HAL
open science

# An Optimized Linear Model Predictive Control Solver for Online Walking Motion Generation

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

## To cite this version:

Dimitar Dimitrov, Pierre-Brice Wieber, Olivier Stasse, Joachim Ferreau, Holger Diedam. An Optimized Linear Model Predictive Control Solver for Online Walking Motion Generation. ICRA 2009 - IEEE International Conference on Robotics & Automation, May 2009, Kobe, Japan. pp.1171-1176, 10.1109/ROBOT.2009.5152380 . inria-00390593

## HAL Id: inria-00390593

## https://inria.hal.science/inria-00390593

Submitted on 2 Jun 2009

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# An Optimized Linear Model Predictive Control Solver for Online Walking Motion Generation

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

***Abstract*— This article addresses the fast solution of a Quadratic Program underlying a Linear Model Predictive Control scheme that generates walking motions. We introduce an algorithm which is tailored to the particular requirements of this problem, and therefore able to solve it efficiently. Different aspects of the algorithm are examined, its computational complexity is presented, and a numerical comparison with an existing state of the art solver is made. The approach presented here, extends to other general problems in a straightforward way.**

## I. INTRODUCTION

The difficulty in generating stable walking motions is mostly due to the fact that moving one's Center of Mass (CoM) entirely relies on the unilateral contact forces between the feet and the ground [1]. In the presence of disturbances, this restriction severely limits the capacity of the system to follow predefined motions. It is necessary therefore to generate walking motions which are always adapted online to the current state of the system.

This issue is addressed in [2] through a Zero Moment Point (ZMP) Preview Control scheme that generates dynamically stable motions for a humanoid robot by approximating its nonlinear dynamics with the linear dynamics of a point mass. The walking motion generation problem is formulated then as a typical infinite horizon Linear Quadratic Regulator (LQR). This approach explicitly accounts for the system state and leads to fast computations.

One disadvantage of this scheme is that it does not explicitly account for the constraints on the ZMP, which must always lie in the support polygon: in case of disturbances, the system's response can overshoot the boundaries of this polygon. This limitation is addressed in [3], where instead of solving an infinite horizon LQR, the utilization of a receding horizon LQR with constraints is proposed, in other words, a Linear Model Predictive Control scheme (LMPC). In [4] the authors develop the idea further by using an augmented state vector that addresses the problem of feet repositioning in the presence of strong disturbances.

The guarantee given by this LMPC scheme of keeping the system's response within a given set of constraints comes at a price of increased computation demands. At each control step, the application of this LMPC scheme amounts to forming and solving a Quadratic Program (QP) [5].

A more efficient formulation of this LMPC scheme was proposed in [6]. The authors proposed the utilization of a variable sampling time scheme, in combination with a method for developing a reliable guess for the active constraints at the solution by introducing the notion of "sliding constraints". It is shown that such a guess can drastically reduce the computational burden for each iteration.

In practice, the solution of the underlying QP is left to state of the art QP solvers [7]. 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 sometimes speed up computations a lot.

In this article, we introduce an optimized algorithm for the fast solution of a particular QP in the context of LMPC for online walking motion generation. It 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 compare then the run-time of our algorithm with a state of the art solver.

The article is organized as follows: in Section II we briefly outline the LMPC scheme presented in [3]. In Section III we discuss alternative methods for the solution of a QP, and analyze how do their properties reflect on our problem. In Section IV we present our algorithm and discuss its features, state its complexity and compare it with the algorithm in [8]. Finally, in Section V we present a numerical comparison with QL [7].

## II. AN LMPC SCHEME GENERATING WALKING MOTIONS

The Model Predictive Control (MPC) scheme introduced in [2], [3] for generating walking motions works primarily with the motion of the CoM of the walking robot. In order to obtain an LMPC scheme, it is assumed that the robot walks on a constant horizontal plane, and that the motion of its CoM is also constrained to a horizontal plane at a distance $h$ above the ground, so that its position in space can be defined using only two variables $(x, y)$.

Only trajectories of the CoM with piecewise constant jerks $\dddot{x}$ and $\dddot{y}$ over time intervals of constant length $T$ are considered. That way, focusing on the state of the system at the instants $t_k = kT$,

$$\hat{x}_k = \begin{pmatrix} x(t_k) \\ \dot{x}(t_k) \\ \ddot{x}(t_k) \end{pmatrix}, \ \hat{y}_k = \begin{pmatrix} y(t_k) \\ \dot{y}(t_k) \\ \ddot{y}(t_k) \end{pmatrix}, \tag{1}$$

Dimitar Dimitrov is with Örebro University - Sweden, mitko@roboresearch.net
Pierre-Brice Wieber is with INRIA Grenoble - France, pierre-brice.wieber@inria.fr
Olivier Stasse is with JRL - Japan, olivier.stasse@aist.go.jp
Hans Joachim Ferreau is with KU Leuven - Belgium, joachim.ferreau@esat.kuleuven.be
Holger Diedam is with Heidelberg University - Germany, hdiedam@ix.urz.uni-heidelberg.de

the integration of the constant jerks over the time intervals of length $T$ gives rise to a simple recursive relationship:

$$\hat{x}_{k+1} = A\,\hat{x}_k + B\,\dddot{x}(t_k), \tag{2}$$

$$\hat{y}_{k+1} = A\,\hat{y}_k + B\,\dddot{y}(t_k), \tag{3}$$

with a constant matrix $A$ and vector $B$.

Then, the position $(z^x, z^y)$ of the ZMP on the ground corresponding to the motion of the CoM of the robot is approximated by considering only a point mass fixed at the position of the CoM instead of the whole articulated robot:

$$z^x_k = \begin{pmatrix} 1 & 0 & -h/g \end{pmatrix} \hat{x}_k, \tag{4}$$

$$z^y_k = \begin{pmatrix} 1 & 0 & -h/g \end{pmatrix} \hat{y}_k, \tag{5}$$

with $h$ the constant height of the CoM above the ground and $g$ the norm of the gravity force.

Using the dynamics (2) recursively, we can derive a relationship between the jerk of the CoM and the position of the ZMP over time intervals of length $NT$:

$$Z^x_{k+1} = P_{zs}\hat{x}_k + P_{zu}\dddot{X}_k, \tag{6}$$

$$Z^y_{k+1} = P_{zs}\hat{y}_k + P_{zu}\dddot{Y}_k, \tag{7}$$

with constant matrices $P_{zs} \in \mathbb{R}^{N\times 3}$ and $P_{zu} \in \mathbb{R}^{N\times N}$, with

$$Z^x_{k+1} = \begin{pmatrix} z^x_{k+1} \\ \vdots \\ z^x_{k+N} \end{pmatrix}, \ \dddot{X}_k = \begin{pmatrix} \dddot{x}_k \\ \vdots \\ \dddot{x}_{k+N-1} \end{pmatrix}, \tag{8}$$

and similar definitions for $Z^y_{k+1}$ and $\dddot{Y}_k$.

In order for a motion of the CoM to be feasible, we need to ensure that the corresponding position of the ZMP always stays within the convex hull of the contact points of the feet of the robot on the ground [1]. This constraint can be expressed at the instants $t_k$ for a whole time interval of length $NT$ as:

$$b^l_{k+1} \leq D_{k+1} \begin{pmatrix} Z^x_{k+1} \\ Z^y_{k+1} \end{pmatrix} \leq b^u_{k+1}, \tag{9}$$

with a $D_{k+1} \in \mathbb{R}^{m\times 2N}$ a matrix varying with time but extremely sparse and well structured, with only $2m$ non zero values on 2 diagonals.

The LMPC scheme involves then a quadratic cost which is minimized in order to generate a "stable" motion [3], [6], leading to a canonical Quadratic Program (QP)

$$\min_u \ \frac{1}{2}u^T Q u + p_k^T u \tag{10}$$

with

$$u = \begin{pmatrix} \dddot{X}_k \\ \dddot{Y}_k \end{pmatrix}, \tag{11}$$

$$Q = \begin{pmatrix} Q' & 0 \\ 0 & Q' \end{pmatrix} \tag{12}$$

where $Q'$ is a positive definite constant matrix, and

$$p_k^T = \begin{pmatrix} \hat{x}_k^T & \hat{y}_k^T \end{pmatrix} \begin{pmatrix} P_{su} & 0 \\ 0 & P_{su} \end{pmatrix} \tag{13}$$

where $P_{su}$ is also a constant matrix (see [4] for more details).

With the help of the relationships (6) and (7), the constraints (9) on the position of the ZMP can also be represented as constraints on the jerk $u$ of the CoM:

$$b'^l_{k+1} \leq D_{k+1} \begin{pmatrix} P_{zu} & 0 \\ 0 & P_{zu} \end{pmatrix} u \leq b'^u_{k+1}. \tag{14}$$

Since the matrix $Q$ is positive definite and the set of linear constraints (14) forms a (polyhedral) convex set, there exists a unique global minimizer $u^*$ [9].

The number of variables in the minimization problem (10) is equal to $n = 2N$ and the number of constraints (14) is of the same order, $m \approx 2N$. Typical uses of this LMPC scheme consider $N = 75$ and $T = 20\,ms$, for computations made on a time interval $NT = 1.5\,s$, approximately the time required to make 2 walking steps [6]. This leads to a QP which is typically considered as small or medium sized.

Another important measure to take into account about this QP is the number $m_a$ of *active constraints* at the minimum $u^*$, the number of inequalities in (14) which hold as equalities. We have observed that at steady state, this number is usually very low, $m_a \leq m/10$, and even in the case of strong disturbances, we can observe that it remains low, with usually $m_a \leq m/2$ [6].

## III. General design choices for a QP solver

### A. Interior point or active set method

The choice of an algorithm that can solve efficiently a QP with the above characteristics is not unique. Fast and reliable solvers based on *interior point* or *active set* methods are generally available and there has been a great deal of research related to the application of both approaches in the context of MPC [5], [10], [11], [12].

Finding the solution of a QP in the case when the set of active constraints at $u^*$ is known, amounts to solving a linear system of equations that has a unique solution [9]. 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 minimum $u^*$. They usually consider active constraints one at a time, inducing a computation time directly related to the number of active constraints. 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 such as the one here, a small QP with relatively few active constraints.

The question of "warm-starting" the solvers is the final important feature to consider. Indeed, what we have to solve is not a unique QP but a series of QPs at each time $t_k$ which appear to be sequentially related. It is possible then to use information about the solution computed at time $t_k$ to accelerate the computation of the solution at time $t_{k+1}$ [6]. Active set methods typically gain more from "warm-starting" [5]. All these reasons lead to the conclusion that an active set method should be preferable in our case.

### B. Primal or dual strategy

There exist mainly two classes of active set methods, *primal* and *dual* strategies. Primal strategies ensure that all the constraints (14) are satisfied at every iteration. An important implication for us of this feature is that if there is a limit on computation time, for example because of the sampling period of the control law, the iterative process can be interrupted and still produce at any moment a feasible motion, satisfying all the constraints on the ZMP.

Obviously, this comes at the cost of producing a sub-optimal solution. A theoretical analysis [13] of our LMPC scheme shows however, that the choice of a specific quadratic cost is of secondary importance as long as some broad properties are satisfied. This indicates that, some small amount of sub-optimality should be acceptable. Some care must be taken though because stability of the resulting walking motion directly derives from this minimization, so sub-optimality should remain a "second choice", when we really don't have time for a better solution.

Combined with a "warm-start" of the solver when solving a series of sequentially related QPs, as discussed in the previous subsection, this interruption of the iterative process gives rise to some sort of "lazy" or delayed optimizing scheme, improving optimality from QP to QP similarly to what is described in [14].

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 deterrent. This is why, dual methods are usually preferred: they satisfy all the constraints (14) only at the last iteration, but they don't require such an initial value.

In our case however, an initial value for $u$ satisfying all the constraints (14) can be computed easily and efficiently. First of all, a position of the ZMP satisfying the constraints (9) can be obtained easily, considering for example the point $(Z_m^x, Z_m^y)$ in the middle of the convex hull of the contact points. Then, observing that the matrix $P_{zu}$ that appears in the relationships (6) and (7) is invertible, it is straightforward to invert these relationships and obtain a feasible initial value

$$u_0 = \begin{pmatrix} P_{zu}^{-1} & 0 \\ 0 & P_{zu}^{-1} \end{pmatrix} \begin{pmatrix} Z_m^x - P_{zs}\hat{x}_k \\ Z_m^y - P_{zs}\hat{y}_k \end{pmatrix}. \tag{15}$$

Alternatives strategies exist, such as the primal-dual one introduced in [14], however the decisive property that a primal method can be interrupted at all time and still produce a feasible solution will direct our choice here.

### C. Null space or range space algebra

There exist mainly two ways of making computations with the linear constraints (14), either considering the *null space* of the matrix $D_{k+1}P_{zu}$, 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 $m_a \times m_a$. The most efficient of those two options from the point of view of computation time will depend therefore on whether $m_a < n/2$ or not. Considering that this is always true in our case, the choice of a range space linear algebra is obvious. One must take care however that, range space algebras can behave poorly with ill-conditioned matrices. Fortunately, this is not the case for our LMPC.

## IV. An optimized QP solver

### A. Off-line change of variable

The first action of a range space active set method is usually to make a Cholesky decomposition of the matrix $Q = L_Q L_Q^T$ and make an internal change of variable

$$v = L_Q^T u. \tag{16}$$

That way, the Quadratic Problem (10) simplifies to a Least Distance Problem (LDP) [15]

$$\min_v \ \frac{1}{2}\|v + L_Q^{-T} p_k\|^2.$$

In our case, we need to solve online a sequence of QPs (10)-(14) where the matrices $Q'$, $P_{zu}$ and $P_{su}$ are constants. We can therefore make this change of variable completely off-line and save a lot of online computation time by directly solving online the LDP:

$$\min_v \ \frac{1}{2}\|v + p'_k\|^2 \tag{17}$$

with

$$p_k'^T = \begin{pmatrix} \hat{x}_k^T & \hat{y}_k^T \end{pmatrix} \begin{pmatrix} P_{su}L_Q^{-T} & 0 \\ 0 & P_{su}L_Q^{-T} \end{pmatrix} \tag{18}$$

and constraints

$$b_{k+1}'^l \leq D_{k+1} \begin{pmatrix} P_{zu}L_Q^{-T} & 0 \\ 0 & P_{zu}L_Q^{-T} \end{pmatrix} v \leq b_{k+1}'^u. \tag{19}$$

Considering the complexity counts presented in [8], realizing this change of variable off-line allows saving $n^2$ *flops* at each iteration of our algorithm. Note that, 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.

### B. The iterative process

As mentioned earlier, active set methods are iterative processes that try to guess at each iteration which are the active constraints, the inequalities in (19) which hold as equalities at the minimum $v^*$. Indeed, once we identify these equalities, noted

$$Ev = b,$$

the minimum of our LDP is [9],

$$v^* = -p'_k + E^T\lambda \tag{20}$$

with Lagrange multipliers $\lambda$ solution of

$$EE^T\lambda = b + Ep'_k. \tag{21}$$

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 happens to violate one constraint, include it in our guess and try again. Once the solution does not violate any other constraint, there remains to check that all the constraints we have included in our guess should really hold as equalities, what is done by checking the sign of the Lagrange multipliers. A whole new series of iterations begins then which alternate removing or adding constraints to our guess. All necessary details can be found in [15].

We have observed however that, not checking the sign of the Lagrange multipliers and not considering removing constraints from our guess does not affect the result we obtain from our LMPC scheme in a noticeable way. As we will see in the next Section, our final guess for 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. Furthermore, we have observed that, this does not affect the stability of our scheme: the difference in the generated walking motions is negligible.

### C. Efficient update methods

At each iteration, we need to solve the equations (20) and (21) with a new guess of active set, but the only thing that changes from one iteration to the next is that we add one new constraint to our guess and therefore one new line to the matrix $E$. Thanks to this structure, there exist efficient ways to compute the solution of (20) and (21) at each iteration by updating the solution obtained at the previous iteration without requiring computing the whole solution from scratch.

Probably the most efficient way to do so in the general case is the method described in [8]. There, a Gram-Schmidt decomposition of the matrix $E$ is updated at each iteration at a cost of $2nm_a$ flops. This Gram-Schmidt decomposition is used then in an clever way, allowing to update the solution of (20) and (21) at a negligible cost. In this way, the only computational cost of an iteration when solving an LDP is the $2nm_a$ flops of the Gram-Schmidt update.

In our specific case, we can propose a slightly better option, based on a Cholesky decomposition of the matrix $EE^T = L_E L_E^T$. When we add a new row $e$ to the matrix $E$, we need the decomposition of the new matrix

$$\begin{pmatrix} E \\ e \end{pmatrix} \begin{pmatrix} E^T & e^T \end{pmatrix} = \begin{pmatrix} EE^T & Ee^T \\ eE^T & ee^T \end{pmatrix}. \tag{22}$$

We need first of all to update this matrix with the dot products $Ee^T$ and $ee^T$. Since the rows of the matrix $E$ and the vector $e$ are taken from the constraints (19), these dot products can be obtained at a negligible cost from the dot products of rows of the matrix $P_{zu} L_Q^{-T}$ which is constant and computed off-line, under the action of the varying but extremely sparse and well structured matrix $D_{k+1}$. The matrix (22) can be updated then at a negligible cost, and from there, classical methods for updating its Cholesky decomposition typically require only $m_a^2/2$ flops.

With the help of this Cholesky decomposition, the equation (21) can be solved in three very efficient steps:

$$w_1 = b + Ep'_k, \tag{23a}$$

$$L_E w_2 = w_1, \tag{23b}$$

$$L_E^T \lambda = w_2. \tag{23c}$$

Since the matrix $E$ only gains one new row at each iteration, 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 the equation (21) is solved and we have the Lagrange multipliers $\lambda$, the computation of the optimum $v^*$ with equation (20) only requires a $nm_a$ matrix-vector product. In total, we need $nm_a + m_a^2$ flops, which is slightly better than the $2nm_a$ found in [8], what's possible in our case thanks to the pre-computation of the dot products in (22).

### D. Maintaining feasible iterates

We still need to produce a feasible point at each iteration since we can observe that the solution $v^*$ of equations (20) and (21) satisfies all the constraints (19) only at the last iteration. We also need to choose which constraint is included in our guess at each iteration. These two questions are related and form the last building block or our algorithm, which is a very classical procedure [9].

The feasible solution $v^{(i)}$ at each iteration $i$ will be a point between the feasible solution of the previous iteration $v^{(i-1)}$ and the solution $v^*$ of equations (20) and (21). This point will correspond to the first constraint which is hit when moving from $v^{(i-1)}$ to $v^*$. And it is this constraint which will be added to our guess for the active set. In that way, if we begin with a feasible point $v^{(0)}$ and an empty guess for the active set, we obtain a series of points $v^{(i)}$ which always lie exactly on the constraints included in the guess of active set. And since we always include the first constraint which is hit, we ensure that all the constraints (19) are always satisfied by this series of points at all time. More precisely, with

$$v^{(i)} = v^{(i-1)} + \alpha d, \tag{24}$$

$$d = v^* - v^{(i-1)}, \tag{25}$$

considering separately each constraint $j$ of (19),

$$b'^l_j \leq e_j v^{(i)} \leq b'^u_j,$$

the scalar $\alpha$ corresponding to the first constraint hit is

$$\alpha = \min_j \left\{ \begin{array}{l} \dfrac{b'^u_j - e_j v^{(i-1)}}{e_j d} \text{ when } e_j d > 0 \\ \dfrac{b'^l_j - e_j v^{(i-1)}}{e_j d} \text{ when } e_j d < 0 \end{array} \right\}. \tag{26}$$

If a step $\alpha = 1$ can be realized without violating any constraint, it means we have reached the last iteration of our

algorithm: the optimum computed with our guess of active set satisfies all the constraints.

Obviously, from equation (24),

$$e_j v^{(i)} = e_j v^{(i-1)} + \alpha e_j d,$$

so computing the value of $e_j d$ is enough to update at no cost the value of $e_j v^{(i)}$ for the next iteration. Since these values need only be computed for the $m - m_a$ constraints which haven't been included yet in our guess of active set, the cost of computing the step $\alpha$ is $n(m - m_a)$ flops.

**Algorithm 1:** Primal Least Distance Problem solver

**input** : LDP, initial active set guess $W^{(0)}$ and corresponding $v^{(0)}$

**output**: $v^*$ and $W^*$, approximate solution to the LDP

*Complexity:* $nm + m_a^2$;

**(0)** Set $k \leftarrow 0$;

**(1)** Compute the direction $d^{(k)}$ from (20) and (25) after following the procedure in (23);

**(2)** Compute the step $\alpha^{(k)}$ with (26);

**(3)** Check the exit condition.
**if** $\alpha^{(k)} > 1$ **then**
  set $\alpha^{(k)} \leftarrow 1$;
**else**
  determine the index $j \notin W^{(k)}$ of the blocking constraint corresponding to $\alpha^{(k)}$;
**end**

**(4)** Make a step using (24).
**if** $\alpha^{(k)} = 1$ **then**
  set $v^* \leftarrow v^{(k+1)}$, $W^* \leftarrow W^{(k)}$ and stop!
**else**
  set $W^{(k+1)} \leftarrow [W^{(k)}\ j]$;
  set $k \leftarrow k + 1$ and continue with step (1);
**end**

### *E. The warm start*

Summing up the previous computations, each iteration of our algorithm requires $nm + m_a^2$ flops. However, since constraints are added one at a time in our iterative guess of the active set, summing up all the iterations with $m_a$ increasing by one each time leads finally to $nmm_a + m_a^3/3$ flops, which is remarkably more. Fortunately, the active set of the QP solved for our LMPC scheme at time $t_k$ will closely resemble the one of the QP that needs to be solved at time $t_{k+1}$. There lies a possibility to warm-start our solver: start with a first guess for the active set which is in fact the active set identified for the solution of the previous QP.

When a nonempty initial active set is specified, the initial point $v^{(0)}$ needs to lie on the constraints in this set. Fortunately, the last iteration computed for the previous QP satisfies this condition, and can be used therefore for warm starting our algorithm. The only necessary computation required for realizing this warm start is the Cholesky decomposition of the initial matrix $E$ which is not empty anymore (but which can be obtained or updated from the previous QP) and the corresponding initial values of $w_1$ and $w_2$. This requires at most $m_a^3/3$ flops, a tremendous improvement over the $nmm_a + m_a^3/3$ flops that would have been necessary to reach the same active set through the whole set of iterations.

An important detail now is that we decided in Section IV-B to never check the sign of the Lagrange multipliers, which indicate whether all the constraints included in our guess for the active set should really be there. Passing on an inadequate guess from one QP to the next, always including new constraints each time and never removing any, could lead to a serious degradation of the quality of our solutions, and it is what we have observed. Correcting this problem is easy though, we check the sign of the Lagrange multipliers before warm starting and do not include the constraints which fail this test in our first guess. This proved to work perfectly, at no cost.

## V. Numerical results

Before implementing the algorithm described in this publication, the computation of our LMPC scheme relied on QL [7], a state of the art QP solver implementing a dual active set method with range space linear algebra. The fact that it implements a dual strategy implies that it can not be interrupted before reaching its last iteration since intermediary iterates are not feasible. Furthermore, no possibilities of warm starting are offered to the user. However, since it relies on a range space algebra, comparisons of computation time with our algorithm without warm starting are meaningful.

We naturally expect to gain $n^2$ flops at each iteration thanks to the off-line change of variable. Furthermore, QL does not implement double sided inequality constraints like the ones we have in (19), so we need to double artificially the number $m$ of inequality constraints. Since computing the step $\alpha$ requires $nm$ flops at each iteration and $m \approx n$ in our case, that's a second $n^2$ flops which we save with our algorithm. The mean computation time when using QL is 7.86 ms on the CPU of our robot, 2.81 ms when using our Primal Least Distance Problem (PLDP) solver. Detailed time measurements can be found in Fig. 1.

Even more interesting is the comparison with our warm start scheme combined with a limitation to two iterations for solving each QP. As discussed in section III-B, this generates short periods of sub-optimality of the solutions, but with no noticeable effect on the walking motions obtained in the end: this scheme works perfectly well, with a mean computation time of only 0.74 ms and, most of all, a maximum time less than 2 ms!

A better understanding of how these three options relate can be obtained from Fig. 2, which shows the number of constraints activated by QL for each QP, which is the exact number of active constraints. This figure shows then the difference between this exact number and the approximate number found by PLDP, due to the fact that we decided to never check the sign of the Lagrange multipliers. Most

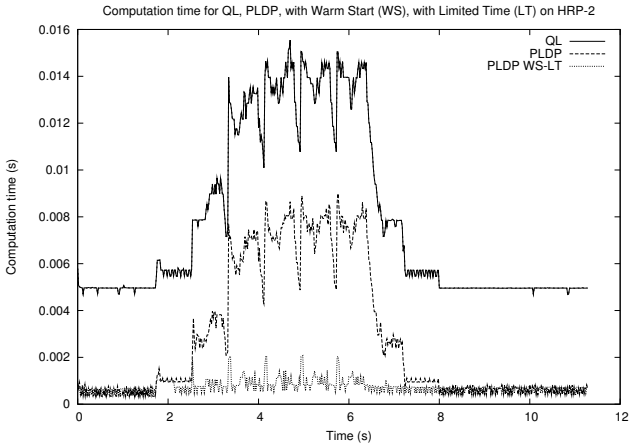


Fig. 1. Computation time required by a state of the art generic QP solver (QL), our optimized solver (PLDP), and our optimized solver with warm start and limitation of the computation time, over 10 seconds of experiments.

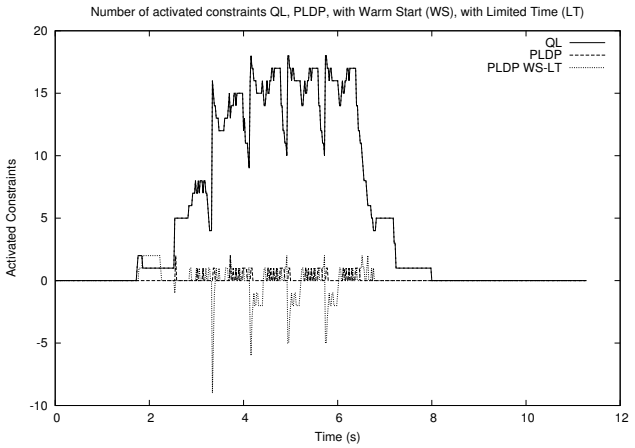


Fig. 2. Number of active constraints detected by a state of the art solver (QL), difference with the number of active constraints approximated by our algorithm (PLDP), between 0 and 2, and difference with the approximation by our algorithm with warm start and limitation of the computation time, between -9 and 2.

often, the two algorithms match or PLDP activates only one constraint in excess. The difference is therefore very small.

This difference naturally grows when implementing a maximum of two iterations for solving each QP in our warm starting scheme: when a whole group of constraints needs to be activated at once, this algorithm can identify only two of them each time a new QP is treated. The complete identification of the active set is delayed therefore over subsequent QPs: for this reason this algorithm appears sometimes to miss identifying as many as 9 active constraints, while still activating at other times one or two constraints in excess. Note that, regardless of how far we are from the real active set, the solution obtained in the end is always feasible.

## VI. Conclusion

In this article we introduced an optimized algorithm for the fast solution of a particular QP in the context of LMPC for online walking motion generation. We discussed alternative methods for the solution of QPs, and analyzed how do their properties reflect on our particular problem. Our algorithm was designed with the intension to use as much as possible data structures which are pre-computed off-line. In such a way, we are able to decrease the online computational complexity. We made a C++ implementation of our algorithm and presented both theoretical and numerical comparison with state of the art QP solvers. The issue of "warm-starting" in the presence of a real-time bound on the computation time was tested numerically, and we presented quantifiable results.

## Acknowledgments

This research was supported by the French Agence Nationale de la Recherche, grant reference ANR-08-JCJC-0075-01. This research was also supported by Research Council KUL: CoE EF/05/006 Optimization in Engineering Center (OPTEC). The research of Pierre-Brice Wieber was supported by a Marie Curie International Outgoing Fellowship within the 7th European Community Framework Programme. Hans Joachim Ferreau holds a PhD fellowship of the Research Foundation - Flanders (FWO).

## References

[1] P.-B. Wieber, "Holonomy and nonholonomy in the dynamics of articulated motion," in *Proc. of the Ruperto Carola Symposium on Fast Motion in Biomechanics and Robotics,* 2005.

[2] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa, "Biped walking pattern generation by using preview control of zero-moment point," in *Proc. of the IEEE Int. Conf. on Robot. & Automat.,* pp.1620-1626, 2003.

[3] 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.

[4] 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,* 2008.

[5] S. Wright, "Applying new optimization algorithms to model predictive control," in *Proc. of CPC-V*, 1996.

[6] 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.,* 2008.

[7] K. Schittkowski, "QL: A Fortran code for convex quadratic programming - User's guide," *Department of Mathematics, University of Bayreuth*, Report, Version 2.11, 2005.

[8] 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

[9] J. Nocedal, and S. J. Wright, "Numerical optimization," *Springer Series in Operations Research, 2nd edition*, 2000.

[10] R. A. Bartlett, A. Wächter, and L. T. Biegler, "Active set vs. interior point strategies for model predictive control," in *Proc. of the American Control Conference,* pp.4229-4233, June, 2000.

[11] C. Rao, J. B. Rawlings, and S. Wright, "Application of interior point methods to model predictive control," *J. Opt. Theo. Applics.,* pp. 723-757, 1998.

[12] I. Das, "An active set quadratic programming algorithm for real-time predictive control," *Optimization Methods and Software,* Vol.21, No.5, pp.833-849, 2006.

[13] P.-B. Wieber, "Viability and Predictive Control for Safe Locomotion", Proceedings IEEE/RSJ International Conference on Intelligent RObots and Systems 2008.

[14] 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.

[15] R. Fletcher, "Practical Methods of Optimization," *John Wiley & Sons,* 1981.