computational demand and the memory demand of the overall algorithm. We assume that not more than \( N + 1 \) parallel computation entities are used \((N + 1 > U)\) and each subproblem is solved using a fixed, common maximum number of iterations \( a_{ij}[M_{\text{Max}}] = a_{ij}[M_{\text{Max}}, \forall j\) of Nesterov’s method.

We outline first the total demand. Clearly the demand per computing entity is smaller; each entity needs to store only the data necessary to solve the subproblem(s) and only perform a part of the computations.

We need to store the dynamic \((1)\), cost function \((6)\), Hessian matrices \((19)\). Since the appearing matrices are time-invariant the memory demand is \( O(N(n + p) + U(n + p)^2) \).
Moreover, we need to store the optimization variables \( \{\hat{z}_j\} \), multipliers \( \{\lambda_j\} \) and errors \( \{w_j\} \), the demand is \( O(N(n+p)) \). In addition, to evaluate (22) each computation unit needs additional memory to storage \( g_j(i), \zeta_j, \tilde{\zeta}_j \) and an old version of \( \zeta_j \) for (22c). In summary, the total memory demand is in the order of \( O(N(n+p) + U(n+p)^2) \). The computational effort of the multipliers update is in the order of \( O(Nn) \).

Note that memory demand and time complexity per iteration of the overall algorithm are linear in \( N \). Moreover, assuming an equal distribution of the subproblems to the entities, the computational and memory effort per computation entity is the total demand divided by about \( U \), i.e. decreases as expected if more entities are used.

5.3 Remark on Stability

We do not focus on deriving stability results for the outlined scheme in this paper. However, it is possible to derive such results considering terminal box-constraints and a periodically varying horizon, we refer for first results to Kögel and Findeisen (2012).

Also, it is well known that ellipsoidal terminal constraints

\[
\|C_{qc}^{\top}x_{k+N}\|_2^2 \leq f',
\]

(23)
can guarantee stability and recursive feasibility under certain assumptions, see e.g. Chen and Allgöwer (1998). If \( C_{qc}^{\top} \) is invertible, using \( \tilde{x}_{k+N} = C_{qc}^{\top}x_{k+N} \) the general quadratic constraint (23) reduces to a norm constraint

\[
\|\tilde{x}_{k+N}\|_2^2 \leq f',
\]

(24)
which can easily be treated by Nesterov’s method in contrast to (23). The above transformation can be included into the algorithm, which changes the data of subproblem \( T_N \). We now have \( M_{N,N} = C_{qc}^{\top}, \tilde{T} = (C_{qc}^{\top})^{-T}T(C_{qc}^{\top})^{-1}, \tilde{t}_k = (C_{qc}^{\top})^{-T}t_k \) and (24) instead of (21c).

Unfortunately, using the norm constraint the convergence of the ADM method might not be guaranteed in this case.

6. SIMULATION EXAMPLE

We use as an example, a chain of masses as illustrated in Figure 1. The six masses are connected by springs to each others and to walls. Actuators can apply a force to each mass. We assume that the actuators have a maximum force and a limited slew rate. We assume the spring constants as 1, the inertias of the masses as 1. There is no damping.

We consider the forces \( F_I, \ldots, F_{VI} \) as (integrating) states and the slew rates \( \Delta F_I, \ldots, \Delta F_{VI} \) as inputs. Thus the overall system has \( n = 18 \) states and \( p = 6 \) inputs. It is stable, but not asymptotically. It is discretized by a zero order hold and a sampling time of 0.5.

The limitations of the actuators are

\[
-0.5 \leq \Delta F_i \leq 0.5, \quad 1 \leq F_i \leq 1,
\]

(25)
where \( i = I, II, \ldots, VI \). We choose the weighting matrices \( Q, R \) and \( T \) as identities and use a rather large horizon of \( N = 35 \). The linear weightings \( q_k, r_k, t_k \) and the disturbance/exogenous input \( d_k \) are assumed to be zero.

We want to evaluate the speed and accuracy of the proposed algorithm. We estimate the accuracy using Monte Carlo simulations by comparing the obtained cost using inexact solution and exact solution of (8). We use 20 Monte Carlo simulations each 6000 steps long and accumulate the cost for the last 5000 steps. At the beginning of each simulation the masses are at rest at a randomly chosen initial position (uniform distribution \([-5, 5]\)) and the actuators apply no force. In each step random forces uniformly distributed between \(-2\) and \(2\) act on each mass.

The MPC problem is always feasible, since there are no constraints on the position or on the speed of the masses. So it is possible to add rather large disturbances resulting in many active constraints without risking infeasibility.

We choose \( c = 8 \), use \( \lambda_{\text{Max}} = 50 \) ADM iterations and \( a_{\text{Max}} = 5 \) iterations of Nesterov’s method.

In Figure 2 we illustrate a sample path. Note that both types of constraints (25) are sometimes active and that the difference between exact and inexact solution is small.

We evaluate the computation speed on two CPUs. We consider an Intel Core 2 Quad Q6600 CPU with 2.4 GHz and four cores and an Intel Xeon X5675 CPU with six cores, which has a standard frequency of 3.06 GHz (applicable if all six cores are used) and a turbo frequency of 3.46 GHz if a single core is used. The algorithm is implemented in C++; the message passing interface openMPI (Gabriel et al. (2004)) is used to implement the parallel computation and the Eigen template library (Guennebaud et al. (2010)) for the linear algebra computations.

The computation of the MPC problem requires about 13 ms on the X5675 and 20 ms on the Q6600 using a single core. Parallel computation reduces the time by a factor of about 4.3 and 3.1 to 2.9 ms and 6.4 ms, respectively.

Table 1 compares the performance of our algorithm with the active set method qpOASES (Ferreau et al. (2008)) and an interior point method (Wang and Boyd (2010)). To limit the maximum computation time of qpOASES, we bound the working set changes to 35. For the algorithm Wang and Boyd (2010) we set the maximum number of Newton iterations to 15 and the barrier parameter (after trying different values) to \( 10^{-2} \). Since this algorithm solves only problems with box-constraints, we limit the unconstrained states to \( \pm 10^{16} \) (dummy constraints).

For this example the proposed algorithm exhibits a better performance than the other algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>( J_{\text{avg}} )</th>
<th>( T_{Q6600} )</th>
<th>( T_{X5675} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed algorithm (single core)</td>
<td>0.18%</td>
<td>20ms</td>
<td>13ms</td>
</tr>
<tr>
<td>Proposed algorithm (all cores)</td>
<td>0.18%</td>
<td>6.4ms</td>
<td>2.9ms</td>
</tr>
<tr>
<td>qpOASES</td>
<td>0.41%</td>
<td>29ms</td>
<td>19ms</td>
</tr>
<tr>
<td>Wang and Boyd (2010)</td>
<td>0.23%</td>
<td>22ms</td>
<td>14ms</td>
</tr>
</tbody>
</table>

Table 1. Performance for benchmark example.

\( T_{Q6600} / TX5675 \) maximum computation time, \( \Delta J_{\text{avg}} \) cost difference / suboptimality index.
Fig. 1. Benchmark example: Masses connected by springs. \( F_I \) - force of actuator I

Fig. 2. Sample path. Blue: Exact solution. Black: Inexact solution. Red: Actuator I

7. SUMMARY

In this paper we outlined how the alternating directions multipliers method can be used for fast, parallel solution of MPC problems by temporal decomposition.

REFERENCES


