An Optimized Linear Model Predictive Control Solver

Dimitar Dimitrov, Pierre-Brice Wieber, Olivier Stasse, Hans Joachim Ferreau, Holger Diedam

An Optimized Linear Model Predictive Control Solver

Dimitar Dimitrov[1], Pierre-Brice Wieber[2], Olivier Stasse[3], Hans Joachim Ferreau[4], and Holger Diedam[5]

[1] Örebro University - Sweden, mitko@roboresearch.net
[2] INRIA Grenoble - France pierre-brice.wieber@inria.fr
[3] JRL - Japan olivier.stasse@aist.go.jp
[4] KU Leuven - Belgium joachim.ferreau@esat.kuleuven.be
[5] Heidelberg University - Germany hdiedam@ix.urz.uni-heidelberg.de

Summary. This article addresses the fast on-line solution of a sequence of quadratic programs underlying a linear model predictive control scheme. We introduce an algorithm which is tailored to efficiently handle small to medium sized problems with relatively small number of active constraints. Different aspects of the algorithm are examined and its computational complexity is presented. Finally, we discuss a modification of the presented algorithm that produces "good" approximate solutions faster.

1 INTRODUCTION

Model Predictive Control (MPC) is an advanced control tool that originates in the late seventies. Due to its simplicity, it quickly became the preferred control tool in many industrial applications [1]. Some of the fields where MPC is already considered to be a mature technique involve linear and rather slow systems like the ones usually encountered in the chemical process industry. However, the application of MPC to more complex systems, involving nonlinear, hybrid, or very fast processes is still in its infancy.

MPC does not designate a specific control strategy but rather an ample range of control methods which use a model of a process to obtain control actions by minimizing an objective function, possibly subject to given constraints. The various algorithms in the MPC family can be distinguished mainly by: (i) the model used to represent the process; (ii) the objective function to be minimized; (iii) the type of constraints. The most popular scheme applied in practice involves a linear time-invariant process model, linear constraints and quadratic objective function [2]. In general, it is referred to as linear MPC (LMPC). The computational burden associated with the application of LMPC is mainly due to forming and solving a Quadratic Program

(QP) at each sampling interval. This imposes restrictions on the application of LMPC to systems that require short sampling times.

In practice, the solution of the underlying sequence of QPs is left to state of the art QP solvers [3]. Even though such solvers implement very efficient algorithms, in most cases they do not make use of the properties of each particular problem, which could speed up computations considerably.

In this article we present an algorithm for the fast on-line solution of a sequence of QPs in the context of LMPC. When the control sampling times become so small, that classical methods fail to reach a solution (within a given sampling interval), algorithms that can exploit the particular structure of LMPC problems become attractive. The proposed algorithm is tailored to efficiently utilize data that can be precomputed off-line, leading to smaller on-line computational burden. We assume that the problem to be solved is small to medium sized, with relatively small[6] number of active constraints.

The presented algorithm can be classified as a *primal active set method* with *range space linear algebra*. We motivate our choice by analyzing the requirements of our problem. Different aspects of the algorithm are examined, and its computational complexity is presented. We discuss details related to efficient update methods for the solution of the underlying systems of linear equations. Finally, we present a strategy for altering the *working set* resulting in a "good" approximate solutions that can be computed faster.

2 Linear Model Predictive Control

There is a great variety of models commonly used in the context of MPC. In general, they can be divided into two groups: (i) *first principles models* and (ii) *identified models*. The former are based on physical or chemical laws of nature, whereas the latter are built as a result of empirical measurements of the real process. Here, we assume that regardless of the way the model is obtained, it is represented in the following form

$$x_{k+1} = A\,x_k + B\,u_k, \tag{1a}$$

$$y_k = C\,x_k \tag{1b}$$

where, $x_k \in R^{n_x}$ represents the state of the system, $u_k \in R^{n_u}$ is control input, $y_k \in R^{n_y}$ is a vector of measured outputs which are to be controlled (to satisfy given constraints and when possible to follow certain reference profile), and A, B, C are constant matrices with appropriate dimensions.

In order to express the behavior of system (1) for N discrete steps in the future as a function of x_k and $n = Nn_u$ control actions, equation (1a) is iterated N times (combined with N versions of (1b)) as follows

[6] The term "relatively small" will be properly defined in Section 3.

$$y_{k+\tau} = CA^{\tau}x_k + C\sum_{\rho=0}^{\tau-1} A^{(\tau-\rho-1)}Bu_{k+\rho}\,, \quad (\tau = 1, ..., N). \tag{2}$$

Using the notation

$$Y_{k+1} = \begin{bmatrix} y_{k+1} \\ \vdots \\ y_{k+N} \end{bmatrix} \in \mathbb{R}^{Nn_y}, \qquad U = \begin{bmatrix} u_k \\ \vdots \\ u_{k+N-1} \end{bmatrix} \in \mathbb{R}^{n}$$

recursion (2) can be expressed in the following compact way

$$Y_{k+1} = P_x\, x_k + P_u\, U \tag{3}$$

where, $P_x \in \mathbb{R}^{Nn_y \times n_x}$ and $P_u \in \mathbb{R}^{Nn_y \times n}$ are constant matrices (independent of k).

MPC uses a process model in order to predict the process behavior starting at a given discrete time k, over a future *prediction horizon* $k + N$. Assuming that information about disturbances and state measurement noise is not available, the predicted behavior depends on the current state x_k and the assumed control input trajectory U that is to be applied over the prediction horizon. The idea is in step (i) to select U which leads to the "best" predicted behavior (according to a given objective function). Once U is obtained, in step (ii) only the first control action (u_k) is applied to the system until the next sampling instant. Then in step (iii) the new state x_{k+1} is measured (or estimated), and the process is repeated again from step (i). Hence, MPC is a feedback control strategy. In a standard LMPC scheme, the n future control actions U are computed to minimize given quadratic cost function [2]

$$\underset{U}{\text{minimize}} \;\; \frac{1}{2}U^T QU + U^T p_k \tag{4}$$

where, $Q \in R^{n\times n}$ is a symmetric and positive-definite constant Hessian matrix, and $p_k \in R^n$ is a gradient vector.

Furthermore, the profile of the outputs Y_{k+1} are possibly constrained to satisfy a set of m linear constraints of the form

$$D_{k+1}Y_{k+1} \leq b'_{k+1}, \tag{5}$$

for some matrix $D_{k+1} \in R^{m\times Nn_y}$ and vector $b'_{k+1} \in R^m$. If the i^{th} row of D_{k+1} $(i = 1, \ldots, m)$ imposes constraints only on y_{k+i} (which is very common in practice), D_{k+1} will be extremely sparse and well structured matrix with at most n_y nonzero entries in each row. Hereafter, we assume that D_{k+1} has such structure. Introducing (3) in (5) leads to

$$G_{k+1}U \leq b_{k+1} \tag{6}$$

where, $G_{k+1} = D_{k+1}P_u$, and $b_{k+1} = b'_{k+1} - D_{k+1}P_x x_k$. Additional constraints accounting for actuator limits etc. could be imposed.

The objective function (4) in combination with the constraints (6) define a canonical optimization problem known as quadratic program. Its solution is required for the application of a LMPC scheme.

3 General design choices for a QP solver

The choice of algorithm that can efficiently solve a sequence of quadratic programs defined by (4) and (6) is not unique. In general, the choice depends mostly on: (i) the number m_a of active constraints (constraints that hold as equalities at the optimal point) and the dimension N; (ii) whether a "warm start" is available; (iii) whether there is a cheap way to determine an initial feasible point that satisfies the constraints in (6). The following short overview aims at outlining some of the considerations that need to be made when choosing a QP solver.

3.1 Interior point vs. active set methods

Fast and reliable solvers for solving QPs are generally available, usually based on *interior point* or *active set* methods, and there has been a great deal of research related to the application of both approaches in the context of MPC [4].

Finding the solution of a QP in the case when the set of active constraints at the optimum is known, amounts to solving a linear system of equations that has a unique solution [5]. Active set methods are iterative processes that exploit the above property and try to guess at each iteration which are the active constraints at the optimal point. They usually consider active constraints one at a time, inducing a computation time directly related to m_a. On the contrary, the computation time of interior point methods is relatively constant, regardless of the number of active constraints. However, this constant computation time can be large enough to compare unfavorably with active set methods in cases where m_a is relatively small.

It should be noted that what we have to solve is not a singe QP but a series of QPs, which appear to be sequentially related. It is possible then to use information about the solution computed at sampling time k to accelerate the computation of the solution at sampling time $k + 1$. Such information is usually referred to as "warm starting", and *active set* methods typically gain more from it [4]. Hence, they are preferred when dealing with small to medium sized QPs where m_a is kept small.

3.2 Primal vs. dual strategies

There exist mainly two classes of active set methods, *primal* and *dual* strategies. Primal strategies ensure that all the constraints (6) are satisfied at every iteration. An important implication of this feature is that if there is a limit on

computation time (a real-time bound), e.g. because of the sampling period of the control law, the iterative process can be interrupted and still produce at any moment a feasible motion. Obviously, this comes at the cost of obtaining a sub-optimal solution.

One limitation of primal strategies is that they require an initial value for the variables U which already satisfy all the constraints. For a general QP, computing such an initial value can take as much time as solving the QP afterwards, which is a strong disadvantage. This is why dual methods are usually preferred: they satisfy all the constraints (6) only at the last iteration, but they do not require such an initial value.

3.3 Null space vs. range space algebra

There exist mainly two ways of making computations with the linear constraints (6), either considering the *null space* of the matrix G_{k+1}, orthogonal to the constraints, or the *range space* of this matrix, parallel to the constraints. The first choice leads to working with matrices of size $(n - m_a) \times (n - m_a)$, while the second choice leads to working with matrices of size $m_a \times m_a$. Hence, the most efficient of those two options depends on whether $m_a < n/2$ or not. It should be noted that, when dealing with ill-conditioned matrices, range space algebras can behave poorly.

3.4 Problem structure

As it was already pointed out, in practice the solution of the QP underlying a LMPC scheme is left to state of the art QP solvers [3]. Even though such solvers implement very efficient algorithms, in most cases they do not exploit the properties of each particular problem. One such property is that the matrix G_{k+1} of the constraints (6) can be expressed as a product of $D_{k+1}P_u$, where P_u is constant. In many applications [6] D_{k+1} is well structured and extremely sparse (a property that is lost after G_{k+1} is formed explicitly). The primal algorithm in [7] and dual algorithm in [8] are probably the ones that are considered as first choices when dealing with small to medium sized problems, however, they are not able to take advantage of the sparsity pattern of D_{k+1} and the fact that P_u is constant, leading to a requirement for new algorithms that account for this structure.

3.5 Our choice

- If the system in (1) is output controllable, P_u will have full row rank and by computing its (generalized) inverse off-line, a feasible U can be obtained at a low cost [9], [10]. Furthermore, if the solution of a QP can not be obtained within a predefined sampling time, we want to be able to interrupt the process and still obtain a feasible motion of our system. These considerations led to the development of a *primal solver*.

- Due to our assumption, that the number of active constraints is relatively small *i.e.* $m_a < n/2$, we chose to use a solver with *range space linear algebra*.

4 An optimized QP solver

4.1 Off-line change of variable

Typically, the first action of an active set method is to make a Cholesky decomposition of the matrix $Q = L_Q L_Q^T$. When range space algebra is used, at each iteration a change of variable involving L_Q is performed twice [7]. First, when adding a constraint to the so called *working set*, and second, when the search direction is evaluated. This results in using n^2 *flops* at each iteration[7]. In this way, the QP defined by (4) and (6) simplifies to a Least Distance Problem (LDP) [11]

$$\underset{V}{\text{minimize}} \quad \frac{1}{2} V^T V + V^T g_k \tag{7a}$$

$$\text{subject to} \quad \underbrace{D_{k+1} P_u}_{G_{k+1}} L_Q^{-T} V \leq b_{k+1}, \tag{7b}$$

where, $V = L_Q^T U$ and $g_k = L_Q^{-1} p_k$. In a general setting, representing (4) and (6) in the form of (7) using one change of variable before solving the QP is not performed, because the matrix-matrix product $G_{k+1} L_Q^{-T}$ has to be evaluated (which is computationally expensive if both matrices are dense).

For the problem treated in this article, however, the matrices L_Q and P_u are constant and the product $P_u L_Q^{-T}$ can be precomputed off-line. Furthermore, due to the assumption that D_{k+1} is sparse (with at most n_y nonzero entries in each row), forming $D_{k+1} P_u L_Q^{-T}$ requires mnn_y flops, which is computationally cheaper than using n^2 flops during each step of the solution. Note that in many applications, large parts of D_{k+1} can remain unchanged from one sampling time to the next. Due to the above considerations, we perform a change of variable and solve on-line the LDP (7).

4.2 The iterative process

Active set methods are iterative processes that try to guess at each iteration which are the active constraints, the inequalities in (7b) which hold as equalities at the minimum V^*. Indeed, once these equalities, denoted by

$$EV = q$$

[7] We measure computational complexity in number of floating-point operations, flops. We define a flop as one multiplication/division together with an addition. Hence, a dot product $a^T b$ of two vectors a, $b \in \mathbb{R}^n$ requires n flops.

are identified, the minimum of the LDP is [5]

$$V^* = -g_k + E^T\lambda \tag{8}$$

with Lagrange multipliers λ solving

$$EE^T\lambda = q + Eg_k. \tag{9}$$

In the case of a primal strategy, the iterations consist in solving these equations with a guess of what the active set should be, and if the corresponding solution violates some of the remaining constraints, include (usually) one of them (using a give criterion) in our guess (*working set*) and try again. Once the solution does not violate any other constraint, it remains to check that all the constraints we have included in our guess should actually hold as equalities. That is done by checking the sign of the Lagrange multipliers. A whole new series of iterations could begin then which alternate removing or adding constraints to our guess. All necessary details can be found in [5], [11].

4.3 Efficient update method

At each iteration we need to solve equations (8) and (9) with a new guess of the active set (here, we assume that the constraints in our guess are linearly independent, *i.e.* EE^T is full rank). The only thing that changes from one iteration to the next is that a single constraint is added or removed to/from the *working set*, *i.e.* only one line is either added or removed to/from the matrix E. Due to this structure, there exist efficient ways to compute the solution of (8) and (9) at each iteration by updating the solution obtained at the previous iteration without requiring the computation of the whole solution from scratch.

Probably the most efficient way to do so in the general case is the method described in [7]. There, a Gram-Schmidt decomposition of the matrix E is updated at each iteration at a cost of $2nm_a$ flops. Consequently, the Gram-Schmidt decomposition is used in a "clever way", allowing to update the solution of (8) and (9) at a negligible cost. In this way, the only computational cost when adding a constraint is the $2nm_a$ flops of the Gram-Schmidt update.

In our specific case, we can propose a slightly better option, based on the Cholesky decomposition of the matrix $EE^T = L_E L_E^T$. Below we describe the update procedure when a new row e is added to the matrix E. In such case, we need the decomposition of the new matrix

$$\begin{bmatrix} E \\ e \end{bmatrix} \begin{bmatrix} E^T & e^T \end{bmatrix} = \begin{bmatrix} EE^T & Ee^T \\ eE^T & ee^T \end{bmatrix}. \tag{10}$$

First note that, since the matrix $P_u L_Q^{-T}$ is constant, we can form off-line the Gramian matrix $\mathcal{G} = P_u L_Q^{-T} L_Q^{-1} P_u^T$, which is the matrix containing the dot products of each row of $P_u L_Q^{-T}$ with all others. Noting that, the rows of

matrix E and the (row) vector e are taken from the constraints (7b), the dot products Ee^T and ee^T can be obtained at a negligible cost from the entries of $\mathcal{G}$ under the action of the varying but extremely sparse and well structured matrix D_{k+1}. After matrix (10) is formed, classical methods for updating its Cholesky decomposition (once $EE^T = L_E L_E^T$ is known) require $m_a^2/2$ flops.

Using Cholesky decomposition, equation (9) can be solved in three very efficient steps:

$$w_1 = q + Eg_k, \tag{11a}$$

$$L_E w_2 = w_1, \tag{11b}$$

$$L_E^T \lambda = w_2. \tag{11c}$$

When one constraint is added to the matrix E, updating the value of w_1 requires only one dot product to compute its last element. Since only the last element of w_1 changes and only one new line is added to L_E, only the last element of w_2 needs to be computed to update its value, at the cost of a dot product. Only the third step requires more serious computations: since the matrix L_E is lower triangular of size m_a, solving this system requires $m_a^2/2$ flops.

Once equation (9) is solved for the Lagrange multipliers λ, the computation of V in (8) requires a nm_a matrix-vector product. In total, the above update requires $nm_a + m_a^2$ flops, which is slightly better than the $2nm_a$ found in [7], which is possible in our case due to the precomputation of the matrix $\mathcal{G}$ off-line. Even though V (computed from (8)) satisfies the equality constraints $EV = q$, it is not guaranteed to satisfy all the inequality constraint not included in the *working set*. In order to produce feasible iterates, at each step, the scheme presented in [5] (pp. 468-469), [9] is used.

The case when a constraint is removed from E is handled in a classical way (see [12]), and is not presented here.

4.4 Approximate solution & warm start

Depending on the sampling time of the control, obtaining the solution of each QP might not be possible. Because of this, here we present a modification of a standard primal algorithm that computes a "good" approximate solution faster. As observed in [9], [14], a "good" approximate solution does not result in a significant decrease in the quality of the MPC control law.

As already mentioned in Section 4.2, once a solution V (for some guess of the active set) that does not violate any of the constraints (7b) is found, it can be certified to be the optimal point if $\lambda_i > 0$ $(i = 1, \ldots, m_a)$. If this test fails, (usually) one constraint is dropped from the *working set*, resulting in a new series of iterations. In order to speed-up the on-line computation, we propose to terminate the solution of each QP once a solution V of (8) that satisfies all constraints in (7b) is found, regardless of signs of the Lagrange multipliers. Accounting for the negative entries of λ is then performed when

formulating the warm start for the next QP. Under the assumption that the active set of the QP solved at sampling time k closely resembles the one of the QP that needs to be solved at sampling time $k+1$, we use as an initial guess for the *working set* all active constraints from the previous QP except the ones that correspond to negative Lagrange multipliers. In that way, the modification of the *working set* is no longer treated separately at a local level (for each separate QP), but rather considered as a shared resource among the whole sequence.

The reasoning for starting with a nonempty *working set* can be motivated by noting that, if only adding constraints to our guess for the active set is considered, each iteration of the presented algorithm requires $nm + m_a^2$ flops. If the solution of each QP starts with an empty *working set*, the complexity of adding m_a constraints (one at a time) is approximately $nmm_a + m_a^3/3 + m_a^2/2$ flops[8]. In contrast, if matrix E from the previous QP is used (with some rows removed), the only necessary computation required for realizing the warm start is finding the Cholesky decomposition of the modified EE^T. This can be done by updating the already available factorization $L_E L_E^T$ from the previous QP, which (depending on which constraints are removed) requires at most $m_a^3/3$ flops, which is a tremendous improvement over the $nmm_a + m_a^3/3 + m_a^2/2$ flops that would have been necessary to reach the same active set through the whole set of iterations.

In [9], we already applied the above idea using the LMPC scheme for walking motion generation for a humanoid robot proposed in [13], and the active set when doing so is in most cases correct or includes only one, and in rare cases two unnecessarily activated constraints. This leads to slightly sub-optimal solutions, which nevertheless are feasible. We have observed that this does not affect the stability of the scheme: the difference in the generated walking motions is negligible, however, the computation time is considerably smaller (see [9] for results from a numerical comparison with a state of the art QP solver).

When a nonempty initial active set is specified, the initial point needs to lie on the constraints in this set. If the system in (1) is output controllable, such point can be generated by using a procedure similar to the one presented in [10]. In the general case, however, a general feasibility problem has to be solved.

5 Conclusion

In this article we presented an optimized algorithm for the fast solution of a quadratic program in the context of model predictive control. We discussed

[8] To this count one should add $nmm_a - nm_a^2/2 - nm_a/2$ flops, which is the complexity of checking whether V (computed from (8)) violates any of the inequality constraints not included in the active set. This check is common for all active set algorithms, and is not discussed in this article.

alternative solution methods, and analyzed their properties for different problem structures. The presented algorithm was designed with the intention of using as much as possible data structures which can be precomputed off-line. In such a way, we are able to decrease the on-line computational complexity. A strategy for producing "good" approximate solutions in the presence of a real-time bound on the computation time was presented.

References

1. S. J. Qin, and T. A. Badgwell, "An overview of industrial model predictive control technology. In chemical process control: Assessment and new directions for research," in *AIChE Symposium Series 316, 93, Jeffrey C. Kantor, Carlos E. Garcie and Brice Carnahan Eds.,* 232-256, 1997.
2. J. Maciejowski, "Predictive Control with Constraints," in *Prentice Hall,* 2001.
3. K. Schittkowski, "QL: A Fortran code for convex quadratic programming - User's guide," *University of Bayreuth*, Report, Version 2.11, 2005.
4. S. Wright, "Applying new optimization algorithms to model predictive control," in *Proc. of CPC-V*, 1996.
5. J. Nocedal, and S. J. Wright, "Numerical optimization," *Springer Series in Operations Research, 2nd edition*, 2000.
6. H. Diedam, D. Dimitrov, P.-B. Wieber, M. Katja, and M. Diehl, "Online walking gait generation with adaptive foot positioning through linear model predictive control," in *Proc. of the IEEE/RSJ IROS,* pp. 1121-1126, 2008.
7. P.E. Gill, N.I. Gould, W. Murray, M.A. Saunders, and M.H. Wright "A weighted gram-schmidt method for convex quadratic programming," *Mathematical Programming,* Vol.30, No.2, pp.176-195, 1984
8. D. Goldfarb, and A. Idnani, "A numerically stable dual method for solving strictly convex quadratic programs," *Mathematical Programming,* 27:1-33, 1983.
9. D. Dimitrov, P.-B. Wieber, O. Stasse, J. Ferreau, and H. Diedam, "An optimized linear model predictive control solver for online walking motion generation," in *Proc. of the IEEE Int. Conf. on Robot. & Automat.,* pp. 1171-1176, 2009.
10. D. Dimitrov, J. Ferreau, P.-B. Wieber, and M. Diehl, "On the implementation of model predictive control for on-line walking pattern generation," in *Proc. of the IEEE Int. Conf. on Robot. & Automat.,* pp. 2685-2690, 2008.
11. R. Fletcher, "Practical Methods of Optimization," *John Wiley & Sons,* 1981.
12. H.J. Ferreau, "An online active set strategy for fast solution of parametric quadratic programs with applications to predictive engine control," *University of Heidelberg,* 2006.
13. P.-B. Wieber, "Trajectory free linear model predictive control for stable walking in the presence of strong perturbations," in *Proc. of IEEE-RAS Int. Conf. on Humanoid Robots,* pp.137-142, 2006.
14. Y. Wang, and S. Boyd, "Fast model predictive control using online optimization," in *Proc. of 17th IFAC World Congress on Automatic Control,* pp. 6974-6979, 2008.