A sparse model predictive control formulation for walking motion generation

Dimitar Dimitrov, Alexander Sherikov, Pierre-Brice Wieber

A sparse model predictive control formulation for walking motion generation

Dimitar Dimitrov, Alexander Sherikov, Pierre-Brice Wieber

***Abstract*— This article presents a comparison between dense and sparse model predictive control (MPC) formulations, in the context of walking motion generation for humanoid robots. The former formulation leads to smaller, the latter one to larger but more structured optimization problem. We put an accent on the sparse formulation and point out a number of advantages that it presents. In particular, motion generation with variable center of mass (CoM) height, as well as variable discretization of the preview window, come at a negligible additional computational cost. We present a sparse formulation that comprises a diagonal Hessian matrix and has only simple bounds (while still retaining the possibility to generate motions for an omnidirectional walk). Finally, we present the results from a customized code used to solve the underlying quadratic program (QP).**

I. INTRODUCTION

The importance of generating online "safe" walking/running motions for a humanoid robot has been recognized by researchers, judging by the large number of papers dedicated to this problem. There is a variety of control/planning schemes proposed. A promising approach is based on the use of an approximate dynamical model, where the approximation error is compensated for by the application of a preview type of controller with (possibly) fast control sampling rates. Such control schemes usually involve the minimization of an objective function over a given finite prediction horizon subject to input and state constraints ($i.e.$, a classical MPC scheme). This is the type of schemes we address here.

A variety of MPC formulations for walking motion generation have been presented [2]-[8] (some of them have been successfully tested on the HRP-2 and NAO platforms [1]). They address important problems related to humanoid walking, like stabilization of the system along a given reference trajectory, optimal foot repositioning (accounting for safety zones for foot placement), tracking a given reference speed (for both rotation and translation), analysis related to the efficient solution of the underlying QP, etc. Even though when addressing particular problems, the above mentioned references use different objective functions ($e.g.$, including alternative formulations with ℓ_2-, ℓ_1-, ℓ_∞-penalties), and account for different constraints, they have one thing in common. Namely, the MPC formulation is always developed by choosing *the minimum number of decision variables*. In the context of MPC, this is labeled as the "standard approach" [9], where (usually) only the control inputs are considered as variables, while the equality constraints due to the system dynamics are eliminated. Even though this reduces the size of the problem to be solved, the constraints and Hessian matrix are (in general) dense, $i.e.$, the structure of the problem is lost. When solving such a dense QP, the computational cost per iteration is $O(N^3)$ (if using an *interior-point* method) [9] and $O(N^2)$ (if using an *active-set* method), with N being the number of sampling times in the preview window. This approach performs well, provided that N is "relatively small". When using a dense formulation, apart from the problem of solving the QP, in many cases forming the Hessian matrix and the vector of the objective function is computationally expensive and is usually performed off-line. Depending on the particular setting, this might be entirely reasonable ($e.g.$, when the Hessian is constant). However, in general (and in the context of the formulation that we will discuss) this poses unnecessary limitations. For example, if the scheme in [4] is used, each distinct value for the desired altitude c^z of the CoM (above the flat floor) results in a distinct Hessian matrix. It can be precomputed offline only when the desired values of c^z are known in advance. If this is not the case, the Hessian has to be formed online, which (for this particular example) could turn out to be more expensive than solving the actual QP afterwards. In order to emphasize the "two step" approach (forming, then solving) associated with the dense formulation, in [10] it is a called a "*sequential*" approach to MPC. The dense formulation is usually appealing to practitioners, because in many cases it is possible to directly use off-the-shelf dense solvers. In contrast, exploiting the structure of the problem in the sparse formulation usually requires writing customized code.

A well known strategy for overcoming the limitations of the *sequential* approach, is to have the equality constraints due to the dynamics of the system explicitly appear in the formulation of the problem (see Section II). This results in a larger but more structured QP. In [10] this formulation is referred to as "*simultaneous*" approach to MPC, and its solution can be obtained at a cost of $O(N)$ per iteration [9]. Note that efficient algorithms that (in practice) require a number of iterations only weakly related to N are readily available (see Section V).

In this paper we present a sparse MPC formulation for walking motion generation, adopting the *simultaneous* approach, and analyze its advantages over the already proposed dense formulations in the context of humanoid walking. We derive an optimization problem that has a diagonal Hessian matrix and only simple bounds, while still retaining the possibility to generate motions for an omnidirectional walk. We point out that concepts already introduced with the dense formulation (like foot variation, and using alternative penalties in the objective function) are straightforward to adopt with the *simultaneous* approach.

The article is organized as follows: In Section II, we motivate the *simultaneous* approach, by considering first as an example a linear quadratic regulator (LQR) with discrete-time finite-horizon. In Sections III and IV we introduce our sparse formulation and analyze its advantages. Section V discusses the online solution of the underlying QP. Finally, Section VI compares the online computation time required by the dense and sparse formulations. For conciseness of notation, in some cases, we will use $\boldsymbol{x} = (x_1, \ldots, x_N)$ to denote the elements of a column vector $\boldsymbol{x}$.

II. A MOTIVATING EXAMPLE

The main difference between the *sequential* and *simultaneous* approaches can be emphasized using a simple example of a discrete-time finite-horizon LQR. Consider a discrete-time system

$$\begin{aligned} &\boldsymbol{x}_{k+1} = \mathbb{A}\boldsymbol{x}_k + \mathbb{B}\boldsymbol{u}_k, \quad k = 0, \ldots, N-1, \\ &\boldsymbol{x}_0 \text{ is a known initial state.} \end{aligned}$$

Define a quadratic cost function

$$\begin{aligned} J(\boldsymbol{v}_x, \boldsymbol{v}_u) &= \boldsymbol{v}_x^T \boldsymbol{H}_x \boldsymbol{v}_x + \boldsymbol{v}_u^T \boldsymbol{H}_u \boldsymbol{v}_u, \\ &:= \sum_{k=1}^{N-1} \boldsymbol{x}_k^T \boldsymbol{Q} \boldsymbol{x}_k + \boldsymbol{x}_N^T \boldsymbol{Q}_f \boldsymbol{x}_N + \sum_{k=0}^{N-1} \boldsymbol{u}_k^T \boldsymbol{P} \boldsymbol{u}_k, \end{aligned} \tag{1}$$

where $\boldsymbol{v}_u = (\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{N-1})$ is a column vector containing the control inputs $\boldsymbol{u}_k \in \mathbb{R}^m$, $\boldsymbol{v}_x = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ is a column vector containing the states $\boldsymbol{x}_k \in \mathbb{R}^n$. $\boldsymbol{Q}$ and $\boldsymbol{Q}_f$ are symmetric and positive semidefinite matrices that represent state cost, and final state cost, respectively, while $\boldsymbol{P}$ is a symmetric positive definite matrix representing the input cost. $\boldsymbol{Q}$, $\boldsymbol{Q}_f$ and $\boldsymbol{P}$ are assumed to be given. We want to choose $\boldsymbol{v}_x$ and $\boldsymbol{v}_u$ (starting from $\boldsymbol{x}_0$), so that $J(\boldsymbol{v}_x, \boldsymbol{v}_u)$ is minimized. There are multiple ways to solve this problem. One approach is to express $\boldsymbol{v}_x$ as a function of $\boldsymbol{v}_u$ and eliminate it from (1) as follows.

$$\boldsymbol{v}_x = \underbrace{\begin{bmatrix} \mathbb{A} \\ \mathbb{A}^2 \\ \vdots \\ \mathbb{A}^N \end{bmatrix}}_{\boldsymbol{w}} \boldsymbol{x}_0 + \underbrace{\begin{bmatrix} \mathbb{B} & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbb{A}\mathbb{B} & \mathbb{B} & \ldots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{A}^{N-1}\mathbb{B} & \mathbb{A}^{N-2}\mathbb{B} & \ldots & \mathbb{B} \end{bmatrix}}_{\boldsymbol{W}} \boldsymbol{v}_u,$$

$$\tilde{J}(\boldsymbol{v}_u) = \boldsymbol{v}_u^T \boldsymbol{H} \boldsymbol{v}_u + \boldsymbol{v}_u^T \tilde{\boldsymbol{w}}, \tag{2}$$

where $\boldsymbol{H} = \boldsymbol{W}^T \boldsymbol{H}_x \boldsymbol{W} + \boldsymbol{H}_u$, $\tilde{\boldsymbol{w}} = 2\boldsymbol{W}^T \boldsymbol{H}_x \boldsymbol{w} \boldsymbol{x}_0$. Some drawbacks of this approach are: (i) forming the matrix $\boldsymbol{H}$ requires two matrix-matrix multiplications, (ii) $\boldsymbol{H}$ is in general dense, hence without any structure to exploit, the cost of minimizing (2) grows like N^3.

An alternative approach is to directly minimize (1), subject to the system dynamics, $i.e.$,

$$\begin{aligned} \underset{\boldsymbol{v}_x, \boldsymbol{v}_u}{\text{minimize}} \quad & \begin{bmatrix} \boldsymbol{v}_x \\ \boldsymbol{v}_u \end{bmatrix}^T \begin{bmatrix} \boldsymbol{H}_x & \mathbf{0} \\ \mathbf{0} & \boldsymbol{H}_u \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_x \\ \boldsymbol{v}_u \end{bmatrix} \\ \text{subject to} \quad & \boldsymbol{x}_{k+1} = \mathbb{A}\boldsymbol{x}_k + \mathbb{B}\boldsymbol{u}_k, \quad k = 0, \ldots, N-1, \\ & \boldsymbol{x}_0 \text{ is a known initial state.} \end{aligned}$$

This is a problem with $(n+m)N$ variables subject to nN equality constraints, however, as is well known [9], [11] pp. 553, by exploiting the structure of the Hessian matrix and constraints, the computational complexity grows linearly with N. The same complexity can be demonstrated by using the Riccati recursion or dynamic programming. We will revisit this issue in Section V.

III. SPARSE MPC FORMULATION

In order to present the major ideas as clearly as possible, and for notation simplicity, we leave some special cases aside. In particular, we assume that double support constraints are modeled as rectangular polygons (for justification, and more details see [2]).

A. The approximate model

We use the 3D linear inverted pendulum [12], constrained to move on a horizontal plane with height c^z, as an approximate model of a humanoid robot. Consider the following linear dynamical system [13] for $k = 0, \ldots, N-1$

$$\begin{aligned} &\hat{\boldsymbol{c}}_{k+1} = \mathbb{A}_k \hat{\boldsymbol{c}}_k + \mathbb{B}_k \dddot{\boldsymbol{c}}_k, \\ &\boldsymbol{z}_{k+1} = \mathbb{C}_{k+1} \hat{\boldsymbol{c}}_{k+1}, \\ &\hat{\boldsymbol{c}}_0 \text{ is a known initial state,} \end{aligned} \tag{3}$$

where $\boldsymbol{c}_k = (c_k^x, c_k^y)$ and $\boldsymbol{z}_k = (z_k^x, z_k^y)$ are the coordinates of the CoM and the zero moment point (ZMP) on the flat floor, during the k^{th} sampling time of the preview window, $\hat{\boldsymbol{c}}_k = (c_k^x, \dot{c}_k^x, \ddot{c}_k^x, c_k^y, \dot{c}_k^y, \ddot{c}_k^y)$.

$$\mathbb{A}_k = \begin{bmatrix} 1 & T_k & T_k^2/2 & 0 & 0 & 0 \\ 0 & 1 & T_k & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & T_k & T_k^2/2 \\ 0 & 0 & 0 & 0 & 1 & T_k \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbb{B}_k = \begin{bmatrix} T_k^3/6 & 0 \\ T_k^2/2 & 0 \\ T_k & 0 \\ 0 & T_k^3/6 \\ 0 & T_k^2/2 \\ 0 & T_k \end{bmatrix},$$

$$\mathbb{C}_k = \begin{bmatrix} 1 & 0 & -h_k & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -h_k \end{bmatrix},$$

$$\mathbb{C}_p = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}, \quad \mathbb{C}_v = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix},$$

$h_k = c_k^z/g$, with g being the norm of the acceleration due to gravity ($e.g.$, $g \approx 9.8\ m/s^2$), and T_k is the length of the k^{th} sampling time in the preview window. We defined $\mathbb{C}_p$ and $\mathbb{C}_v$ for future reference. For this system, the standard ($i.e.$, dense) MPC formulation can be found in [2] (where the sampling time and c_k^z are assumed to be constant).

B. Sparse formulation (objective function)

Consider the following objective function (which is the same as the one in [2], equation (14))

$$\begin{aligned} f(\boldsymbol{v}) &= \frac{\gamma}{2} \sum_{k=0}^{N-1} \|\dddot{\boldsymbol{c}}_k\|^2 + \frac{\alpha}{2} \sum_{k=1}^{N} \|\dot{\boldsymbol{c}}_k\|^2 + \frac{\beta}{2} \sum_{k=1}^{N} \|\boldsymbol{z}_k - \boldsymbol{z}_k^{\text{ref}}\|^2 \\ &:= \frac{\gamma}{2} \sum_{k=0}^{N-1} \left(\dddot{\boldsymbol{c}}_k^T \dddot{\boldsymbol{c}}_k\right) + \frac{\alpha}{2} \sum_{k=1}^{N} \left(\dot{\boldsymbol{c}}_k^T \dot{\boldsymbol{c}}_k\right) + \frac{\beta}{2} \sum_{k=1}^{N} \left(\boldsymbol{z}_k^T \boldsymbol{z}_k - 2\boldsymbol{z}_k^T \boldsymbol{z}_k^{\text{ref}}\right), \end{aligned} \tag{4}$$

where $\alpha, \beta, \gamma > 0$ are gains, and the constant quadratic term in $\boldsymbol{z}_k^{\text{ref}}$ (which is the reference ZMP) was dropped. The variable $\boldsymbol{v} \in \mathbb{R}^p$ is defined as $\boldsymbol{v} = (\boldsymbol{v}_c, \boldsymbol{v}_u)$, $\boldsymbol{v}_c = (\hat{\boldsymbol{c}}_1, \ldots, \hat{\boldsymbol{c}}_N)$, $\boldsymbol{v}_u = (\dddot{\boldsymbol{c}}_0, \ldots, \dddot{\boldsymbol{c}}_{N-1})$, where $p = (n+m)N = (6+2)N$.

Using

$$\frac{\alpha}{2}\dot{\boldsymbol{c}}_k^T\dot{\boldsymbol{c}}_k = \hat{\boldsymbol{c}}_k^T\underbrace{\frac{\alpha}{2}\mathbb{C}_v^T\mathbb{C}_v}_{\boldsymbol{Q}_v}\hat{\boldsymbol{c}}_k,$$

$$\frac{\beta}{2}\boldsymbol{z}_k^T\boldsymbol{z}_k - \beta\boldsymbol{z}_k^T\boldsymbol{z}_k^{\text{ref}} = \hat{\boldsymbol{c}}_k^T\underbrace{\frac{\beta}{2}\mathbb{C}_k^T\mathbb{C}_k}_{\boldsymbol{Q}_{z_k}}\hat{\boldsymbol{c}}_k - \hat{\boldsymbol{c}}_k^T\underbrace{\beta\mathbb{C}_k^T\boldsymbol{z}_k^{\text{ref}}}_{\boldsymbol{q}_k},$$

$$\boldsymbol{P} = \frac{\gamma}{2}\boldsymbol{I}, \quad \boldsymbol{Q}_k = \boldsymbol{Q}_v + \boldsymbol{Q}_{z_k},$$

where $\boldsymbol{I}$ denotes the identity matrix, the objective function in (4) can be expressed in the following compact way

$$f(\boldsymbol{v}) = \begin{bmatrix}\boldsymbol{v}_c\\ \boldsymbol{v}_u\end{bmatrix}^T\begin{bmatrix}\boldsymbol{H}_c & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{H}_u\end{bmatrix}\begin{bmatrix}\boldsymbol{v}_c\\ \boldsymbol{v}_u\end{bmatrix} + \begin{bmatrix}\boldsymbol{v}_c\\ \boldsymbol{v}_u\end{bmatrix}^T\begin{bmatrix}\boldsymbol{g}_c\\ \boldsymbol{0}\end{bmatrix},$$

$$\boldsymbol{H}_c = \begin{bmatrix}\boldsymbol{Q}_1 & \dots & \boldsymbol{0}\\ \vdots & \ddots & \vdots\\ \boldsymbol{0} & \dots & \boldsymbol{Q}_N\end{bmatrix}, \quad \boldsymbol{H}_u = \begin{bmatrix}\boldsymbol{P} & \dots & \boldsymbol{0}\\ \vdots & \ddots & \vdots\\ \boldsymbol{0} & \dots & \boldsymbol{P}\end{bmatrix},$$

where $\boldsymbol{g}_c = (-\boldsymbol{q}_1, \dots, -\boldsymbol{q}_N)$. Note that $\boldsymbol{H}_u$ is a diagonal matrix, while $\boldsymbol{H}_c$ is a block diagonal (and variable) matrix.

C. Sparse formulation (constraints)

As pointed out in the example in Section II, instead of eliminating the states (and leaving only the control inputs as decision variables), one can minimize the objective function over both $\boldsymbol{v}_c$ and $\boldsymbol{v}_u$ subject to the equality constraints due to the system dynamics (3). The equality constraints can be expressed in a matrix form as

$$\boldsymbol{E}_c\boldsymbol{v}_c + \boldsymbol{E}_u\boldsymbol{v}_u = \boldsymbol{e},$$

$$\boldsymbol{E}_c = \begin{bmatrix}-\boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{0}\\ \mathbb{A}_1 & -\boldsymbol{I} & \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{0}\\ \boldsymbol{0} & \mathbb{A}_2 & -\boldsymbol{I} & \dots & \boldsymbol{0} & \boldsymbol{0}\\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots\\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \dots & \mathbb{A}_{N-1} & -\boldsymbol{I}\end{bmatrix},$$

$$\boldsymbol{E}_u = \begin{bmatrix}\mathbb{B}_0 & \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{0}\\ \boldsymbol{0} & \mathbb{B}_1 & \boldsymbol{0} & \dots & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{0} & \mathbb{B}_2 & \dots & \boldsymbol{0}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \dots & \mathbb{B}_{N-1}\end{bmatrix}, \quad \boldsymbol{e} = \begin{bmatrix}-\mathbb{A}_0\hat{\boldsymbol{c}}_0\\ \boldsymbol{0}\\ \boldsymbol{0}\\ \vdots\\ \boldsymbol{0}\end{bmatrix}.$$

Apart from the equality constraints, we have to define inequality constraints that limit the motion of the ZMP to be within a given polygon on the flat floor. Let $\boldsymbol{R}_k \in \mathbb{R}^{2\times 2}$ and $\boldsymbol{r}_k \in \mathbb{R}^2$ denote a rotation matrix and position vector defining the orientation and position of a rectangular *polygon of support* appearing in the k^{th} sampling time of the preview window. Define the constraint for $\boldsymbol{z}_k$ as

$$\boldsymbol{D}_z\boldsymbol{R}_k^T(\boldsymbol{z}_k - \boldsymbol{r}_k) \le \boldsymbol{d}_z,$$

where, $\boldsymbol{D}_z = \begin{bmatrix}\boldsymbol{I} & -\boldsymbol{I}\end{bmatrix}^T \in \mathbb{R}^{4\times 2}$ and $\boldsymbol{d}_z$ is a constant vector reflecting the size of the polygon of support (for more details, and additional inequality constraints that can be considered see [2]). By rearranging terms, and using $\boldsymbol{z}_k = \mathbb{C}_k\hat{\boldsymbol{c}}_k$ we obtain

$$\boldsymbol{D}_z\boldsymbol{R}_k^T\mathbb{C}_k\hat{\boldsymbol{c}}_k \le \underbrace{\boldsymbol{d}_z + \boldsymbol{D}_z\boldsymbol{R}_k^T\boldsymbol{r}_k}_{\boldsymbol{d}_k}. \tag{5}$$

The following QP can be used to perform stabilization of a humanoid robot along a given reference profile for the ZMP

$$\begin{aligned}&\underset{\boldsymbol{v}}{\text{minimize}} \;\; f(\boldsymbol{v}) \\ &\text{subject to} \;\; \boldsymbol{E}_c\boldsymbol{v}_c + \boldsymbol{E}_u\boldsymbol{v}_u = \boldsymbol{e},\\ &\qquad\quad \boldsymbol{D}_z\boldsymbol{R}_k^T\mathbb{C}_k\hat{\boldsymbol{c}}_k \le \boldsymbol{d}_k, \quad k = 1, \dots, N.\end{aligned} \tag{6}$$

D. Change of variable (diagonal Hessian)

In order to obtain a formulation with a diagonal Hessian (which is computationally more attractive, see Section V), we interchange $\boldsymbol{c}_k$ with $\boldsymbol{z}_k$ in the state vector of (3), to obtain the following dynamical system for $k = 0, \dots, N-1$

$$\begin{aligned}&\tilde{\boldsymbol{c}}_{k+1} = \tilde{\mathbb{A}}_k\tilde{\boldsymbol{c}}_k + \tilde{\mathbb{B}}_k\dddot{\boldsymbol{c}}_k,\\ &\boldsymbol{z}_{k+1} = \mathbb{C}_p\tilde{\boldsymbol{c}}_{k+1},\\ &\tilde{\boldsymbol{c}}_0 \text{ is a known initial state.}\end{aligned} \tag{7}$$

$$\tilde{\mathbb{A}}_k = \begin{bmatrix}1 & T_k & T_k^2/2 - \Delta h_k & 0 & 0 & 0\\ 0 & 1 & T_k & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & T_k & T_k^2/2 - \Delta h_k\\ 0 & 0 & 0 & 0 & 1 & T_k\\ 0 & 0 & 0 & 0 & 0 & 1\end{bmatrix},$$

$$\tilde{\mathbb{B}}_k = \begin{bmatrix}T_k^3/6 - h_{k+1}T_k & 0\\ T_k^2/2 & 0\\ T_k & 0\\ 0 & T_k^3/6 - h_{k+1}T_k\\ 0 & T_k^2/2\\ 0 & T_k\end{bmatrix}, \quad \tilde{\boldsymbol{c}}_k = \begin{bmatrix}z_k^x\\ \dot{c}_k^x\\ \ddot{c}_k^x\\ z_k^y\\ \dot{c}_k^y\\ \ddot{c}_k^y\end{bmatrix}.$$

Where $\Delta h_k = h_{k+1} - h_k$. With the above change of variable, the objective function is given by

$$\tilde{f}(\tilde{\boldsymbol{v}}) = \begin{bmatrix}\tilde{\boldsymbol{v}}_c\\ \boldsymbol{v}_u\end{bmatrix}^T\underbrace{\begin{bmatrix}\tilde{\boldsymbol{H}}_c & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{H}_u\end{bmatrix}}_{\boldsymbol{H}}\begin{bmatrix}\tilde{\boldsymbol{v}}_c\\ \boldsymbol{v}_u\end{bmatrix} + \begin{bmatrix}\tilde{\boldsymbol{v}}_c\\ \boldsymbol{v}_u\end{bmatrix}^T\begin{bmatrix}\tilde{\boldsymbol{g}}_c\\ \boldsymbol{0}\end{bmatrix},$$

where $\tilde{\boldsymbol{g}}_c = (-\tilde{\boldsymbol{q}}_1, \dots, -\tilde{\boldsymbol{q}}_N)$,

$$\tilde{\boldsymbol{H}}_c = \begin{bmatrix}\tilde{\boldsymbol{Q}} & \dots & \boldsymbol{0}\\ \vdots & \ddots & \vdots\\ \boldsymbol{0} & \dots & \tilde{\boldsymbol{Q}}\end{bmatrix}, \quad \tilde{\boldsymbol{v}}_c = \begin{bmatrix}\tilde{\boldsymbol{c}}_1\\ \vdots\\ \tilde{\boldsymbol{c}}_N\end{bmatrix}, \quad \tilde{\boldsymbol{v}} = \begin{bmatrix}\tilde{\boldsymbol{v}}_c\\ \boldsymbol{v}_u\end{bmatrix},$$

$$\tilde{\boldsymbol{Q}}_z = \frac{\beta}{2}\mathbb{C}_p^T\mathbb{C}_p, \quad \tilde{\boldsymbol{q}}_k = \beta\mathbb{C}_p^T\boldsymbol{z}_k^{\text{ref}}, \quad \tilde{\boldsymbol{Q}} = \boldsymbol{Q}_v + \tilde{\boldsymbol{Q}}_z. \tag{8}$$

Note that the new Hessian matrix is constant and has nonzero entries only on the main diagonal. The equality constraints are given by

$$\tilde{\boldsymbol{E}}_c\tilde{\boldsymbol{v}}_c + \tilde{\boldsymbol{E}}_u\boldsymbol{v}_u = \tilde{\boldsymbol{e}},$$

where the structure of $\tilde{\boldsymbol{E}}_c$, $\tilde{\boldsymbol{E}}_u$ and $\tilde{\boldsymbol{e}}$ is identical to that of $\boldsymbol{E}_c$, $\boldsymbol{E}_u$ and $\boldsymbol{e}$, however, $\mathbb{A}_k$, $\mathbb{B}_k$ and $\hat{\boldsymbol{c}}_0$ are interchanged with $\tilde{\mathbb{A}}_k$, $\tilde{\mathbb{B}}_k$ and $\tilde{\boldsymbol{c}}_0$. The inequality constraints in (5) become

$$\boldsymbol{D}_z\boldsymbol{R}_k^T\mathbb{C}_p\tilde{\boldsymbol{c}}_k \le \boldsymbol{d}_k. \tag{9}$$

E. Change of variable (simple bounds)

By a second change of variable, we can express the general inequality constraints in (9) as simple bounds. Define the following rotation matrix ($c_{\theta_k} = \cos\theta_k$, $s_{\theta_k} = \sin\theta_k$, where θ_k is the angle, with respect to the world frame, of the support polygon appearing in the k^{th} sampling time)

$$\bar{\boldsymbol{R}}_k = \begin{bmatrix} c_{\theta_k} & 0 & 0 & -s_{\theta_k} & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ s_{\theta_k} & 0 & 0 & c_{\theta_k} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Define $\bar{\boldsymbol{c}}_k = \bar{\boldsymbol{R}}_k^T \tilde{\boldsymbol{c}}_k$, hence $\tilde{\boldsymbol{c}}_k = \bar{\boldsymbol{R}}_k \bar{\boldsymbol{c}}_k$. Performing a change of variable in (7), leads to

$$\bar{\boldsymbol{c}}_{k+1} = \bar{\boldsymbol{R}}_{k+1}^T \tilde{\mathbb{A}}_k \bar{\boldsymbol{R}}_k \bar{\boldsymbol{c}}_k + \bar{\boldsymbol{R}}_{k+1}^T \tilde{\mathbb{B}}_k \dddot{\bar{\boldsymbol{c}}}_k. \tag{10}$$

The objective function becomes $\bar{f}(\bar{\boldsymbol{v}}) = \bar{\boldsymbol{v}}^T \boldsymbol{H} \bar{\boldsymbol{v}} + \bar{\boldsymbol{v}} \boldsymbol{g}$, where $\boldsymbol{g} = (\bar{\boldsymbol{g}}_c, \boldsymbol{0})$, $\bar{\boldsymbol{q}}_k = \beta \bar{\boldsymbol{R}}_k^T \mathbb{C}_p^T \boldsymbol{z}_k^{\text{ref}}$, and

$$\bar{\boldsymbol{v}}_c = \begin{bmatrix} \bar{\boldsymbol{c}}_1 \\ \vdots \\ \bar{\boldsymbol{c}}_N \end{bmatrix}, \quad \bar{\boldsymbol{g}}_c = \begin{bmatrix} -\bar{\boldsymbol{q}}_1 \\ \vdots \\ -\bar{\boldsymbol{q}}_N \end{bmatrix}, \quad \bar{\boldsymbol{v}} = \begin{bmatrix} \bar{\boldsymbol{v}}_c \\ \boldsymbol{v}_u \end{bmatrix}.$$

The equality constraints are given by $\bar{\boldsymbol{E}}_c \bar{\boldsymbol{v}}_c + \tilde{\boldsymbol{E}}_u \boldsymbol{v}_u = \bar{\boldsymbol{e}}$, where $\bar{\boldsymbol{e}} = (-\tilde{\mathbb{A}}_0 \bar{\boldsymbol{R}}_0 \bar{\boldsymbol{c}}_0, \boldsymbol{0}, \ldots, \boldsymbol{0})$, with

$$\bar{\boldsymbol{E}}_c = \begin{bmatrix} -\bar{\boldsymbol{R}}_1 & \boldsymbol{0} & \boldsymbol{0} & \ldots & \boldsymbol{0} & \boldsymbol{0} \\ \tilde{\mathbb{A}}_1 \bar{\boldsymbol{R}}_1 & -\bar{\boldsymbol{R}}_2 & \boldsymbol{0} & \ldots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \tilde{\mathbb{A}}_2 \bar{\boldsymbol{R}}_2 & -\bar{\boldsymbol{R}}_3 & \ldots & \boldsymbol{0} & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \ldots & \tilde{\mathbb{A}}_{N-1} \bar{\boldsymbol{R}}_{N-1} & -\bar{\boldsymbol{R}}_N \end{bmatrix}.$$

The inequality constraints become

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \end{bmatrix} \bar{\boldsymbol{c}}_k \leq \boldsymbol{d}_k,$$

which are simple bounds for $\bar{\boldsymbol{z}}_k$, where $\bar{\boldsymbol{z}}_k \in \mathbb{R}^2$ contains the first and fourth components of $\bar{\boldsymbol{c}}_k$. To this end we obtain the following formulation with diagonal Hessian matrix and simple bounds $\boldsymbol{\ell}, \boldsymbol{u} \in \mathbb{R}^p$

$$\begin{aligned} \underset{\bar{\boldsymbol{v}}}{\text{minimize}} \;\; & \bar{f}(\bar{\boldsymbol{v}}) \\ \text{subject to} \;\; & \boldsymbol{E}\bar{\boldsymbol{v}} = \bar{\boldsymbol{E}}_c \bar{\boldsymbol{v}}_c + \tilde{\boldsymbol{E}}_u \boldsymbol{v}_u = \bar{\boldsymbol{e}}, \\ & -\boldsymbol{\ell} \leq \bar{\boldsymbol{v}} \leq \boldsymbol{u}, \\ & \bar{\boldsymbol{c}}_0 \text{ is a given initial condition,} \end{aligned} \tag{11}$$

If $\bar{v}_k$ is not subject to bounds, $u_k = -\ell_k = \infty$ is assumed.

IV. Sparse formulation - implications

After the change of variable in both Sections III-D, and III-E, the parameters that influence the dynamical system (3), T_k and h_k, appear only in the equality constraints, while in the dense formulations in [2]-[8] they appear either in the objective function, in the constraints, or in both. Next, we discuss some of the resulting implications.

A. Nonuniform time discretization

Being able to use a variable time discretization of the preview window increases the flexibility of the MPC scheme, as we argue next.

To be concrete, let us consider a humanoid robot that has a control sampling rate of 2 ms. This implies that every 2 ms a control input has to be provided to the system. If 1.5 sec long preview window is considered, one option is to have a constant discretization of $T_k = 2$ ms, which leads to $N = 750$. Such a QP can not be solved within 2 ms (at present). As a workaround, it is common to consider $T_k = 20$ ms, which leads to $N = 75$. This is a tractable problem, however, after obtaining $\bar{\boldsymbol{c}}_0$, interpolation has to be performed (in order to generate the actual control input to be applied to the system). Note that using a discretization of $\{2, 20, 20, \ldots, 20\}$ (all in ms) for each preview window is not desirable, because depending on the particular footsteps envisioned to be made within the current preview window, it might turn out that decision variables are not dedicated to the switching between single and double support for example. Other options, based on imposing additional equality constraints [7], or designing a number of discretization sequences (to be applied in turn), exist. When using a dense formulation, the latter option would require the pre-computation (and storage) of a number of Hessian (and other) matrices to be interchanged online. In our opinion, this complicates the code and is not flexible. On the other hand, when using formulation (11), each computation can be performed with the most suitable discretization at a negligible additional computational cost.

Nonuniform discretization is not only useful when dealing with the interpolation problem mentioned above. In some cases it could be desirable to dedicate more sampling times ($i.e.$, decision variables) in a particular time in the preview window because, for example, we model a disturbance expected in the future. On the other hand we could decrease the overall number of decision variables by having fewer variables dedicated to other, "less important", parts of the preview window. The discretization issues become even more interesting when foot variation is allowed, see Section IV-C.

B. Variable CoM height

An interesting option when using the sparse formulation is the potential ability to change the height of the CoM c_k^z. We are aware that variation of c_k^z while the robot is walking would violate the assumptions under which the model (3) is derived. Nevertheless, in practice, if the vertical acceleration of the CoM is relatively small, and due to the safety margin usually used when specifying the polygons that model the feet of the robot, it appears that small variations of c_k^z can be treated simply as disturbances to the system. This has been recognized by other researchers as well [14]. In [1], the height of the CoM is altered in order to perform "knee singularity avoidance". Such an option gives additional flexibility to the scheme. Indeed, when using the sparse formulation, if it is desirable to change the height of the CoM, different c_k^z can be defined for different sampling

times in one preview window (without having to pre-compute anything offline).

C. Foot variation

Including the possibility for the QP to calculate a deviation from the predefined footsteps (in the presence of strong disturbances) is equally straightforward with the sparse formulation. The notion of foot repositioning was first proposed in [6]. The basic idea is to introduce new variables (apart from $\boldsymbol{v}_c$ and $\boldsymbol{v}_u$) that relax the ZMP inequality constraints. This leads to

$$\boldsymbol{D}_z \boldsymbol{R}_k^T \left(\boldsymbol{z}_k - \Delta \boldsymbol{r}_i \right) \leq \boldsymbol{d}_k,$$

in which the i^{th} additional variable $\Delta \boldsymbol{r}_i$ appears in the k^{th} sampling iteration. In order to follow as much as possible the predefined footsteps, the $\Delta \boldsymbol{r}_i$'s are penalized [2]. If a quadratic penalty $\mu \Delta \boldsymbol{r}_i^T \Delta \boldsymbol{r}_i$ is considered (with $\mu > 0$ being a gain), the diagonal structure of the Hessian is preserved, however, one has to consider more general constraints than the simple bounds in (11). As an alternative, it is possible to perform a change of variable $\boldsymbol{w}_k = \boldsymbol{z}_k - \Delta \boldsymbol{r}_i$, which would bring back the simple bounds, however, would introduce coupling between $\boldsymbol{w}_k$ and $\Delta \boldsymbol{r}_i$ in the Hessian matrix. Therefore, from a computational point of view, the price to pay when considering variable feet is either loosing the simple bounds or the diagonal structure of the Hessian matrix.

Note that it is possible to derive a dense formulation which has simple bounds (even when foot variation is considered), however, this would result in a completely dense Hessian matrix, which is variable with the preview window (forming it requires a matrix-matrix multiplication to be performed). On the other hand, a dense formulation with a trivial Hessian matrix can be derived by performing a change of variable as described in [5], however, in this case the constraints would become completely dense.

V. Solving the underlying QP

A. Generation of a feasible point

Both methods discussed in this Section can benefit from a feasible initial point. Here, we demonstrate how to generate one at a negligible cost. A similar approach has already been proposed in the context of the dense formulation [7].

First, we note that the problem we are solving is always feasible, since the constraints for $\boldsymbol{z}_k$, to be within its corresponding rectangular polygon, are always consistent (by construction). Hence, one can choose feasible profile for the ZMP (with respect to the inequality constraints) and then recursively identify the remaining entries of $\bar{\boldsymbol{v}}$ (so that they satisfy the equality constraints). The following recursion generates a feasible $\bar{\boldsymbol{v}}$.

$$\begin{aligned} \dddot{\tilde{\boldsymbol{c}}}_k &= -\left(\mathbb{C}_p \tilde{\mathbb{B}}_k \right)^{-1} \mathbb{C}_p \tilde{\mathbb{A}}_k \tilde{\boldsymbol{c}}_k + \left(\mathbb{C}_p \tilde{\mathbb{B}}_k \right)^{-1} \boldsymbol{z}_{k+1}^F \\ \tilde{\boldsymbol{c}}_{k+1} &= \tilde{\mathbb{A}}_k \tilde{\boldsymbol{c}}_k + \tilde{\mathbb{B}}_k \ddot{\tilde{\boldsymbol{c}}}_k, \quad k = 0, \ldots, N-1, \end{aligned}$$

where $\tilde{\boldsymbol{c}}_0$ is given and $\boldsymbol{z}_{k+1}^F$ is a $\boldsymbol{z}_{k+1}$ that satisfies (9) ($e.g.$, it could be defined to be in the center of each support polygon). In order to obtain $\bar{\boldsymbol{c}}_k$, a change of variable $\bar{\boldsymbol{c}}_k = \bar{\boldsymbol{R}}_k^T \tilde{\boldsymbol{c}}_k$ is performed for $i = 1, \ldots, N$. Note that $\mathbb{C}_p \tilde{\mathbb{B}}_k$ is a diagonal matrix which is invertible if $T_k^3/6 - h_{k+1} T_k \neq 0$ (which is the case in almost all practical applications).

B. Solution strategies

The major difference between solving the LQR problem in Section II and solving (11) is the presence of inequality constraints in the latter. Two approaches for handling inequality constraints (popular in the context of MPC) are outlined next. The first one, an *interior-point* method, is particularly well suited to problems having relatively many active constraints (constraints that hold as equalities) at the solution. Furthermore, solvers based on *interior-point* methods, tend to require a number of iterations only weakly related to N [11]. This is relevant to the problem of motion generation, as in the presence of strong disturbances many constraints are usually activated. The second approach, an *active-set* method, is particularly well suited to small and medium sized problems with relatively few active constraints (this is the method commonly used in practice in the context of the current application [1]).

C. Interior-point method

One popular approach for accounting for the inequality constraints is the modification of $\bar{f}(\bar{\boldsymbol{v}})$, and in turn $\nabla \bar{f}(\bar{\boldsymbol{v}})$. Consider the following problem

$$\begin{aligned} \underset{\bar{\boldsymbol{v}},\, \boldsymbol{\ell} < \bar{\boldsymbol{v}} < \boldsymbol{u}}{\text{minimize}} \quad & \bar{f}_\phi(\bar{\boldsymbol{v}}) = \bar{f}(\bar{\boldsymbol{v}}) + \kappa \phi(\bar{\boldsymbol{v}}) \\ \text{subject to} \quad & \boldsymbol{E} \bar{\boldsymbol{v}} = \bar{\boldsymbol{e}}, \end{aligned} \tag{12}$$

where $\phi(\bar{\boldsymbol{v}}) = -\sum_{i=1}^p \log(u_i - \bar{v}_i) - \sum_{i=1}^p \log(\bar{v}_i - \ell_i)$, $\kappa > 0$ is a penalty factor (commonly referred to as *barrier parameter*), and $\log(a)$ is the natural logarithm of a scalar a ($a > 0$ is implicitly assumed). Note that $\boldsymbol{\ell} < \bar{\boldsymbol{v}} < \boldsymbol{u}$ is an implicit constraint ($\bar{\boldsymbol{v}}$ that satisfies it is referred to as *strictly* feasible). Since $\log(0) = -\infty$, if any of the inequality constraints holds as an equality, $\phi(\bar{\boldsymbol{v}})$ takes on the value ∞. Hence, moving away from a strictly feasible $\bar{\boldsymbol{v}}$ would be subject to "repelling forces" from the boundary of $\boldsymbol{\ell} \leq \bar{\boldsymbol{v}} \leq \boldsymbol{u}$. For more interpretations of/implications from the log-barrier method see [11], Chapter 11.

Note that (12) is a convex problem and $\phi(\bar{\boldsymbol{v}})$ is a continuously differentiable function. The dual residual of its (first-order) optimality conditions is given by

$$\boldsymbol{r}_d = \nabla \bar{f}_\phi(\bar{\boldsymbol{v}}, \kappa) + \boldsymbol{E}^T \boldsymbol{\nu} = \boldsymbol{0}.$$

This is a nonlinear function of $\bar{\boldsymbol{v}}$ and one possible way to find a solution is by using Newton's method. Performing linearization around a point $\bar{\boldsymbol{v}}$ (for a fixed value of κ), using $\nabla \bar{f}_\phi(\bar{\boldsymbol{v}} + \Delta \bar{\boldsymbol{v}}, \kappa) = \nabla \bar{f}_\phi(\bar{\boldsymbol{v}}, \kappa) + \nabla^2 \bar{f}_\phi(\bar{\boldsymbol{v}}, \kappa) \Delta \bar{\boldsymbol{v}}$ leads to

$$\begin{bmatrix} \boldsymbol{G}(\bar{\boldsymbol{v}}, \kappa) & \boldsymbol{E}^T \\ \boldsymbol{E} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \Delta \bar{\boldsymbol{v}} \\ \boldsymbol{\nu} \end{bmatrix} = \begin{bmatrix} -\nabla \bar{f}_\phi(\bar{\boldsymbol{v}}, \kappa) \\ \boldsymbol{0} \end{bmatrix}, \tag{13}$$

where $\boldsymbol{G}(\bar{\boldsymbol{v}}, \kappa)$ is the Hessian of $\bar{f}_\phi(\bar{\boldsymbol{v}}, \kappa)$, defined as

$$\boldsymbol{G}(\bar{\boldsymbol{v}}, \kappa) = \begin{bmatrix} \boldsymbol{M} & \boldsymbol{0} \\ \boldsymbol{0} & 2\boldsymbol{H}_u, \end{bmatrix},$$

$$\boldsymbol{M}(\bar{\boldsymbol{v}}) = 2\tilde{\boldsymbol{H}}_c + \kappa\,\mathrm{diag}\left(\frac{1}{u_1-\bar{v}_1},\ldots,\frac{1}{u_p-\bar{v}_p}\right) + \kappa\,\mathrm{diag}\left(\frac{1}{\bar{v}_1-\ell_1},\ldots,\frac{1}{\bar{v}_p-\ell_p}\right).$$

Minimizing (12) for a given value of κ, amounts to solving a number of linear systems of the form (13). Loosely speaking, for a very small κ, the solution $\bar{\boldsymbol{v}}^\star(\kappa)$ of (12) approximates very well the solution of the original problem (11). This is due to the fact that the "repelling forces" are scaled down by κ, and $\boldsymbol{v}^\star(\kappa)$ can approach more and more the boundary of the feasible set. If $\bar{\boldsymbol{v}}$ satisfies at least one of the inequalities as an equality, $\phi(\bar{\boldsymbol{v}}) = \infty$, and scaling with κ has no effect, hence, solutions of (12) belong to the interior of the feasible set. After $\Delta\bar{\boldsymbol{v}}$ is obtained, a step $\bar{\boldsymbol{v}}^+ = \bar{\boldsymbol{v}} + \tau\Delta\bar{\boldsymbol{v}}$ is performed, where τ is chosen so that $\boldsymbol{\ell} < \bar{\boldsymbol{v}}^+ < \boldsymbol{u}$, and the function value $\bar{f}_\phi(\bar{\boldsymbol{v}}^+)$ is sufficiently decreased (by using one of the many line search heuristics).

When κ is chosen very small, the Hessian $\boldsymbol{G}(\bar{\boldsymbol{v}},\kappa)$ varies rapidly near the boundary of the feasible domain and, unless a good starting point is available, a large number of (constrained) Newton steps (13) is required until convergence. In practice, this is addressed by solving a sequence of problems (12) with a decreasing value of the parameter κ, where each problem is initialized with the solution of the previous one. The first problem is initialized with a point that satisfies the equality and strictly satisfies the inequality constraints (see Section V-A). Solution methods that do not require a feasible initial point are popular as well in the context of MPC [15] (usually they require more iterations until convergence).

The solution of (13) can be obtained as follows:

1) form $\boldsymbol{S} = \boldsymbol{E}\boldsymbol{G}^{-1}\boldsymbol{E}^T$ and $\boldsymbol{s} = -\boldsymbol{E}\boldsymbol{G}^{-1}\nabla\bar{f}_\phi$,
2) solve $\boldsymbol{S}\boldsymbol{\nu} = \boldsymbol{s}$,
3) solve $\boldsymbol{G}\Delta\boldsymbol{v} = -\nabla\bar{f}_\phi - \boldsymbol{E}^T\boldsymbol{\nu}$.

It is straightforward to demonstrate that the KKT matrix (in the context of our application) in (13) is invertible, however, this does not mean that $\boldsymbol{G}$ is. In the above three steps we slightly abused our notation, since $\boldsymbol{G}$ is only positive semidefinite. Even though it is possible (but computationally more expensive) to perturb $\boldsymbol{G}$ so that it becomes invertible and the solution of (13) is unchanged (see [11], pp. 547), we perform a simple regularization by adding a "small" positive number ϵ to the zero entries on the main diagonal of $\tilde{\boldsymbol{Q}}$ in (8), which renders $\boldsymbol{G}$ invertible. Applying such regularization actually means adding the term $\epsilon\sum_{k=1}^{N}(\ddot{\boldsymbol{c}}_k^T\ddot{\boldsymbol{c}}_k)$ to the objective function. In our tests this has an insignificant effect on the computed control policy.

Considering the structure of $\boldsymbol{E}$ and $\boldsymbol{G}^{-1}$, it follows that $\boldsymbol{S}$ has the following block triangular form ($\boldsymbol{M}_{k,k} \in \mathbb{R}^{6\times 6}$ denotes the k^{th} diagonal block of $\boldsymbol{M}$)

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{S}_{11} & \boldsymbol{S}_{12} & \boldsymbol{0} & \ldots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{S}_{21} & \boldsymbol{S}_{22} & \boldsymbol{S}_{23} & \ldots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{S}_{32} & \boldsymbol{S}_{33} & \ldots & \boldsymbol{0} & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \ldots & \boldsymbol{S}_{N-1,N-1} & \boldsymbol{S}_{N-1,N} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \ldots & \boldsymbol{S}_{N,N-1} & \boldsymbol{S}_{NN} \end{bmatrix},$$

$$\begin{aligned} \boldsymbol{S}_{11} &= \bar{\boldsymbol{R}}_1\boldsymbol{M}_{1,1}^{-1}\bar{\boldsymbol{R}}_1^T + \frac{1}{2}\tilde{\mathbb{B}}_0\boldsymbol{P}^{-1}\tilde{\mathbb{B}}_0^T, \\ \boldsymbol{S}_{kk} &= \bar{\boldsymbol{R}}_k\boldsymbol{M}_{k,k}^{-1}\bar{\boldsymbol{R}}_k^T + \frac{1}{2}\tilde{\mathbb{B}}_{k-1}\boldsymbol{P}^{-1}\tilde{\mathbb{B}}_{k-1}^T + \\ &\qquad \tilde{\mathbb{A}}_{k-1}\bar{\boldsymbol{R}}_{k-1}\boldsymbol{M}_{k-1,k-1}^{-1}\bar{\boldsymbol{R}}_{k-1}^T\tilde{\mathbb{A}}_{k-1}^T, \\ \boldsymbol{S}_{k,k+1} &= \boldsymbol{S}_{k+1,k}^T = -\bar{\boldsymbol{R}}_k\boldsymbol{M}_{k,k}^{-1}\bar{\boldsymbol{R}}_k^T\mathbb{A}_k^T, \end{aligned}$$

The second step is carried out by forming the Cholesky factors $\boldsymbol{S} = \boldsymbol{L}\boldsymbol{L}^T$, with $\boldsymbol{L}$ being lower triangular

$$\boldsymbol{L} = \begin{bmatrix} \boldsymbol{L}_{11} & \boldsymbol{0} & \boldsymbol{0} & \ldots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{L}_{21} & \boldsymbol{L}_{22} & \boldsymbol{0} & \ldots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{L}_{32} & \boldsymbol{L}_{33} & \ldots & \boldsymbol{0} & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \ldots & \boldsymbol{L}_{N-1,N-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \ldots & \boldsymbol{L}_{N,N-1} & \boldsymbol{L}_{NN} \end{bmatrix}.$$

Directly from observation we have

$$\begin{aligned} &\boldsymbol{S}_{11} = \boldsymbol{L}_{11}\boldsymbol{L}_{11}^T, \\ &\boldsymbol{S}_{12} = \boldsymbol{S}_{21}^T = \boldsymbol{L}_{11}\boldsymbol{L}_{21}^T, \quad \text{hence} \quad \boldsymbol{L}_{21}^T = \boldsymbol{L}_{11}^{-1}\boldsymbol{S}_{12}, \\ &\boldsymbol{S}_{22} = \boldsymbol{L}_{21}\boldsymbol{L}_{21}^T + \boldsymbol{L}_{22}\boldsymbol{L}_{22}^T, \quad \text{and so on.} \end{aligned}$$

In the second step $\boldsymbol{L}_{21}^T$ is computed by forward substitution, and in the third step, forming $\boldsymbol{L}_{22}$ requires the computation of the Cholesky factors of $\boldsymbol{S}_{22} - \boldsymbol{L}_{21}\boldsymbol{L}_{21}^T$. Note that even though $n = 6$, because in our particular application the forward and lateral motions are decoupled (and have identical dynamics), in order to form $\boldsymbol{L}_{kk}$, one has to form the Cholesky factors of only one 3×3 matrix. The above factorization scheme is closely related to the Riccati recursion. After $\boldsymbol{L}$ is formed, a backward and forward substitution with $\boldsymbol{s}$ should be performed (accounting for the block structure).

The third step is not computationally demanding, since $\boldsymbol{G}$ is a diagonal matrix. Essentially, computing one constrained Newton step ($i.e.$, solving (13)) amounts to forming $\boldsymbol{L}$ which in this particular case can be done very efficiently due to the diagonal form of the Hessian matrix, the simple bounds and the inherently decoupled dynamics. Note that the complexity of each step is proportional to N. If foot variation is considered, due to the fact that the Hessian ceases to be diagonal, some modifications should be made (which would make the computation slightly more demanding [15]).

D. Active-set method

Let us consider again problem (11). What we did in Section V-C was to make a nonlinear approximation of the feasible set. This basically led to the need to solve a number of systems of linear equations of the form (13) *from scratch*, $i.e.$, the factorization of the KKT matrix is carried out separately for the solution of each system. This is due to the fact that the matrix $\boldsymbol{G}$ varies with $\bar{\boldsymbol{v}}$ (and κ).

Instead of a nonlinear approximation of the feasible set, another option is to perform a "*combinatorial approximation*". Or in other words, to repeatedly make a guess about the active constraints at the solution of problem (11), and then verify it by solving an equality constrained problem that resembles (13).

Algorithm 1: A modified primal active-set method

Input : Definition of problem (11), set $i \leftarrow 0$, initial guess $\mathcal{G}^{(i)} \leftarrow \emptyset$ and corresponding feasible $\bar{\boldsymbol{v}}^{(i)}$

Output: $\bar{\boldsymbol{v}}^{\star}$, an approximate solution of problem (11)

(1) Compute $\Delta\bar{\boldsymbol{v}}^{(i)}$ by solving

$$\begin{aligned} \underset{\bar{\boldsymbol{v}}^{(i)}}{\text{minimize}} \;& \Delta\bar{\boldsymbol{v}}^{(i)T}\boldsymbol{H}\Delta\bar{\boldsymbol{v}}^{(i)} + \Delta\bar{\boldsymbol{v}}^{(i)T}\nabla\bar{f}(\bar{\boldsymbol{v}}^{(i)}) \quad (14)\\ \text{subject to} \;& \boldsymbol{E}\Delta\bar{\boldsymbol{v}}^{(i)} = \boldsymbol{0},\\ & \boldsymbol{a}_j^T\Delta\bar{\boldsymbol{v}}^{(i)} = 0, \quad j \in \mathcal{G}^{(i)} \end{aligned}$$

if $\|\Delta\bar{\boldsymbol{v}}^{(i)}\| = 0$ **then return** $\bar{\boldsymbol{v}}^{\star} = \bar{\boldsymbol{v}}^{(i)}$

(2) Compute the largest step $\tau^{(i)}$ that satisfies $\boldsymbol{\ell} \leq \bar{\boldsymbol{v}}^{(i)} + \tau^{(i)}\Delta\bar{\boldsymbol{v}}^{(i)} \leq \boldsymbol{u}$, and the index j of a corresponding *blocking constraint* using

$$\tau^{(i)} = \min_{j \notin \mathcal{G}^{(i)}} \left\{ \begin{array}{l} \dfrac{u_j - \bar{v}_j^{(i)}}{\bar{v}_j^{(i)}} \text{ when } \Delta\bar{v}_j^{(i)} > 0 \\ \dfrac{\ell_j - \bar{v}_j^{(i)}}{\bar{v}_j^{(i)}} \text{ when } \Delta\bar{v}_j^{(i)} < 0 \end{array} \right\}$$

if $\tau^{(i)} = 1$ **then return** $\bar{\boldsymbol{v}}^{\star} = \bar{\boldsymbol{v}}^{(i)} + \Delta\bar{\boldsymbol{v}}^{(i)}$

(3) Update our guess $\mathcal{G}^{(i+1)} \leftarrow \{\mathcal{G}^{(i)}, j\}$, perform a step $\bar{\boldsymbol{v}}^{(i+1)} = \bar{\boldsymbol{v}}^{(i)} + \tau^{(i)}\Delta\bar{\boldsymbol{v}}^{(i)}$, and update $i \leftarrow i + 1$.

Algorithm 1 is a modification of a classical primal active-set scheme. $\mathcal{I}$ denotes the set of indexes of the inequality constraints. Only one index in $\mathcal{I}$ is associated with the bounds for the j^{th} variable $\ell_j \leq \bar{v}_j \leq u_j$ (since both bounds for $\bar{v}_j$ can not be active at the same time), *i.e.*, $|\mathcal{I}| = p$. The gradient of both $\bar{v}_j \geq \ell_j$ and $\bar{v}_j \leq u_j$ is denoted by $\boldsymbol{a}^T$ which is a (row) vector of zeros, whose j^{th} element is equal to 1. $\mathcal{G}^{(i)} \subset \mathcal{I}$ denotes the set of indexes of inequality constraints guessed to be active at the i^{th} iteration of the algorithm. The difference from the classical algorithm in [16] pp. 472 is that if $\|\Delta\bar{\boldsymbol{v}}^{(i)}\| = 0$, we do not check the sign of the Lagrange multipliers associated with the inequality constraints in $\mathcal{G}^{(i)}$ (and do not remove constraints from our guess). We have already observed in [5] that in the context of the current application this does not affect the results in a noticeable way. Note that this approximation is not required by the sparse formulation. The algorithm is presented in this way, so that it reflects our numerical implementation (see Section VI).

The most time consuming procedure in **Algorithm 1** is the solution of the linear system of equations associated with the minimization in step **(1)**. When $i = 0$ (hence, $\mathcal{G}^{(0)} = \emptyset$), we have to solve a system that resembles (13), the difference being that here $\kappa = 0$ is assumed (since, the term $\phi(\bar{\boldsymbol{v}})$ does not appear in the objective). As a result $\boldsymbol{S}^{(0)}$ simplifies to

$$\begin{aligned} 2\boldsymbol{S}_{11}^{(0)} &= \tilde{\boldsymbol{Q}}^{-1} + \tilde{\mathbb{B}}_0\boldsymbol{P}^{-1}\tilde{\mathbb{B}}_0^T,\\ 2\boldsymbol{S}_{kk}^{(0)} &= \tilde{\boldsymbol{Q}}^{-1} + \tilde{\mathbb{B}}_{k-1}\boldsymbol{P}^{-1}\tilde{\mathbb{B}}_{k-1}^T + \tilde{\mathbb{A}}_{k-1}\tilde{\boldsymbol{Q}}^{-1}\tilde{\mathbb{A}}_{k-1}^T,\\ 2\boldsymbol{S}_{k,k+1}^{(0)} &= \boldsymbol{S}_{k+1,k}^T = -\tilde{\boldsymbol{Q}}^{-1}\mathbb{A}_k^T. \end{aligned}$$

Algorithm 2: Update of Cholesky factorization in $O(N)$

Input : $m_e = 6N$, $m_a = |\mathcal{G}^{(\text{now})}|$, $\boldsymbol{L}^{(m_a)}$, $\boldsymbol{a}_n^T$ to add.

Output: $\boldsymbol{l}^T$ is the last row of $\boldsymbol{L}^{(m_a+1)}$

(1) Initialize $\boldsymbol{l}^T = \frac{1}{2}\left[\begin{array}{ccc} \boldsymbol{a}_n^T\boldsymbol{H}^{-1}\boldsymbol{E}^T & \boldsymbol{0}_{m_e}^T & \boldsymbol{a}_n^T\boldsymbol{H}^{-1}\boldsymbol{a}_n \end{array}\right]$

(2) The index of the last element in $\boldsymbol{l}^T$ is $q = m_e + m_a + 1$.

for i = *index of the first nonzero element of* $\boldsymbol{l}$ **to** m_e **do**

(3) $l_i = l_i / L_{ii}^{(m_a)}, \quad l_q = l_q - l_i^2$

(4) Since $\boldsymbol{L}^{(0)}$ is sparse, no more than three subsequent elements with (known) indexes $k \leq m_e$ in $\boldsymbol{l}^T$ must be updated: $l_k = l_k - l_i L_{ki}^{(m_a)}$

for $j = m_e + 1$ **to** $q - 1$ **do**

(5) $l_j = l_j - l_i L_{ji}^{(m_a)}$

end

end

for $i = m_e + 1$ **to** $q - 1$ **do**

(6) $l_i = l_i / L_{ii}^{(m_a)}, \quad l_q = l_q - l_i^2$

for $j = i + 1$ **to** $q - 1$ **do**

(7) $l_j = l_j - l_i L_{ji}^{(m_a)}$

end

end

(8) $l_q = \sqrt{l_q}$

As in Section V-C, $\tilde{\boldsymbol{Q}}$ is regularized. If constant sampling time and CoM height are assumed, $\boldsymbol{S}^{(0)}$ is constant.

Once $\Delta\bar{\boldsymbol{v}}^{(0)}$ is computed, **Algorithm 1** proceeds by adding inequality constraints (one at a time) to $\mathcal{G}$ and resolving (14). The difference with the method in Section V-C is that instead of solving the KKT system from scratch every time, since $\boldsymbol{H}$ is constant, and only one constraint is added (at a given i), there exist efficient ways for updating the previously computed factorizations. The updating scheme used in our implementation is summarized in **Algorithm 2**. Note that the complexity of the update is proportional to N.

The size of the system $\boldsymbol{S}^{(i)}\boldsymbol{\nu}^{(i)} = \boldsymbol{s}^{(i)}$ increases by one at each iteration. The right-hand-size vector is given by, $\boldsymbol{s}^{(0)} = -\frac{1}{2}\boldsymbol{E}\boldsymbol{H}^{-1}\nabla\bar{f}(\bar{\boldsymbol{v}}^{(0)})$, $\boldsymbol{s}^{(1)} = (\boldsymbol{s}^{(0)}, -\boldsymbol{a}_n^T\bar{\boldsymbol{v}}^{(1)} - \frac{1}{2}\boldsymbol{a}_n^T\boldsymbol{H}^{-1}\boldsymbol{g})$, etc., where $\boldsymbol{a}_n^T$ is the newly added inequality constraint at $i = 0$. This trivializes the forward substitution to a dot product.

VI. Numerical results

In this section we present numerical results from a C++ implementation of the *active-set* method presented in Section V-D. We perform a comparison between the use of the dense formulation as presented in [1] and the sparse formulation presented here, when the positions of the footsteps are predefined. We use $N = 75$, a constant CoM height and sampling time $T = 20$ ms (*i.e.*, a preview horizon of 1.5 sec). The initial and final double support phases are excluded from the results. The code is compiled using gcc 4.4 with optimization flag `-O3`, and executed on a 2 GHz processor.

In order to perform a fair comparison, we formulate the dense QP by using the change of variable presented in [2] Section IV (that leads to a simply-bounded dense for-

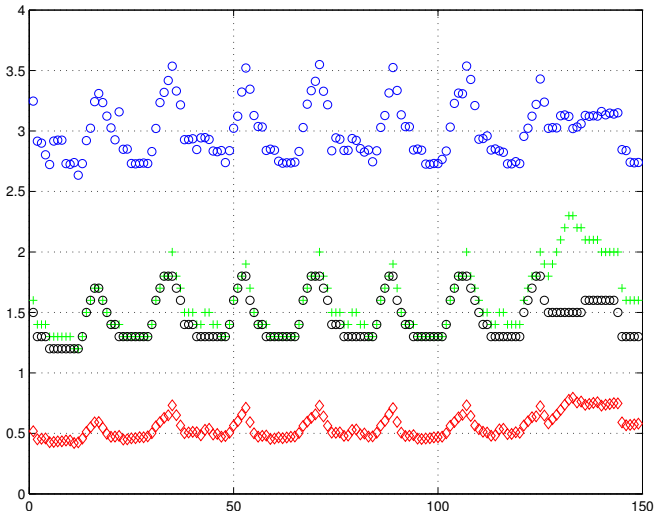

Fig. 1. Red diamonds and blue circles stand for the computation time in ms for our customized solver and the off-the-shelf solver [17], respectively. The black circles represent the number of active constraints (scaled down by 10) at the solution of each QP, while the green "plus" marks give the number of constraints (scaled down by 10) activated by our solver. The increase of activated constraints towards the end does not affect significantly the computation time because the activated constraints appear to be towards the end of the preview horizon (see the first **for** loop in **Algorithm 2**).

mulation). The computation time for performing this change of variable is not considered in the results. The time to solve (using [17]) the resulting dense QP with 150 variables and simple bounds is depicted in Fig. 1 with blue circles. The black circles depict the number of active constraints (scaled down by 10) at the solution of each QP.

The sparse formulation (11) is solved using a customized code. The QP has 600 variables, 450 equality constraints and bounds for 150 of the variables. The fact that the sampling time and CoM height are constant is not considered and the matrix $\boldsymbol{S}^{(0)}$ is formed and factored for each QP even though it is constant in this particular case (this has a negligible effect on the computation time). The computation time is depicted with red diamonds. The number of constraints (scaled down by 10) activated at the (approximate) solution are depicted using green "plus" marks. We did not use "hot-start" ($i.e.$, $\mathcal{G}^{(0)} = \emptyset$), hence even faster computation time could be achieved.

The fast performance of our algorithm is not due to the approximate solution it generates. Currently we are implementing Cholesky *downdate*, which even though is slightly more expensive than the Cholesky update in **Algorithm 2** is not expected to decrease the performance in a "dramatic" way (as usually constraints are dropped from $\mathcal{G}$ much less often than added). The error in the evolution of the CoM (due to the difference in the number of active constraints) is typically of the order of $1e^{-3}$ m. We have observed that most of the time the next state $\bar{\boldsymbol{c}}_1$ generated by our algorithm is identical to (or only slightly different from) the next state that a QP solver produces (which explains the small difference).

From the fact that the complexity of each iteration of our algorithm is $O(N)$, we gain much more for larger problems. At around $N = 10$, the computation times for the dense and sparse formulations become identical (in average 0.03 ms).

VII. Conclusions

We presented a sparse model predictive control scheme for walking motion generation for humanoid robots. We discussed a number of advantages it presents over the "standard" dense formulation, among which: (i) an arbitrary preview window discretization can be used at each iteration; (ii) variation of the height of the center of mass during walking. Both are achieved at a negligible additional computational cost. Even though the underlying quadratic program has a larger dimension (compared to the dense formulation), due to its structure, it can be solved very efficiently. In particular, we derived an optimization problem with a diagonal Hessian matrix subject to only simple bounds.

References

[1] D. Gouaillier, C. Collette, and C. Kilner, "Omni-directional Closed-loop Walk for NAO," *IEEE-RAS International Conference on Humanoid Robots,* pp. 448-454, 2010.

[2] D. Dimitrov, A. Paolillo, and P.-B. Wieber, "Walking motion generation with online foot position adaptation based on ℓ_1- and ℓ_∞-norm penalty formulations," *Proc. of the IEEE Int. Conf. on Robot. & Automat. (ICRA),* pp. 3523-3529, 2011.

[3] A. Herdt, N. Perrin, and P.-B. Wieber, "Walking without thinking about it," *Proc. of the IEEE/RSJ (IROS),* pp. 190-195, 2010.

[4] A. Herdt, H. Diedam, P.-B. Wieber, D. Dimitrov, K. Mombaur, and M. Diehl, "Online Walking Motion Generation with Automatic Foot Step Placement," in *Advanced Robotics Vol. 24 No. 5-6,* April, 2010.

[5] D. Dimitrov, P.-B. Wieber, O. Stasse, H. 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. (ICRA),* pp. 1171-1176, 2009.

[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] 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. (ICRA),* pp. 2685-2690, 2008.

[8] 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.

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

[10] M. Diehl, H. J. Ferreau, N. Haverbeke, "Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation," in *Nonlinear Model Predictive Control, L. Magni, M.D. Raimondo, F. Allgöwer (eds.),* pp. 391-417, 2009.

[11] S. Boyd, L. Vandenberghe, "Convex Optimization," Cambridge, 2004.

[12] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Yokoi, and H. Hirukawa, "A realtime pattern generator for biped walking," in *Proc. of the IEEE Int. Conf. on Robot. & Automat.,* pp.31-37, 2002.

[13] 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.

[14] K. Nishiwaki, and S. Kagami, "Online Design of Torso Height Trajectories for Walking Patterns that takes Future Kinematic Limits into Consideration," in *Proc. of the IEEE Int. Conf. on Robot. & Automat.,* pp. 2029-2034, 2011.

[15] Y. Wang, and S. Boyd, "Fast model predictive control using online optimization," in *IEEE Transactions on Control Systems Technology,* 18(2) pp. 267-278, March 2010.

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

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