Open access • Journal Article • DOI:10.1177/0278364914521306

Hierarchical quadratic programming — Source link

Adrien Escande, Nicolas Mansard, Pierre-Brice Wieber

Institutions: Centre national de la recherche scientifique, University of Toulouse, French Institute for Research in Computer Science and Automation

Published on: 01 Jun 2014 - The International Journal of Robotics Research (SAGE Publications)

Topics: Quadratic programming, Hierarchy (mathematics), Solver, Inverse kinematics and Redundancy (engineering)

Hierarchical quadratic programming: Fast online humanoid-robot motion generation

Adrien Escande, Nicolas Mansard, Pierre-Brice Wieber

Hierarchical Quadratic Programming
Part 1: Fundamental Bases

Adrien Escande
JRL-CNRS/AIST
Tsukuba, Japan

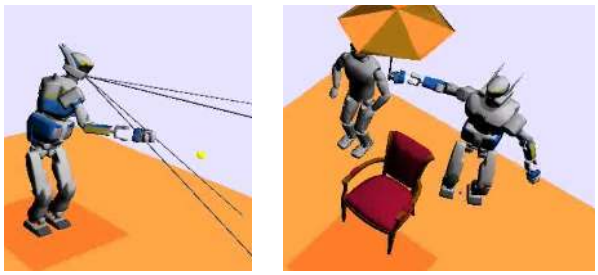
Nicolas Mansard
LAAS-CNRS, Univ. Toulouse
Toulouse, France

Pierre-Brice Wieber
INRIA Grenoble
St Ismier, France

Abstract

Hierarchical least-square optimization is often used in robotics to inverse a direct function when multiple incompatible objectives are involved. Typical examples are inverse kinematics or dynamics. The objectives can be given as equalities to be satisfied (e.g. point-to-point task) or as areas of satisfaction (e.g. the joint range). This two-part paper proposes a complete solution to resolve multiple least-square quadratic problems of both equality and inequality constraints ordered into a strict hierarchy. Our method is able to solve a hierarchy of only equalities ten time faster than the classical method and can consider inequalities at any level while running at the typical control frequency on whole-body size problems. This generic solver is used to resolve the redundancy of humanoid robots while generating complex movements in constrained environment.

In this first part, we establish the mathematical bases underlying the hierarchical problem and propose a dedicated solver. When only equalities are involved, the solver amounts to the classical solution used to handle redundancy in inverse kinematics in a far more efficient way. It is able to handle inequalities at any priority levels into a single resolution scheme, which avoids the high number of iterations encountered with cascades of solvers. A simple example is given to illustrate the interest of our approach. The second part of the paper focuses on the implementation of the solver for inverse kinematics and its use to generate robot motions.



(a) (b)

Fig. 1: *Various situations of inequality and equality constraints. (a) reaching a distant object while keeping balance. The visibility and postural tasks are satisfied only if possible. (b) The target is kept inside a preference area.*

1 Introduction

Least squares are a mean to satisfy *at best* a set of constraints that may not be feasible. When the constraints are linear, the least squares are written as a quadratic program, whose solution is given for example by the pseudo-inverse. Linear least squares have been widely used in robot control in the frame of instantaneous task resolution [De Schutter and Van Brussel, 1988], inverse kinematics [Whitney, 1972] or operational-space inverse dynamics [Khatib, 1987]. These approaches describe the objectives to be pursued by the robot using a function depending on the robot configuration, named the task function [Samson et al., 1991]. The time derivative of this function depends linearly on

the robot velocity, which gives a set of linear constraints, to be satisfied at best in the least-square sense.

When the constraint does not require the use of all the robot degrees of freedom (DOF), the remaining DOF can be used to perform a secondary objective. This redundancy was first emphasized in [Liégeois, 1977]. Least squares can be used again to execute at best a secondary objective using the redundant DOF [Hanafusa et al., 1981], and by recurrence, any number of constraints can be handled [Siciliano and Slotine, 1991]. Here again, the pseudo-inverse is used to compute the least-square optimum. The same technique has been widely used for redundant manipulator [Chiaverini et al., 2008], mobile or underwater [Antonelli and Chiaverini, 1998] manipulator, multi manipulator [Khatib et al., 1996], platoon of mobile robots [Antonelli and Chiaverini, 2006, Antonelli et al., 2010],visual servoing [Mansard and Chaumette, 2007b], medical robots [Li et al., 2012], planning under constraints [Berenson et al., 2011], control of the dynamics [Park and Khatib, 2006], etc. Such a hierarchy is now nearly systematically used in humanoid animation [Baerlocher and Boulic, 2004] and robotics [Sian et al., 2005, Mansard et al., 2007, Khatib et al., 2008].

Very often, the task objectives are defined as equality constraints. On the opposite, some objectives would naturally be written as inequality constraints, such as joints limits [Liégeois, 1977, Chaumette and Marchand, 2001], collision avoidance [Marchand and Hager, 1998, Stasse et al., 2008a], singularities avoidance [Padois et al., 2007, Yoshikawa, 1985] or visibility [Garcia-Aracil et al., 2005, Remazeilles et al., 2006] (see Fig. 1). A first solution to account for these tasks is to embed the corresponding inequality constraint into a potential function [Khatib, 1986] such as a log barrier function, whose gradient is projected into the redundant DOF let free by the first objective [Liégeois, 1977, Gienger et al., 2006]. The potential function simultaneously prevents the robot to enter into a forbidden region and pushes it away from the obstacles when it is coming closer. The potential function in fact transforms the inequality into an equality constraint, that is always applied even far from the obstacle. However, this last property prevents the application of this solution for top-priority constraints.

To ensure that avoidance is realized whatever the situation (and not only when there is enough DOF), several solutions have been proposed, that try to specify inequality objectives as higher-priority tasks. In [Nelson and Khosla, 1995], a trade-off between the avoidance and the motion objectives was performed. In [Chang and Dubey, 1995], the joint limit avoidance was used as a damping factor to stop the motion of a joint close to the limit. Both solutions behave improperly when the motion objective and the avoidance criteria become incompatible. In [Mansard and Chaumette, 2007a], the constraints having priority were relaxed to improve the execution of the avoidance constraint, set at a lower priority level. However, this solution is only valid far from the target point, and the obstacle may finally collide at the convergence of the motion task. In the cases where damping is not sufficient, clamping was proposed [Raunhardt and Boulic, 2007]. It was applied for example to avoid the joint limits of a humanoid [Sentis, 2007]. However, this solution requires several iterations and might be costly. Moreover, it is difficult to relax a DOF that was clamped. In [Mansard et al., 2009], it was proposed to realize an homotopy between the control law with and without avoidance. A proper balance of the homotopy factors ensures that the objectives having priority are respected. However, the cost of this solution in complex cases is prohibitive.

Alternatively, the inequality constraint can be included in the least-square program as it is [Nenchev, 1989, Sung et al., 1996, Decré et al., 2009]. Such a solution is very popular for controling the dynamic of the simulated system [Collette et al., 2007, Salini et al., 2009, Bouyarmane and Kheddar, 2011]. In [Kanoun et al., 2011], a first solution to account for inequality constraints in a hierarchy of tasks was proposed. A simplified version was proposed in [De Lasa et al., 2010], that improves the compu-

tation cost but prevent the inclusion of inequality except at the top priority. In this paper, we propose a complete solution based on [Kanoun et al., 2011] to account for inequality at any level of a hierarchy of least-square quadratic program, while keeping a low computation cost. The contributions of this paper are:

- we propose the first method to solve a hierarchical quadratic program using a single dedicated solver.
- this solver allows to solve a problem of 40 variables (the typical size on a humanoid robot controlled in velocity) in 0.2ms when only equality constraints are set (up to 10 times faster than the state of the art) and in certified 1ms with inequalities, which could allow to do real-time whole-body control at 1kHz.
- we give the analytical computation cost and prove the continuity and the stability of the control laws derived from this solver.
- we give a complete experimental study of the performances of our method against state-of-the-art solvers and demonstrate its capabilities in the robotics context.

The two parts of the paper are structured as follows. The first part establishes the decomposition and the active-search algorithm dedicated to the hierarchical structure. We first recall the mathematical bases of the previous works in Sec. 2. Our solution requires to rewrite the classical hierarchy of equality constraints [Siciliano and Slotine, 1991] in a more efficient manner, which is done in Sec. 3. Then, the algorithm to resolve the hierarchical quadratic least-square program in the presence of inequalities is proposed in Sec. 4. Finally, a small example is given in Sec. 5 as an illustration of the use of the solver in the robotics context. The second part focuses on the implementation of the solver and its use to generate motion by inverse kinematics, while providing a more complete set of experimentations to demonstrate the performances of the method.

2 Hierarchies of tasks

2.1 Context: the task-function approach

The task-function approach [Samson et al., 1991] is an elegant solution to express the objectives to be performed by the robot and deduce from this expression the control to be applied at the joint level. Consider a robot defined by its configuration vector q and whose control input is the joint velocity $\dot{q}$. A task function is any derivable function e of q. The image space of e is named the task space. Its jacobian is denoted $J = \frac{\partial e}{\partial q}$. The evolution in the task space with respect to the robot input is given by:

$$\dot{e} = J\dot{q} \tag{1}$$

The objective to be accomplished by the robot can then be expressed in the task space by giving a reference task velocity $\dot{e}^*$. Computing the robot input boils down to solving the following quadratic least-square program (QP):

$$\text{Find } \dot{q}^* \in \underset{\dot{q}}{\text{Arg min}} \, \|J\dot{q} - \dot{e}^*\| \tag{2}$$

Such a QP formulation can also be encoutered to inverse the system dynamics in the operational space [Khatib, 1987, Collette et al., 2007], compute a walking pattern [Herdt et al., 2010] or optimize a linearized trajectory of the robot whole body [Pham and Nakamura, 2012].

More generically, we consider in this article the case where a robot needs to satisfy a set of linear equality constraints $Ax = b$. In case this set of constraints is not feasible, it has to be satisfied at best in the least-square sense:

$$\text{Find } x^* \in \underset{x}{\text{Arg min}} \, \|Ax - b\| \tag{3}$$

Among the possible x^*, the solution that minimizes the norm of x^* is given by the pseudo-inverse:

$$x^* = A^+ b \tag{4}$$

where A^+ is the (Moore-Penrose) pseudo-inverse of A, which is the unique matrix satisfying the following

four conditions:

$$AA^+A = A \tag{5}$$
$$A^+AA^+ = A^+ \tag{6}$$
$$AA^+ \text{ is symmetric} \tag{7}$$
$$A^+A \text{ is symmetric} \tag{8}$$

The set of all the solutions to (3) is given by [Liégeois, 1977]:

$$x^* = A^+b + P\tilde{x}_2 \tag{9}$$

where P is a projector on the null space of A (*i.e.* such that $AP = 0$ and $PP = P$), and $\tilde{x}_2$ is any arbitrary vector of the parameter space, that can be used as a secondary input to satisfy a second objective.

2.2 Hierarchy of equality constraints [Siciliano and Slotine, 1991]

In this paper, we consider the case where p linear constraints (A_1, b_1) ... (A_p, b_p) have to be satisfied at best simultaneously. If the constraints do not conflict, then the solution is directly obtained by stacking them into a single constraint:

$$\underline{A}_p = \begin{bmatrix} A_1 \\ \vdots \\ A_p \end{bmatrix}, \quad \underline{b}_p = \begin{bmatrix} b_1 \\ \vdots \\ b_p \end{bmatrix} \tag{10}$$

The resulting QP can be solved as before. If the constraints are conflicting, a weighting matrix Q is often used to give more importance to some constraints with respect to others or to artificially create a balance between objectives of various physical dimensions:

$$\min_x \ (\underline{A}_p x - \underline{b}_p)^T Q (\underline{A}_p x - \underline{b}_p) \tag{11}$$

Rather than selecting a-priori values of Q, it was proposed in [Siciliano and Slotine, 1991] to impose a strict hierarchy between the constraints. The first constraint (with highest priority) (A_1, b_1) will be solved at best in the least-square sense using (4). Then the second constraint (A_2, b_2) is solved in the null space of the first constraint without modifying the obtained minimum of the first constraint. Introducing (9) in $A_2x = b_2$, a QP in $\tilde{x}_2$ is obtained:

$$\min_{\tilde{x}_2} \ \|A_2P_1\tilde{x}_2 - (b_2 - A_2A_1^+b_1)\| \tag{12}$$

The generic solution to this QP is:

$$\tilde{x}_2^* = (A_2P_1)^+(b_2 - A_2A_1^+b_1) + \tilde{P}_2\tilde{x}_3 \tag{13}$$

with $\tilde{P}_2$ the projector into the null space of $(A_2P_1)^+$ and $\tilde{x}_3$ any vector of the parameter space that can be used to fulfill a third objective. The complete solution solving (A_1, b_1) at best and (A_2, b_2) if possible is:

$$x_2^* = A_1^+b_1 + (A_2P_1)^+(b_2 - A_2A_1^+b_1) + P_2\tilde{x}_3 \tag{14}$$

where x_2^* denotes the solution for the hierarchy composed of the two first levels, and $P_2 = P_1\tilde{P}_2$ is the projector over $\underline{A}_2$.

This solution can be extended recursively to solve the p levels of the hierarchy [Siciliano and Slotine, 1991]:

$$x_p^* = \sum_{k=1}^{p} (A_kP_{k-1})^+(b_k - A_kx_{k-1}^*) + P_p\tilde{x}_{p+1} \tag{15}$$

with $P_0 = I$, $x_0 = 0$ and $P_k = P_{k-1}\tilde{P}_k$ the projector into the null space of $\underline{A}_k$ [Baerlocher and Boulic, 2004]. $\tilde{x}_{p+1}$ is any vector of the parameter space that denotes the free space remaining after the resolution of the whole hierarchy.

2.3 Projection versus basis multiplication

Given a basis Z_1 of the null space of A_1 (*i.e.* $A_1Z_1 = 0$ and $Z_1^TZ_1 = I$), the projector in the null space of A_1 can be written $P_1 = Z_1Z_1^T$. In that case, it is easy to show that

$$(A_2P_1)^+ = (A_2Z_1Z_1^T)^+ = Z_1(A_2Z_1)^+ \tag{16}$$

The last writing is more efficient to compute than the first one due to the corresponding matrix size. Then,

(15) can be rewritten equivalently under a more efficient form:

$$x_p^* = \sum_{k=1}^{p} Z_{k-1}(A_k Z_{k-1})^+(b_k - A_k x_{k-1}^*) + Z_p z_{p+1} \tag{17}$$

where Z_k is a basis of the null space of $\underline{A}_k$ and z_{p+1} is a vector of the dimension of the null space of $\underline{A}_p$. This observation was exploited in [Escande et al., 2010] (which constitute a preliminary version of this work) to fasten the computation of (15).

2.4 Inequalities inside a cascade of QP [Kanoun et al., 2011]

The problem (3) is an equality-only least-square quadratic program (eQP). Searching a vector x that satisfy a set of linear inequalities is straightforward to write:

$$\text{Find } x^* \in \{x,\ s.t.\ Ax \leq b\} \tag{18}$$

If the polytope defined by $Ax \leq b$ is empty (the set of constraints is infeasible), then x^* can be searched as before as a minimizer in the least-square sense. The form (3) can be extended to inequalities by introducing an additional variable w, named the slack variable, in the parameter vector:

$$\min_{x,w} \|w\| \tag{19}$$

$$\text{subject to} \quad Ax \leq b + w \tag{20}$$

The slack variable can relax the constraint in case of infeasibility [Hofmann et al., 2009]. This QP is named a inequality QP (iQP, by opposition to the eQP). In the remaining of the paper, we keep this reduced shape with only upper bound, since it encompasses lower bounds $Ax \geq b$, double bounds $b_- \leq Ax \leq b_+$ and equalities $Ax = b$ by setting respectively $-Ax \leq -b$, $\begin{bmatrix} -A \\ A \end{bmatrix} x \leq \begin{bmatrix} -b_- \\ b_+ \end{bmatrix}$ and $\begin{bmatrix} -A \\ A \end{bmatrix} x \leq \begin{bmatrix} -b \\ b \end{bmatrix}$. Such an iQP can be solved, for example, using an active-search method (recalled in App. A).

The work in [Siciliano and Slotine, 1991] is limited to a hierarchy of eQP. In [Kanoun et al., 2011], a complete solution to extend the hierarchy to inequality constraints was proposed. The method begins with minimizing the violation $\|w_1\|$ of the first level of constraints in a least-squares sense through (19). This gives a unique optimal value w_1^* since the cost function is strictly convex in w_1. It proceeds then in minimizing the violation of the second level of constraints in a least-squares sense:

$$\min_{x,w_2} \|w_2\| \tag{21}$$

$$\text{subject to} \quad A_1 x \leq b_1 + w_1^* \tag{22}$$

$$A_2 x \leq b_2 + w_2 \tag{23}$$

The first line of constraints (22) is expressed with respect to the fixed value w_1^* obtained from the first QP, which ensures that the new x will not affect the first level of constraints and therefore enforces a strict hierarchy. In that sense, (22) is a strict constraint while (23) is a relaxed constraint. The same process is then carried on through all p levels of priority.

2.5 Reduction of the computation cost [De Lasa et al., 2010]

The solution [Kanoun et al., 2011] makes it possible to solve hierarchies of iQP. However, it is very slow, since each constraint k is solved at the iQP of level k and all the following ones. In particular, the first constraint is solved p times.

In [De Lasa et al., 2010], a solution is proposed to lower the computation cost by reducing the generic nature of the problem studied in [Kanoun et al., 2011]: inequalities are considered only at the first level, and this level is supposed feasible. This hypothesis reduces the expressiveness of the method, forbidding the use of weak constraints such as visibility or preference area. However, this expressivity reduction enables to obtain very impressive result for walking, jumping or, as shown in [Mordatch et al., 2012], for planning contacts and manipulation.

The first iQP of the cascade does not need an explicit computation, since $w_1^* = 0$ by hypothesis. Then

each level $k > 2$ is solved in the null space of the levels 2 to $k-1$:

$$\min_{z_k, w_k} \|w_k\| \tag{24}$$

$$\text{subject to} \quad A_1(x^*_{k-1} + Z_{k-1} z_k) \leq b_1 \tag{25}$$

$$A_k(x^*_{k-1} + Z_{k-1} z_k) = b_k + w_k \tag{26}$$

where x^*_{k-1} is the optimal solution for the $k-1$ first levels, and Z_{k-1} is the null space of the levels 2 to $k-1$. The solution in the canonical basis after the k^{th} QP is set to $x^*_k = x^*_{k-1} + Z_{k-1} z^*_k$. The Z_k basis is computed from Z_{k-1} and a singular value decomposition (SVD) of $(A_k Z_{k-1})$.

If the first level is empty (or equivalently, if the bounds are wide enough for never being activated), this method is equivalent to (17). The global working scheme of the method [De Lasa et al., 2010] is the same as [Kanoun et al., 2011] since both rely on a cascade of QP computed successively for each level of the hierarchy. The method of [De Lasa et al., 2010] is faster since each QP is smaller than the previous one (the dimension of z_k decreases with k), while each QP of [Kanoun et al., 2011] was bigger than the previous ones (the number of constraints increases). The method of [De Lasa et al., 2010] requires an additional SVD, but this could be avoided since the SVD or an equivalent decomposition is already computed when solving the corresponding QP.

However, both methods [Kanoun et al., 2011] and [De Lasa et al., 2010] have the same intrinsic problem due to the nature of the underlying active search algorithm[1]. Basically, it searches for the set of active constraints, that holds as equality at the optimum. At each new QP of the cascade, the optimal active set may be completely different. The active search may then activate and deactivate a constraint several times when moving along the cascade, and the succession of all these iterative processes appears to be very inefficient in the end.

Typical examples of this situation are given in Section 5 and in the second part of the paper. Consider a humanoid robot that should keep its center of mass inside the support polygon, put its right hand in the front and its left hand far in the back: when solving the right-hand constraint, the center of mass will saturate in the front, which activates the corresponding constraint. The front constraint is then deactivated when the left-hand constraint brings the center of mass on the back, while the back of the support polygon becomes active. The back constraint may even be deactivated if a last level is added that regulates the robot posture.

[1]The principle of the active search algorithm is recalled in App. A.

2.6 Strict hierarchies and lexicographic order

By minimizing successively $\|w_1\|$, $\|w_2\|$ until $\|w_p\|$, the above approaches end up with a sequence of optimal objectives $\{\|w^*_1\|, \|w^*_2\|, \ldots, \|w^*_p\|\}$ which is itself minimal with respect to a lexicographic order: it is not possible to decrease an objective $\|w_k\|$ without increasing an objective $\|w_j\|$ with higher priority ($j < k$). Considering a hierarchy between these objectives or a lexicographic order appear to be synonyms. The above approaches can therefore be summarized as a lexicographic multi-objective least-squares quadratic problem, looking for

$$\underset{x, w_1 \ldots w_p}{\text{lex min}} \; \{\|w_1\|, \|w_2\|, \ldots, \|w_p\|\}. \tag{27}$$

$$\text{subject to} \quad \forall k = 1:p, \; A_k x \leq b_k + w_k$$

The earliest and most obvious approach to solve such lexicographic multi-objective optimization problems is to solve the single-objective optimization problems successively [Behringer, 1977], exactly in the same way described above [Kanoun et al., 2011, De Lasa et al., 2010]. This is a very effective approach, but in the case of inequality constraints, each optimization problem requires an iterative process to be solved, and the succession of all these iterative processes appears to be very inefficient in the end.

Following the example of the lexicographic simplex method proposed in [Isermann, 1982], our approach is to adapt the classical iterative processes for optimization problems with inequality constraints directly to the situation of lexicographic multi-objective optimization, resulting in a single iterative process to solve the whole multi-objective problem at

once, what appears to be much more efficient. This is what we are going to develop in the following sections for the problem (27).

3 Equality hierarchical quadratic program

3.1 Optimality conditions

At first, we consider an equality-only hierarchical quadratic least-square program (eHQP). It is written as a set of p eQP: at level k, the QP to be solved is written:

$$\min_{x_k, w_k} \|w_k\| \tag{28}$$

$$\text{subject to} \quad A_k x_k = b_k + w_k \tag{29}$$

$$\underline{A}_{k-1} x_k = \underline{b}_{k-1} + \underline{w}^*_{k-1} \tag{30}$$

where $\underline{A}_{k-1}$, $\underline{b}_{k-1}$ and $\underline{w}^*_{k-1}$ are the matrix and vectors composed of the stacked quantities of levels 1 to $k-1$ (by convention, they are empty matrix and vectors for $k-1=0$). $\underline{w}^*_{k-1}$ is the fixed value obtained from the previous QP. The Lagrangian of this problem is:

$$\mathcal{L}_k = \frac{1}{2} w_k^T w_k + \underline{\lambda}_k^T (\underline{A}_{k-1} x_k - \underline{b}_{k-1} - \underline{w}^*_{k-1}) + \lambda_k^T (A_k x_k - b_k - w_k) \tag{31}$$

where $\underline{\lambda}_k$ and λ_k are the Lagrange multipliers corresponding respectively to (30) and (29). Differentiating the Lagrangian over the primal variables x_k, w_k and dual variables $\underline{\lambda}_k, \lambda_k$ gives the optimality conditions:

$$w_k = A_k x_k - b_k \tag{32}$$

$$\underline{A}_{k-1} x_k = \underline{b}_{k-1} + \underline{w}^*_{k-1} \tag{33}$$

$$\lambda_k = w_k \tag{34}$$

$$\underline{A}_{k-1}^T \underline{\lambda}_k = -A_k^T w_k \tag{35}$$

The two first lines give the condition to compute the primal optimum. From (34), we see that w is indeed at the same time a primal and a dual variable. The last equation gives the condition to compute the dual optimum.

3.2 Complete orthogonal decomposition

In the first level, (30) and (33) are empty. The primal optimum is computed by minimizing w_1 in (32), that is to say by a classical pseudo-inverse of A_1. The pseudo-inverse can be computed by performing a complete rank revealing decomposition. Typically a SVD can be chosen. Alternatively, a complete orthogonal decomposition (COD) can be used [Golub and Van Loan, 1996]:

$$A_1 = \begin{bmatrix} V_1 & U_1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ L_1 & 0 \end{bmatrix} \begin{bmatrix} Y_1 & Z_1 \end{bmatrix}^T = U_1 L_1 Y_1^T \tag{36}$$

where $W_1 = \begin{bmatrix} V_1 & U_1 \end{bmatrix}$ and $\begin{bmatrix} Y_1 & Z_1 \end{bmatrix}$ are two orthonormal matrices, U_1 being a basis of the range space of A_1, Z_1 of its kernel and L_1 is a lower triangular matrix whose diagonal is strictly nonzero. If the first level (A_1, b_1) is feasible, then A_1 is full row rank and U_1 is the identity (V_1 is empty). In that case, (36) is the QR decomposition of A_1^T (or LQ decomposition of A_1).

The COD is cheaper to compute than the SVD. The algorithms to compute it involve a rather simple serie of basic transformations (Givens or Householder rotations) and are known to be nearly as robust as the algorithms computing the SVD (and much easier to implement). It is one of the classical ways to solve rank deficient quadratic least-squares problems [Björck, 1996].

The pseudo-inverse of A_1 now only implies the easily computable inversion of L_1:

$$A_1^+ = \begin{bmatrix} Y_1 & Z_1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ L_1^{-1} & 0 \end{bmatrix} \begin{bmatrix} V_1 & U_1 \end{bmatrix}^T = Y_1 L_1^{-1} U_1^T \tag{37}$$

The optimal solution x_1^* is obtained by

$$x_1^* = A_1^+ b_1 = Y_1 L_1^{-1} U_1^T b_1 \tag{38}$$

Rather than computing the explicit pseudo-inverse, the optimum should be computed by realizing a forward substitution of L_1 on $U_1^T b_1$.

The corresponding slack variable is:

$$w_1^* = A_1 x_1^* - b_1 = U_1 U_1^T b_1 - b_1 = -V_1 V_1^T b_1 \tag{39}$$

3.3 Hierarchical complete orthogonal decomposition

Consider now the second level of the hierarchy (28) ($k = 2$). As in Sec. 2.3, condition (33) can be rewritten using (38) and (39) as:

$$x_2 = x_1^* + Z_1 z_2 \tag{40}$$

where z_2 is any parameter of the null space of A_1. Condition (32) is then written:

$$w_2 = (A_2 Z_1) z_2 - (b_2 - A_2 x_1^*) \tag{41}$$

$$= \begin{bmatrix} A_2 Y_1 & A_2 Z_1 \end{bmatrix} \begin{bmatrix} Y_1^T x_1^* \\ z_2 \end{bmatrix} - b_2 \tag{42}$$

because $Y_1 Y_1^T x_1^* = x_1^*$. The matrix A_2 is in fact separated in two parts along the Y_1, Z_1 basis: the first part $A_2 Y_1$ corresponds to the coupling between the two first levels. The corresponding part of the parameter space has already been used for the level 1 and can not be used here. The second part $A_2 Z_1$ corresponds to the free space that can be used to solve the second level.

The optima x_2^* and w_2^* are obtained by performing the pseudo-inverse of $A_2 Z_1$ using its COD:

$$(A_2 Z_1) = \begin{bmatrix} V_2 & U_2 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ L_2 & 0 \end{bmatrix} \begin{bmatrix} \tilde{Y}_2 & \tilde{Z}_2 \end{bmatrix}^T \tag{43}$$

The basis $W_2 = \begin{bmatrix} V_2 & U_2 \end{bmatrix}$ provides a decomposition of the image space of A_2 along its range space and the orthogonal to it. The basis $\begin{bmatrix} Y_2 & Z_2 \end{bmatrix} = Z_1 \begin{bmatrix} \tilde{Y}_2 & \tilde{Z}_2 \end{bmatrix}$ is in fact another basis of the null space of A_1 that also provides a separation of the kernel of A_2. In particular, Z_2 is a basis of the null space of both A_1 and A_2 that can be used to perform the third level.

The optimum x_2^* is finally:

$$x_2^* = x_1^* + Z_1 (A_2 Z_1)^+ (b_2 - A_2 x_1^*) + Z_2 z_3 \tag{44}$$

$$= x_1^* + Y_2 L_2^{-1} U_2^T (b_2 - A_2 x_1^*) + Z_2 z_3 \tag{45}$$

$$= \begin{bmatrix} Y_1 & Y_2 & Z_2 \end{bmatrix} \begin{bmatrix} Y_1^T x_1^* \\ Y_2^T \tilde{x}_2^* \\ z_3 \end{bmatrix} \tag{46}$$

where $\tilde{x}_2^* = Y_2 L_2^{-1} U_2^T (b_2 - A_2 x_1^*)$ is the contribution of the second level to the optimum and z_3 is any parameter of the null space Z_2 used to perform the following levels. The optimum w_2^* is directly obtained using (42). This can be written using the two basis V_2, U_2 and Y_1, Y_2, Z_2:

$$w_2^* = \begin{bmatrix} V_2 & U_2 \end{bmatrix} \begin{bmatrix} N_2 & 0 & 0 \\ M_2 & L_2 & 0 \end{bmatrix} \begin{bmatrix} Y_1^T x_1^* \\ Y_2^T x_2^* \\ z_3 \end{bmatrix} - b_2 \tag{47}$$

with $M_2 = U_2^T A_2 Y_1$ and $N_2 = V_2^T A_2 Y_1$ the coupled part of A_2 corresponding respectively to its feasible space U_2 and its orthogonal, and using $Y_2^T \tilde{x}_2^* = Y_2^T x_2^*$. When the projected matrix $A_2 Z_1$ is full row rank, V_2 and N_2 are empty. If the ranks of A_2 and $A_2 Z_1$ are the same, then N_2 is zero. This case is sometimes called a kinematic singularity [Chiaverini, 1997]. Alternatively, the rank of A_2 can be greater than $A_2 Z_1$. In that case, N_2 is nonzero, and the singularity is caused by a conflict with the level having priority. This case is sometimes called an algorithmic singularity [Chiaverini, 1997].

In (47) a decomposition of the matrix A_2 appears, that can be written generically for any $k \geq 2$:

$$A_k = \begin{bmatrix} V_k & U_k \end{bmatrix} \begin{bmatrix} N_k & 0 & 0 \\ M_k & L_k & 0 \end{bmatrix} \begin{bmatrix} \underline{Y}_{k-1} & Y_k & Z_k \end{bmatrix}^T \tag{48}$$

$$= W_k H_k \underline{Y}_k^T \tag{49}$$

with $N_k = V_k^T A_k \underline{Y}_{k-1}$, $M_k = U_k^T A_k \underline{Y}_{k-1}$, $\underline{Y}_{k-1} = \begin{bmatrix} Y_1 & \dots & Y_{k-1} \end{bmatrix}$ and $H_k = \begin{bmatrix} N_k & 0 \\ M_k & L_k \end{bmatrix}$. Similarly to the second level, N_k is empty if $A_k Z_{k-1}$ is full row rank; it is zero if the rank deficiency is inherent to the level (kinematic singularity) and nonzero if the rank deficiency is due to a conflict with a level having priority (algorithmic singularity). In that third cases, the stronger the norm of a column of N_k, the stronger the coupling of the unfeasible part with the corresponding higher level. The strongest columns of N_k can then indicate the conflicting levels that prevent the realization of level k.

Stacking all the decompositions (48) for the k first levels, a single decomposition of $\underline{A}_k$ is recursively ob-

tained by:

$$\begin{bmatrix}\underline{A}_{k-1}\\ A_k\end{bmatrix} = \begin{bmatrix}\underline{W}_{k-1} & 0\\ 0 & W_k\end{bmatrix}\begin{bmatrix}\underline{H}_{k-1} & 0 & 0\\ N_k & 0 & 0\\ M_k & L_k & 0\end{bmatrix}[\underline{Y}_{k-1}\, Y_k\, Z_k]^T$$
$$= \underline{W}_k \underline{H}_k \underline{Y}_k^T \quad (50)$$

The complete form for the p levels is finally:

$$\begin{bmatrix}A_1\\ \vdots\\ A_p\end{bmatrix} = \begin{bmatrix}W_1 & & \\ & \ddots & \\ & & W_p\end{bmatrix}\begin{bmatrix}0 & 0 & 0 & 0 & 0\\ \llcorner L_1 \lrcorner & 0 & 0 & 0 & 0\\ \boxed{\begin{matrix}N_2\\ M_2\end{matrix}} & \llcorner L_2 \lrcorner & 0 & 0 & 0\\ \vdots & \vdots & \vdots & \vdots & \\ \boxed{\begin{matrix}N_p\\ M_p\end{matrix}} & & & 0 & 0\\ & & & \llcorner L_p \lrcorner & 0\end{bmatrix} Y^T$$

with $Y = \begin{bmatrix}\underline{Y}_p & Z_p\end{bmatrix}$.

If all the levels are feasible, all the matrices A_k and $A_k Z_{k-1}$ are full row rank and all the N_k matrices are empty. In this case, the decomposition is a COD of $\underline{A}_p$. If there is no conflict between the levels, all the N_k are zero. In this case, it is only a matter of row permutations to turn the above decomposition into a perfect COD of the matrix $\underline{A}_p$, involving an invertible lower triangular matrix. This decomposition, which has been designed to enforce a strict hierarchy between different priority levels, looks close to a classical COD and is indeed a COD in particular cases. For this reason, we propose to call this decomposition a Hierarchical Complete Orthogonal Decomposition (HCOD) of the matrix $\underline{A}_p$.

3.4 Primal optimum and hierarchical inverse

3.4.1 Computing x_p^*

We have seen in (45) that the primal optimum of the second level x_2^* is directly computed from H_2 and x_1^*. Using the same reasoning, the optimum of level k, given by (17), is computed using H_k:

$$x_k^* = x_{k-1}^* + Y_k L_k^{-1} U_k^T (b_k - A_k x_{k-1}^*) + Z_k z_{k+1} \quad (51)$$

The least norm solution for x_k^* is obtained for every $z_{i+1} = 0$, what we suppose from now on. As observed in this equation, the optimum of each level k is computed using the level k of the HCOD and the optimum of level $k-1$. By recurrence, x_p^* can be computed directly using the HCOD, by reformulating (51) in the following way:

$$x_p^* = \underline{A}_p^\ddagger \underline{b}_p \quad (52)$$

where $\underline{A}_p^\ddagger$ is defined by a matrix recursion:

$$\underline{A}_k^\ddagger = \begin{bmatrix}(I - Y_k L_k^{-1} U_k^T A_k)\underline{A}_{k-1}^\ddagger & Y_k L_k^{-1} U_k^T\end{bmatrix} \quad (53)$$

Or more simply, with the HCOD:

$$\underline{A}_p^\ddagger = \underline{Y}_p\, \underline{H}_p^\ddagger\, \underline{W}_p^T \quad (54)$$

with

$$\underline{H}_k^\ddagger = \begin{bmatrix}\underline{H}_{k-1}^\ddagger & 0 & 0\\ -L_k^{-1} M_k \underline{H}_{k-1}^\ddagger & 0 & L_k^{-1}\end{bmatrix} \quad (55)$$

From this last form, we can see that the optimum is structured by layer, following the hierarchy of problems. This structure is more evident when the optimum is computed in the Y basis. The primal optimum in the Y basis is denoted $\underline{y}_k^* = \underline{Y}_k^T x_k^*$. The contribution $x_k^* - x_{k-1}^*$ of the level k to the primal optimum is denoted $y_k^* = Y_k^T(x_k^* - x_{k-1}^*)$ ($= Y_k^T x_k^*$ since $Y_k^T x_{k-1}^* = 0$). Then (51) can be rewritten as:

$$\underline{y}_k^* = \begin{bmatrix}\underline{y}_{k-1}^*\\ y_k^*\end{bmatrix} \quad (56)$$

where $y_k^* = L_k^{-1}(U_k^T b_k - M_k \underline{y}_{k-1}^*)$. Each component k of the optimum vector $\underline{y}_p^* = (y_1^*, \ldots, y_p^*)$ corresponds to the contribution of the level k of the hierarchy. The study of $\underline{y}^*$ is thus very informative to understand the obtained x_p^*: for example the hierarchy levels that induce large contributions in x_p^* directly appear in $\underline{y}_p^*$.

3.4.2 Moore-Penrose conditions of $\underline{A}_p^\ddagger$

The obtained matrix $\underline{A}_p^\ddagger$ respects three of the four conditions of Moore-Penrose (5)-(8). First, $\underline{H}_p^\ddagger \underline{H}_p$

is easily shown to be equal to the identity matrix (by recurrence, starting from $\underline{H}_1^\ddagger \underline{H}_1 = L_1^{-1} L_1 = I$). Then:

$$\underline{A}_p^\ddagger \underline{A}_p = \underline{Y}_p \underline{Y}_p^T = \sum_{k=1}^{p} Y_k Y_k^T \tag{57}$$

This matrix product is obviously symmetric: $\underline{A}_p^\ddagger$ respects (8). From this result, the condition (5) is easily demonstrated:

$$\underline{A}_p \underline{A}_p^\ddagger \underline{A}_p = \underline{W}_p \underline{H}_p \underline{Y}_p^T \underline{Y}_p \underline{Y}_p^T = \underline{A}_p \tag{58}$$

With the same argument, (6) is also respected. The last condition (7) is not respected in general:

$$\underline{H}_k \underline{H}_k^\ddagger = \begin{bmatrix} \underline{H}_{k-1} \underline{H}_{k-1}^\ddagger & 0 & 0 \\ N_k \underline{H}_{k-1}^\ddagger & 0 & 0 \\ 0 & 0 & I \end{bmatrix} \tag{59}$$

Since $\underline{H}_{k-1}^\ddagger$ is full row rank, the term $N_k \underline{H}_{k-1}^\ddagger$ is zero *iff* N_k is zero, that is to say if level k does not conflict with the above hierarchy. If all the $N_1 ... N_k$ are zero, the fourth property (7) is then also respected: $\underline{A}_p^\ddagger$ is strictly the pseudo-inverse of $\underline{A}_p$ and the optimum (51) enforcing a strict hierarchy between the different priority levels appears to be equal to a classical least-squares solution to $\underline{A}_k x = \underline{b}_k$ regardless of any hierarchy.

In general, the matrix $\underline{A}_p^\ddagger$ respects only three of the four properties of Moore-Penrose. This matrix is a reflexive generalized inverse of $\underline{A}_p$. It has been designed to enforce a strict hierarchy between different priority levels and looks close to a pseudo-inverse. For this reason, we propose to call this matrix the *hierarchical inverse* of the matrix $\underline{A}_k$.

3.4.3 Damped inverse

The hierarchical inverse computes the same solution than the classical method [Siciliano and Slotine, 1991] and is thus subject to the same singularities [Chiaverini, 1997]. These singularities are studied in the second part (see [Escande et al., 2012], Sec. 4.3). A singularity is reached when one of the diagonal element of a L_k becomes zero. The corresponding row is then moved from M_k to N_k. In the neighborhood of the singularity, the diagonal element of L_k is small and can produce a large response in x_k^*. This is generally a desirable feature in numerical optimization (since it corresponds to an accurate solution of a ill-conditionned problem) but is often undesirable in robotics. In that case, a damped inverse of L_k can be used [Wampler, 1986, Deo and Walker, 1992, Sugihara, 2011]:

$$L_k^{\dagger\eta} = (L_k L_k^T + \eta^2 I)^{-1} L_k^T = \begin{bmatrix} L_k \\ \eta I \end{bmatrix}^+ \begin{bmatrix} I & 0 \end{bmatrix} \tag{60}$$

3.4.4 Computing the $\underline{w}_p^*$

For each level k, the slack variable is directly obtained from x_k^* using (32). By replacing A_k by $W_k H_k \underline{Y}_k^T$ and x_k^* by (56), we obtain:

$$w_k^* = V_k N_k \underline{y}_{k-1}^* - V_k V_k^T b_k \tag{61}$$

$$= V_k V_k^T (A_k x_{k-1}^* - b_k) \tag{62}$$

The slack of level k does not depend on the optimum x_k^* of the same level. It is equal to the projection of b_k into the orthogonal to the range space, from which the contribution of the previous levels is subtracted. In particular, if there is no conflict between the levels, the slack is directly obtained by projecting b_k. Of course, if level k is not in singularity (intrinsically or due to the previous levels), V_k is empty and the slack is zero. A singular point is obtained when V_k is not empty but $A_k x_{k-1}^* = V_k V_k^T b_k$: in that case, the level is conflicting with the previous ones, but it is exactly realized by the contribution of the above hierarchy.

Finally, the algorithm to compute x_p^* and $\underline{w}_p^*$ is summarized in Alg. 1.

3.5 Transposed hierarchical inverse and dual optimum

At level k, the dual optimum is given by (35), recalled here:

$$\underline{A}_{k-1}^T \underline{\lambda}_k = -A_k^T w_k$$

A solution (the least-square one) to this second optimality condition can be obtained with the hierarchical inverse:

$$\underline{\lambda}_k = -\underline{A}_{k-1}^{\ddagger T} A_k^T w_k^* \tag{63}$$

Algorithm 1 Primal eHQP

1: **function eHQP_primal**($\underline{A}_p$, $\underline{b}_p$)
2: **Input:** HCOD of $\underline{A}_p$, $\underline{b}_p$
3: **Output:** $x_p^*, \underline{w}_p^*$ minimizing (28)
4: $\underline{y}_0^* := []$
5: **for** k in 1:p **do**
6: $\quad e = U_k^T b_k - M_k \underline{y}_{k-1}^*$
7: $\quad w_k^* = V_k(N_k \underline{y}_{k-1}^* - V_k^T b_k)$
8: $\quad e := L_k^{-1} e$
9: $\quad \underline{y}_k^* := [\ \underline{y}_{k-1}^*\ ;\ e\ ]$
10: **end for**
11: $x_p^* = \underline{Y}_p \underline{y}_p^*$, $\underline{w}_p^* = (w_1^*, ..., w_p^*)$
12: **return** $x_p^*, \underline{w}_p^*$

Indeed, we can verify that this solution satisfies the condition (35). Setting $\underline{\lambda}_k$ and w_k^* in (35) and using (57), we obtain:

$$\underline{A}_{k-1}^T \underline{\lambda}_k = -\underline{A}_{k-1}^T \underline{A}_{k-1}^{\ddagger T} A_k^T w_k^* = -\underline{Y}_{k-1} \underline{Y}_{k-1}^T A_k^T w_k^* \tag{64}$$

This last form is equal to $A_k^T w_k^*$ since, from (48) and (61), we have:

$$A_k^T w_k^* = \underline{Y}_{k-1} N_k^T (N_k \underline{y}_{k-1}^* - V_k^T b_k) \tag{65}$$

For each level k, there is a multiplier $\underline{\lambda}_k$ that corresponds to all the level of higher priority. There is no real sense in stacking the multipliers of each levels. They can be summarized under a pseudo matrix structure:

$$\Lambda_p = \begin{bmatrix} w_1^* & {}^1\underline{\lambda}_2 & {}^1\underline{\lambda}_3 & \dots & {}^1\underline{\lambda}_{p-1} & {}^1\underline{\lambda}_p \\ & w_2^* & {}^2\underline{\lambda}_3 & \dots & {}^2\underline{\lambda}_{p-1} & {}^2\underline{\lambda}_p \\ & & w_3^* & \dots & {}^3\underline{\lambda}_{p-1} & {}^3\underline{\lambda}_p \\ & & & & \vdots & \vdots \\ & & & & w_{p-1}^* & {}^{p-1}\underline{\lambda}_p \\ & & & & & w_p^* \end{bmatrix} \tag{66}$$

where ${}^j\underline{\lambda}_k$ ($j < k$) denotes the components of the multipliers $\underline{\lambda}_k$ of the level k for the constraints of level j and the empty spaces for $j > k$ express the absence of multipliers on above levels.

There is no direct formulation to compute the whole Λ_p. Alternatively, the multipliers of each level have to be computed iteratively. The solution (63) gives the matrix formulation of the Lagrange multipliers of level k. To compute the multipliers, it is more efficient to avoid the explicit computation of the hierarchical transpose inverse. Using (53), (63) can be rewritten:

$$\underline{\lambda}_k = \begin{bmatrix} {}^1\underline{\lambda}_k \\ \vdots \\ {}^{k-1}\underline{\lambda}_k \end{bmatrix} = \begin{bmatrix} -\underline{A}_{k-2}^{\ddagger T}(A_k^T w_k^* + A_{k-1}^T\ {}^{k-1}\underline{\lambda}_k) \\ -U_{k-1} L_{k-1}^{-T} Y_{k-1}^T A_k^T w_k^* \end{bmatrix} \tag{67}$$

By recurrence, the components ${}^j\underline{\lambda}_k$ of the multipliers of level k can be computed starting from $j = k-1$ down to 1.

$${}^j\underline{\lambda}_k = -U_j L_j^{-T} Y_j^T \Big(\sum_{i=j+1}^{k-1} A_i^T\ {}^i\underline{\lambda}_k + A_k^T w_k^* \Big) \tag{68}$$

Using the HCOD structuration of the $A_i^T\ {}^i\underline{\lambda}_k$, the sum on the right part can be computed for a reduced cost. The algorithm is given in Alg. 2. The cumulative variable ρ is used to propagate the recursion across the k levels. At the end of any iteration j, the following property is respected:

$$\underline{\rho}^{(j)} = \underline{Y}_j^T \Big(\sum_{i=j+1}^{k-1} A_i^T\ {}^i\underline{\lambda}_k + A_k^T w_k^* \Big) \tag{69}$$

In line #7, $\underline{\rho}^{(j+1)}$ is separated in two parts following the separation of $\underline{A}_{j+1} = \begin{bmatrix} \underline{A}_j \\ A_{j+1} \end{bmatrix}$. The first part of the vector is used to satisfy (69) while the second part gives ${}^j\underline{\lambda}_k$ using (68).

While the primal algorithm 1 computes the primal optima x and w for all the levels at the same time, the dual algorithm 2 can only achieve the computation of w and $\underline{\lambda}$ for one level at a time. Both algorithms can compute w naturally. If both are used, then a choice has to be made on where to really perform the computation of w^*.

3.6 Conclusion

In this section, we proposed a complete solution to compute the primal and dual solutions of a hierarchical quadratic least-square program with only

Algorithm 2 Dual eHQP of level k

function eHQP_dual($\underline{A}_p$, $\underline{b}_p, x^*, k$)
Input: HCOD of $\underline{A}_p$, $\underline{b}_p$, Primal optimum x^*, level k
Output: w_k^* and $\underline{\lambda}_k$ satisfying (32) and (35)
$e = N_k \underline{y}_k^* - V_k b_k$
$w_k^* = V_k e$
$\underline{\rho} = -N_k^T e$
for j=k-1 **downto** 1 **do**
  $\begin{bmatrix} \underline{\rho} \\ \rho \end{bmatrix} := \underline{\rho}$
  $r := L_j^{-T} \rho$
  ${}^j\underline{\lambda}_k := U_j r$
  $\underline{\rho} := \underline{\rho} - M_j^T r$
end for
return $w_k^*, \underline{\lambda}_k$

equality constraints. The solved problem is in fact nothing more than the problem already solved in [Siciliano and Slotine, 1991]. The proposed solution enables much faster computation, thanks to the use of a COD instead of a classical SVD but also mainly because of the form $Z_{k-1}(A_k Z_{k-1})^+$ that is much more efficient than the classical $(A_k P_{k-1})^+$. The underlying structures that we have emphasized are also interesting by themselves, for the information they reveal on the problem structure: the HCOD can be used to study the conflicts between the level of the hierarchy, and the optimum $\underline{y}^*$ reveals the hierarchy structure of the optimal solution. Finally, we have proposed a solution to compute the Lagrange multipliers of the hierarchical problem. These multipliers also provide a lot of information about the problem organization: for example, if one level is not properly satisfied because of a conflict, the multipliers can help to select which level to remove to enhance its resolution. Lagrange multipliers are also mandatory for the active-search algorithm proposed in the next section.

4 Inequality hierarchical quadratic program

We now consider the hierarchical problem (27) subject to inequality constraints.

4.1 Available options and principle

The two main classes of algorithms to find the optimum of a quadratic cost function over inequality constraints are the interior-point methods (IPM) and the active-search algorithms. IPM convert the quadratic problem with inequality constraints into a non-linear parametric problem without inequality, and iterates a deterministic number of times toward the optimum [Nemirovski, 1994].

On the opposite, active-search algorithms iterate to find the set of constraints that hold as equalities at the optimum, named the active set. Convergence in finite time is certified by making sure that the objective function is strictly decreasing at each iteration. This approach led to the simplex method in the case of linear programming [Dantzig and Thapa, 2003] or active-set methods in the case of quadratic programming. The number of iterations is non-deterministic, but can be small if a good initial guess of the optimal active set is known. Such a knowledge is more difficult to use with IPM. For example, in sequential quadratic programming, active searches are often preferred to IPM due to this property. In robotics the optimal active set is slowly evolving over time, what makes active-search algorithm more suitable than IPM.

Active-search methods are themselves divided mainly in two classes. The primal active search starts with a feasible solution (which is sometimes costly to compute), and maintains the feasibility over all the iterations. The dual active search does not need a feasible start, but the feasibility is only met at the optimum. Primal algorithm is interesting in the perspective of real-time implementation, since it can be interrupted at any time and returns a consistent solution. This property will be explored in the second part of the paper (see [Escande et al., 2012], Sec. 4.4).

In this section, we propose to extend the classical

primal active-search algorithm (recalled in App. A) to compute the optimal active set and the corresponding optimum of a HQP with inequalities (or iHQP by opposition to the eHQP). To avoid the unnecessary iterations encountered in [Kanoun et al., 2011, De Lasa et al., 2010], the active sets of all the hierarchical levels are computed at the same time. We first propose a rewriting of the classical active search for an iQP. From this reformulation, the original iHQP algorithm will be established.

4.2 Dedicated active search for the two first levels

We consider the reduced case with only two levels and the first level strictly feasible (case (27) with $p = 2$ and $w_1^* = 0$):

$$\min_{x,w_2} \|w_2\| \tag{70}$$

$$\text{subject to} \quad A_2 x \leq b_2 + w_2 \tag{71}$$

$$A_1 x \leq b_1 \tag{72}$$

This iQP can be solved using a classical active-search. However, this would not take into account the specific form of the variable w_2. Indeed, the slack variable w_2 is one of the variable to be optimized. Theoretically, it has the same meaning as the variable x. In practice, its specific role in the constraint equations can be taken into account to reduce the amount of computation. In particular, an algorithm computing the solution of (70) can be used directly to compute the optimum of a problem without slacks with no efficiency lost. We propose below a modified version of the classical active-search algorithm (recalled in App. A). This algorithm is an intermediary result that will be used to build and justify the hierarchical algorithm proposed in the following section.

4.2.1 Algorithm principle

For a given x, a constraint A_r, b_r is *satisfied* if $A_r x \leq b_r$, *violated* if $A_r x > b_r$ and *saturated* if $A_r x = b_r$. The algorithm maintains a set of constraints that are saturated (for level 1) or should be minimized (for level 2) and thus hold as equalities. This set is called the active set and is denoted by $\mathcal{S}$. For a given active set $\mathcal{S}$, the associated eHQP is obtained by selecting the rows of $\mathcal{S}$. The two first algorithms are trivially adapted and denoted by $eHQP\_primal(\underline{A}_p, \underline{b}_p, \mathcal{S})$ and $eHQP\_dual(\underline{A}_p, \underline{b}_p, \mathcal{S})$. The algorithm starts from an initial feasible point. At each iteration, the corresponding eHQP is solved, and depending on the result, the active set is modified. See App. A for details.

First, the initial point needs only to satisfy (72), since for any x, it is easy to build w_2 so that (71) is satisfied. Among the possible w_2, the initial one is chosen with zeros on the components corresponding to inactive constraints. This property is kept along the iterations: the inactive components of w_2 are null. It is preserved at each step (73). It is of course preserved when activating any constraint and deactivating a constraint of A_1. When deactivating a constraint of A_2, the corresponding w_2 (which is also the Lagrange multiplier due to (34)) is strictly negative. Then setting it to zero keeps the constraint satisfaction while strictly decreasing the cost function. With this simple modification of the classical algorithm, the inactive slacks are kept to 0 during all the algorithm.

By recurrence over the algorithm iterations, it is straightforward to show that all the constraints of the second level $A_2 x^{(i)} \leq b_2$ are satisfied or active. For the active component, the slack variable satisfies $w_2^{(i)} = A_2 x^{(i)} - b_2$. From these observations, we see that it is not necessary to keep track of w_2 since it is nothing more than the residue of the constraints of the second level corresponding to the current $x^{(i)}$.

Based on these observations, a dedicated version of the classical active-search algorithm that computes the optimum of (70) is proposed. It is summarized in Alg. 3 and detailed below.

4.2.2 Initialization

The algorithm starts with an initial point $x^{(0)}$ satisfying the strict level (72) and an initial guess of the active set of the second level $\mathcal{S}_2^{(0)}$. The active set for the first level is deduced from $x^{(0)}$.

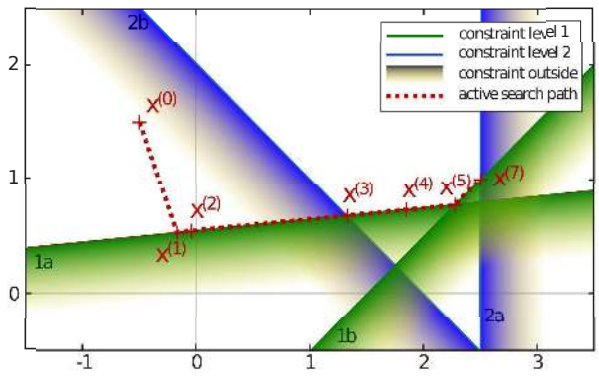


Fig. 2: *An example of execution in two dimensions. The initial point is $x^{(0)}$. The iterations of the algorithm are denoted by $x^{(1)}...x^{(7)}$. The optimum is finally reached in $x^{(7)}$. The constraints of the first levels are always respected, and the path is slidding along these constraints. From $x^{(3)}$, all the constraints of level 2 are satisfied or active. The first deactivation then occurs. The details of the execution are given in App. B.*

4.2.3 Step length

The active-search then maintains a value of the parameter $x^{(i)}$ and the active set $\mathcal{S}^{(i)}$. At each iteration, the algorithm first computes the optimum $x^{(*i)}$ of eHQP associated to $\mathcal{S}^{(i)}$. The current parameter is then moved toward $x^{(*i)}$:

$$x^{(i+1)} = x^{(i)} + \tau(x^{(*i)} - x^{(i)}) \tag{73}$$

The length τ of this step is computed for each constraint component by component. The minimum is then selected.

For the strict level A_1, τ is chosen to prevent inactive constraints of the first level from becoming violated:

$$\tau_{1,r} = \frac{b_{1,r} - A_{1,r}x^{(i)}}{A_{1,r}(x^{(*i)} - x^{(i)})} \tag{74}$$

with $A_{1,r}$ and $b_{1,r}$ the r^{th} row of A_1 and b_1. If the minimal τ corresponds to $\tau_{1,r}$, the corresponding constraint is activated.

Similarly for the relaxed level, if $x^{(i)}$ respects $A_{2,r}x^{(i)} \leq b_{2,r}$, the step length is computed to saturate the constraint. If $x^{(i)}$ does not respect $A_{2,r}x^{(i)} \leq b_{2,r}$, the constraint was not satisfied at the previous step. A full step $\tau = 1$ can be performed. As previously, the constraint r is activated if $\tau_{2,r}$ is the minimum. If several constraints correspond to the minimum, only one of them should be arbitrarily activated.

In summary, $\tau_{2,r}$ is chosen as:

$$\tau_{2,r} = \begin{cases} \frac{b_{2,r} - A_{2,r}x^{(i)}}{A_{2,r}(x^{(*i)} - x^{(i)})} & \text{if } A_{2,r}x^{(i)} \leq b_{2,r} \\ 1 & \text{otherwise} \end{cases} \tag{75}$$

The step-length algorithm is summarized in Alg. 4.

4.2.4 Positivity of the multiplier

If no constraint needs to be activated, it means that the optimum is reached if $\mathcal{S}^{(i)}$ is the optimal active set. This is checked by looking at the components of the Lagrange multipliers. The multiplier of the strict level is λ given by (63). The multiplier of the relaxed level is w_2. If the active set is optimal, then all the components of λ and w_2 are non negative. If one of them is negative, the constraint corresponding to the lowest component of the multipliers is deactivated.

4.2.5 Algorithm termination and proof of convergence

The algorithm stops at the first iteration where no constraint needs to be activated or deactivated. The current value of the parameter $x^{(i)}$ is the (primal) optimum of the iHQP. The current value of $\mathcal{S}^{(i)}$ gives the optimal active set.

When starting the iterative process, some of the constraints of A_2 may be inactive but also not satisfied. After a finite number of iterations (bounded by the total number of constraints of the problem), all the constraints of A_2 are satisfied or active. This is the end of a serie of iterations where only activations have occured. During this serie, the Lagrange multipliers are never computed: there cannot be any deactivations while some constraints of A_2 are still violated. The behavior of the algorithm is then the same as the classical active-search algorithm and thus is ensured to converge.

Algorithm 3 Active search for a strict and a relaxed constraints

1: **Input:** - $x^{(0)}$ such that $A_1 x^{(0)} \leq b_1$
2: - $\mathcal{S}_2^{(0)}$ an initial guess for the active set of A_2
3: **Output:** x^* minimizing (70)
4: $x = x^{(0)}$
5: $\mathcal{S} = \{r \ s.t. \ A_{1,r} x_0 = b_{1,r}\} \bigcup \mathcal{S}_2^{(0)}$
6: **repeat**
7: −−Compute the next optimum−−
8: $x^* =$ eHQP_primal $(\underline{A}_2, \underline{b}_2, \mathcal{S})$
9: −−Compute the step length using Alg. 4−−
10: τ,activate,cst = step_length$(\underline{A}_2, \underline{b}_2, x, x^*)$
11: $x := x + \tau(x^* - x)$
12: −−If necessary, increase the active set−−
13: **if** activate **then**
14: $\mathcal{S} := \mathcal{S} \bigcup \{cst\}$
15: **continue**
16: **end if**
17: −−If necessary, decrease the active set−−
18: $w, \lambda =$ eHQP_dual $(\underline{A}_2, \underline{b}_2, 2, \mathcal{S})$
19: $\nu, cst = \min\{\lambda, w\}$
20: **if** $\nu < 0$ **then**
21: $\mathcal{S} := \mathcal{S} \setminus \{cst\}$
22: **end if**
23: **until** not activate and $\nu > 0$
24: **return** $x^* := x$

Algorithm 4 Computation of the step length of Alg. 3

1: **function step_length** (A, b, x_0, x_1)
2: **Input:** Constraint A, b, current point x_0, target optimum x_1
3: **Output:** step length τ, activation boolean activate, constraint reference cst
4: activate := False
5: **for each** row r of A **do**
6: **if** $A_r x_0 \leq b_r$ **then**
7: $\tau[r] = \frac{b_r - A_r x_0}{A_r(x_1 - x_0)}$ **if** $A_r(x_1 - x_0) \neq 0$ **else** 1
8: **else**
9: $\tau[r] = 1$
10: **if** $A_r x_1 > b_r$ **then**
11: activate := True; cst := r
12: **end if**
13: **end if**
14: **end for each**
15: $\tau_{min} := \min\{\tau, 1\}$
16: **if** $\tau_{min} < 1$ **then**
17: activate := True; cst := $\arg\min\{\tau\}$
18: **end if**
19: **return** τ, activate, cst

4.3 Hierarchical active search

The main loop of Alg. 3 is composed of two sets of instructions: the first one (lines #7 to #15) concerns the activation of needed constraints, while the second one (lines #16 to #21) deals with the deactivation to obtain the optimal active set. The first part of the loop is incrementally building an active set such that all the constraints are activated or satisfied. It can be observed that the hierarchical relation between the levels has strictly no importance in this first part. Indeed, any value can be applied in the null space of the last level without disturbing the algorithm. Consequently, it is possible to search for the active set of all the levels at the same time.

However, the same remark is not possible in the second part of the loop. Indeed, the previous section has shown that the primal optimum can be computed for all the levels at the same time. There is however no direct solution to compute all the Lagrange multipliers at once, and each level must be explored successively to compute the corresponding multiplier.

These two remarks are used to build the hierarchical active search detailed below and summarized in Alg. 5.

4.3.1 Algorithm principle

The proposed algorithm is composed of two loops: an inner loop that first enforces then maintains the property that *all the constraints should be activated or satisfied*. And an outer loop that explores all the levels in ascending order to search for the corresponding optimal active set by removing unnecessary constraints.

4.3.2 Initialization

The algorithm starts with an initial guess of the active set of all the levels. It does not need an initial parameter $x^{(0)}$ since none of the levels (even the first one) is guaranteed to be feasible. It then starts with the arbitrary value $x^{(0)} = 0$.

4.3.3 Step length and activation

At each new inner iteration, the algorithm first computes the optimum of the eHQP associated to the current active set. It then computes the step length following the same rules as in the previous algorithm using (75) at all the levels.

Depending on the initial guess $\mathcal{S}^{(0)}$ and the corresponding eHQP optimum, some of the inactive constraints may be violated. In that case, the activation loop will first make some full steps $\tau = 1$, while each time adding one violated constraint into the active set. The number of such steps is bounded by the number of rows of $\underline{A}_p$. At the end of these steps, *all the constraints should be activated or satisfied*. This property is then maintained throughout all the following iterations.

4.3.4 Positivity of the multipliers and deactivation

When all the necessary constraints have been activated, the multipliers of the first level are computed (it was not necessary to test them before). As before, the active set is optimal with respect to the first level if none of the multiplier components is strictly negative. Otherwise, the constraint corresponding to the lowest component is deactivated, and a new inner iteration is started.

In [Kanoun et al., 2011], it was observed that, if one of the constraint of the first level is not satisfied (the slack variable is then strictly positive), this constraint can be changed into an equality for all the following levels, to avoid undesirable future deactivations. Similarly, when the first optimal active set is found, the multipliers of the second level can be computed and checked, but without deactivating a constraint whose slack at the first level was positive. The active set is optimal with respect to the second level if all the components of the multipliers are non negative or corresponds to a positive slack of the first level.

A constraint of the second level whose slack is positive will also be kept active for all the following levels. For the same reasons as observed in [Kanoun et al., 2011], if one of the constraints of the first level corresponds to a strictly positive component of the multiplier of the second level, this constraint can be *locked* to prevent any future deactivation.

In summary, the outer loop explores each level starting from the first one. At each level, it computes the multipliers. If a constraint is strictly negative and does not correspond to a strictly positive component of the multipliers of the previous levels, it is deactivated. When no more constraint needs to be deactivated, the constraint corresponding to strictly positive components of the multipliers are stored in the set $\mathcal{F}$ of locked constraints.

4.3.5 Algorithm termination and proof of convergence

The outer loop terminates after a fixed number of iterations when all the p levels have been explored. At this point, the active set is such that all the constraints are active or satisfied, and it is optimal for each of the levels.

As in the previous algorithm, the inner loop starts by a series of full step $\tau = 1$ until all the constraints are active or satisfied. Then, for a given iteration k of the outer loop, suppose that the active set of the levels $k + 1$ to p is constant: the inner loop behaves similarly to the previous algorithm and thus converges. In practice, the active set of the upper levels is not constant, but it can only be increased, and then can vary a finite number of times (bounded by the number of rows of the levels $k+1$ to p), which ensures the termination of the inner loop.

4.4 Lexicographic optimization

The algorithm deactivates a constraint if there exists a level k for which the component of the multiplier corresponding to the constraint is negative, while it

Algorithm 5 Hierarchical active search

1: **Input:** Initial guess $\mathcal{S}^{(0)}$
2: **Output:** x^* minimizing (27)
3: $x = 0$; $\mathcal{S} = \mathcal{S}^{(0)}$
4: $\mathcal{F} = \emptyset$
5: **for** $k = 1 : p$ **do**
6: **repeat**
7: −−Compute the next optimum−−
8: $x^* = \text{eHQP\_primal } (\underline{A}_p, \underline{b}_p, \mathcal{S})$
9: −−Compute the step length using Alg. 4−−
10: τ,activate,cst = step_length$(\underline{A}_p, \underline{b}_p, x, x^*)$
11: $x := x + \tau(x^* - x)$
12: −−If necessary, increase the active set−−
13: **if** activate **then**
14: $\mathcal{S} := \mathcal{S} \bigcup \{cst\}$
15: **continue**
16: **end if**
17: −−If necessary, decrease the active set−−
18: $w, \lambda = \text{eHQP\_dual } (\underline{A}_p, \underline{b}_p, k, \mathcal{S})$
19: $\lambda_{\mathcal{F}} := 0$
20: $\nu, cst = \min\{\lambda, w\}$
21: **if** $\nu < 0$ **then**
22: $\mathcal{S} := \mathcal{S} \setminus \{cst\}$
23: **continue**
24: **else**
25: $\mathcal{F} := \mathcal{S} \bigcup \{cst, \lambda_{cst} > 0\} \bigcup \{cst, w_{cst} > 0\}$
26: **break**
27: **end if**
28: **until not** activate **and** $\nu > 0$
29: **end for**
30: **return** $x^* := x$

is zero for all the multipliers of level $j < k$. Consider the pseudo-matrix Λ_p in (66), recalled here:

$$\Lambda_p = \begin{bmatrix} w_1^* & {}^1\underline{\lambda}_2 & \dots & {}^1\underline{\lambda}_{p-1} & {}^1\underline{\lambda}_p \\ & w_2^* & \dots & {}^2\underline{\lambda}_{p-1} & {}^2\underline{\lambda}_p \\ & & & \vdots & \vdots \\ & & & w_{p-1}^* & {}^{p-1}\underline{\lambda}_p \\ & & & & w_p^* \end{bmatrix} \quad (66)$$

A constraint is deactivated *iff* the corresponding row of Λ_p has one strictly negative component on column k preceded by only zeros for the columns $j < k$. In other words, the row is smaller than zero in the lexicographic sense:

$$\Lambda_p = \begin{bmatrix} 0 & \dots & 0 & -\alpha & \times & \times & \dots & \times \end{bmatrix} \prec 0 \quad (76)$$

Using this notation, Alg. 5 can be rewritten on the exact same form than Alg. 3, using a lexicographic test on Λ_p instead on lines #18-#19.

If a lexicographical writing of Alg. 3 is simpler, it is more efficient from a computational point of view to execute Alg. 5 since it avoids computing the whole Λ_p at each iteration.

4.5 Least-square solution

The eHQP algorithm gives the least-square solution among all the solutions of same cost, when z_{p+1} is set to 0. However, this is not the case of the iHQP since some constraints might be uselessly active. Indeed, there is no mechanism to deactivate the unnecessary constraints if they are not disturbing the optimum. To force the deactivation of the unnecessary constraints, an artificial last level can be added by setting $A_{p+1} = I$ and $b_{p+1} = 0$ that is to say:

$$x = 0 \quad (77)$$

This constraint will always be rank deficient, but its satisfaction at best in the least-square sense ensures that the constraints that artificially increase the norm of x are deactivated: the optimal active set is unique and the the returned optimum is the least-square one.

4.6 Conclusion

Based on the hierarchical decomposition, we have proposed an active-search algorithm that solves the iHQP problem. On the contrary to the cascade of QP used in [Kanoun et al., 2011, De Lasa et al., 2010], this algorithm computes the active set of all the level of the hierarchy at the same time. It thus solves the complete hierarchical problem at once, avoiding the back-and-forth effects encoutered with cascades and sparing the cost of computation.

One of the costly computation of the algorithm is the HCOD. This should be computed once at the

start of the loop. Then, the initial decomposition can be updated to follow the changes of the active set. This will be detailed in the second part [Escande et al., 2012].

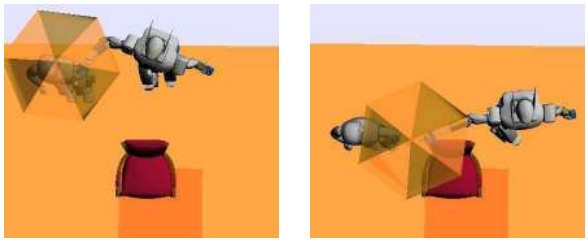
5 Example of use

We now briefly illustrate with a small example the use of our solver to generate a motion with a humanoid robot. At each control cycle of the robot, the motion objectives are written as a set of linear equalities and inequalities as formulated in (1). The robot velocity is then computed by inverse kinematics. A more detailed description will be given in the second part of the paper (see [Escande et al., 2012], Sec. 5.2). In the presented motion, the robot HRP-2 is walking beside a companion robot and should keep it below an umbrella carried in its right hand. The objectives to satisfy are

- e_{jl}: avoid the joint limits
- e_{pg}: track the center-of-mass and flying foot references issued from a walking pattern generator [Kajita et al., 2003, Stasse et al., 2008b]
- e_{θ}: keep the angle of the umbrella stick below a fixed threshold
- e_{umb}: keep the companion below the umbrella
- e_{fov}: look at the companion head
- $e_{\theta min}$: minimize the stick angle

The task e_{jl}, e_{θ}, e_{umb} and e_{fov} are expressed as inequalities. The constraints corresponding to e_{θ} and $e_{\theta min}$ have the same left side, but the first one enables a motion within the given boundaries while the second one tries to prevent any motion by regulating the stick at the vertical. The order of the tasks in the hierarchy follows the list order. The obtained motion is summarized by Figures 3 to 5.

At the beginning, HRP-2 is close to its companion (see Fig. 3-(a)): all the constraints can be satisfied and the companion is safely below the umbrella. HRP-2 has then to step on the right (following a set of given footprints) to avoid an obstacle, which prevents him from staying close to the companion (see Fig. 3-(b)). The task e_{umb} cannot be executed while



(a) (b)

Fig. 3: *Walking with an umbrella: snapshots from a top view. (a) the robot manages to keep its companion below the umbrella (b) the robot does not manage to accomplish the secondary task while following the references of the walking pattern generator.*

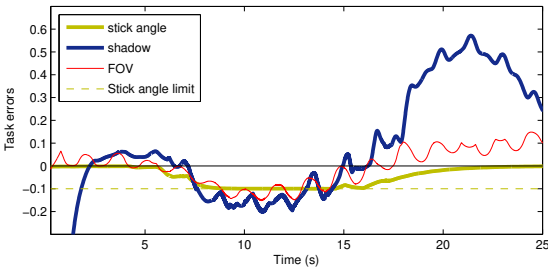


Fig. 4: *Walking with an umbrella: task errors. The companion starts outside of the umbrella shadow and the robot first brings the umbrella to satisfy this objective. When going around the obstacle, the task regulating the angle of the umbrella stick is violated first ($T = 5.8s$), followed quickly by the FOV task ($T = 5.9s$). One second later, the task keeping the companion below the umbrella is also violated ($T = 6.3s$). The angle of the stick then reaches the minimum limit ($T = 8.1s$). This fourth task having priority over the three others, the angle is blocked. Finally, the robot reaches a region where the tasks are feasible (from $T = 15s$).*

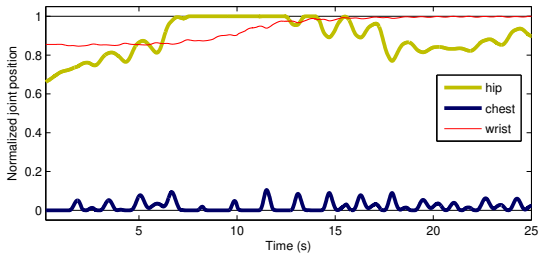


Fig. 5: *Walking with an umbrella: normalized positions of some typical joints. The joint limits are respected at any time.*

satisfying the other tasks. Fig. 4 shows the evolution of the errors of the four tasks of least priority: the task $e_{\theta min}$ is first relaxed at $T = 5.8s$, quickly followed by e_{fov} and e_{umb}. When the angle of the stick reaches the maximal value authorized by e_θ, it stops and remains constant. The robot then continues to walk, with the umbrella slightly tilted and as close as possible to the companion. The tasks become satisfied again as soon as the robot reaches a feasible region. The walking task and the joint limits are satisfied during all the motion, as shown in Fig. 5.

This example illustrates both the hierarchy and the interest of inequalities objectives: when the set of objectives is infeasible, some are perfectly satisfied, while the others, less important, are successively relaxed to keep a safe behavior. This enables to implement in particular avoidance behavior (the joint limits were never violated) and preference area (such as the umbrella shadows) or postural tasks (such as the tilt angle of the stick).

6 Conclusion

In this first part of the paper, we have proposed a method to solve a hierarchical least-square quadratic problem. First, we have focused on the subproblem where only equalities are involved. In that case, the optimal solution corresponds to what is classically computed in inverse kinematics when handling redundancy by successively projecting the next problem in the null space of the previous ones [Siciliano and Slotine, 1991]. We described a dedicated matrix decomposition, the HCOD, that reveals the structure of the problem and can be used to explain the obtained optimum, for example for automatic supervision. With this decomposition, the optimum is obtained using a single inversion. It can also be used to compute the optimum of the dual problem.

Subsequently, we have used this first solver to build an active-search algorithm that solves the whole problem involving inequalities. Contrary to previous methods [Kanoun et al., 2011, De Lasa et al., 2010] that use a cascade of QP, our algorithm solves the problem for all the levels of the hierarchy at the same time. This avoids activations and deactivations encountered when using a cascade, which is undesirable for computation-time reasons but also to avoid numerical instability and bad numerical behavior of the algorithm in general.

In the second part of the paper [Escande et al., 2012], we focus more on practical implementation of the solver. In particular, a method to compute the HCOD and to update it along the active-search iterations is detailed. The theoretical properties of the solver are given (complexity, continuity, stability). And a strong experimental setup demonstrates the interest of the approach.

A Some basics on the active search

The classical active-search algorithm to solve an iQP is quickly recalled. We consider the following generic least-square form:

$$\min_{\chi} \ \|A\chi - b\| \tag{78}$$

$$\text{subject to} \quad C\chi \leq d \tag{79}$$

As in the rest of the paper, only upper bounds are considered here.

A.1 Algorithm principle

For a given χ, a row i of (79) is *satisfied* if $C_i\chi \leq d_i$, *violated* if $C_i\chi > d_i$ and *saturated* if $C_i\chi = d_i$. The constraint (79) defines a polytope in the parameter space. When the optimum of the unconstraint cost function is outside of this polytope, the optimum of the constrained QP is on the border of the polytope, *i.e.* at the saturation of a subset of (79), named the optimal active set. The active search iterates on a candidate active set, denoted by $\mathcal{S}$, until it finds the optimal one. The constraints of $\mathcal{S}$ are said to be *active*. For a given $\mathcal{S}$, an associated eQP can be defined by considering only the active constraints as equalities. If we know the optimal active set, the optimum of the iQP is obtained by solving the associated eQP.

This algorithm could be used directly to compute the optimum of a QP with a slack variable, such as the one defined in (70), using $\chi = (x, w_2)$, but in a less optimal manner than with our method. The algorithm is summarized in Alg. 6 and detailed below.

A.2 Initialization

The active search starts with a feasible point $\chi^{(0)}$ that satisfies (79) and an initial guess of $\mathcal{S}^{(0)}$ (for example, the set of all the saturated constraints at $\chi^{(0)}$). The initial point $\chi^{(0)}$ is not trivial and requires an iterative process, called *Phase I*, to be solved (typically a simplex). It will not be detailed here since this first phase of the algorithm is not necessary with the hierarchical formulation.

A.3 Step length

At each iteration i, the algorithm computes the optimum $\chi^{(*i)}$ of the eQP associated to the current active set. It then moves the current $\chi^{(i)}$ toward the eQP optimum:

$$\chi^{(i+1)} = \chi^{(i)} + \tau(\chi^{(*i)} - \chi^{(i)}) \tag{80}$$

If any constraint is reached during the move, then τ is chosen to saturate the most constraining constraint, which is added to the active set $\mathcal{S}_i$. The algorithm starts a new iteration.

A.4 Positivity of the multipliers

Otherwise, a full step $\tau = 1$ is performed. In that case, $\chi^{(i)}$ is optimal if the active set is optimal, which is true if the Lagrange multipliers are all nonnegative (this is one of the Karush-Kuhn-Tucker iQP optimality conditions [Boyd and Vandenberghe, 2004]). If any multiplier is negative, the current active set is suboptimal and the corresponding constraint needlessly prevents to come closer to the optimum. The constraint corresponding to the multiplier with lowest value is removed from the active set $\mathcal{S}^{(i)}$ and the algorithm starts a new iteration.

Algorithm 6 Classical active search

Input: $\chi^{(0)}$ s.t. $C\chi^{(0)} \leq d$
Output: χ^* minimizing (78)
$\mathcal{S}_0 = \{k \ s.t. \ C_k\chi^{(0)} = d_k\}$
$\chi, \mathcal{S} := \chi^{(0)}, \mathcal{S}_0$
repeat
  $--$Compute eQP optimum$--$
  $\chi^* = C_{\mathcal{S}}^{+} d_{\mathcal{S}} + Z_C(AZ_C)^{+}(b - AC_{\mathcal{S}}^{+} d_{\mathcal{S}})$
  $--$Compute the step length$--$
  $\forall i \notin \mathcal{S}, \ \tau_i = \frac{C_i\chi - d_i}{C_i(\chi - \chi^*)}$
  $\tau, cst = \min\{1, \tau_i\}$
  $\chi := \chi + \tau(\chi^* - \chi)$
  $--$If necessary, increase the active set$--$
  if $\tau < 1$ **then**
    $\mathcal{S} := \mathcal{S} \bigcup \{cst\}$
    continue
  end if
  $--$If necessary, decrease the active set$--$
  $\lambda = -C_{\mathcal{S}}^{+T} A^T(A\chi - b)$
  $\nu, cst = \min\{\lambda\}$
  if $\nu < 0$ **then**
    $\mathcal{S} := \mathcal{S} \setminus \{cst\}$
  end if
until $\tau = 1$ and $\nu \geq 0$
return $\chi^* := \chi$

A.5 Algorithm termination and convergence

Finally, the loops ends when both $\tau = 1$ and all the multipliers components are nonnegative. The convergence of the loop in a finite time can be proven by showing the strict decrease of the cost function at each iteration [Boyd and Vandenberghe, 2004].

B A simple example of active search

Fig. 2 gives an example of execution of Alg. 3 in 2 dimensions. The problem to solve is composed of

two levels. The first (strict) level is:

$$1a: \quad \frac{x}{10} - y \leq -0.55 \tag{81}$$

$$1b: \quad x - y \leq 1.5 \tag{82}$$

The second (relaxed) level is:

$$2a: \quad x \geq 2.5 \tag{83}$$

$$2b: \quad x + y \geq 2 \tag{84}$$

The algorithm starts at the initial point $x^{(0)} = [-0.5, 1.5]$, which satisfies the constraints *1a* and *1b* of the first level, but not the constraints *2a* and *2b* of the second level. All constraints are inactive. For each step of the algorithm, we give the optimum of the eHQP $x^{(*i)}$, the step, the change in the active set and the active set at the end of the iteration $\mathcal{S}^{(i)}$.

- computation of $x^{(1)}$: we have $x^{(*0)} = [0, 0]$ but *1a* prevents a full step and is activated. $\mathcal{S}^{(1)} = \{1a\}$

- $x^{(2)}$: $x^{(*1)}$ is the projection of $[0, 0]$ on *1a*. A full step is taken, but the current point x is still outside of *2a* and *2b*. Both could be activated. *2b* is chosen (arbitrarily, due to the constraint order). $\mathcal{S}^{(2)} = \{1a, 2b\}$

- $x^{(3)}$: $x^{(*2)}$ is the intersection of the two active constraints. A full step is taken, but x still violates *2a*, which is activated. $\mathcal{S}^{(3)} = \{1a, 2a, 2b\}$. From this point, all the constraints are satisfied or active.

- $x^{(4)}$: $x^{(*3)}$ is the point on the boundary of *1a* at the least-square distance of *2a* and *2b*. A full step is taken and no constraint is activated. The Lagrangian of the first level is null since x is on *1a*. The Lagragian of the second level corresponding to *2b* (*i.e.* w_2^*) is negative: *2b* is deactivated. $\mathcal{S}^{(4)} = \{1a, 2a\}$

- $x^{(5)}$: $x^{(*4)}$ is at the intersection of *1a* and *2a*, but *1b* prevents to perform a full step and is activated. $\mathcal{S}^{(5)} = \{1a, 1b, 2a\}$

- $x^{(6)}$: we have $x^{(*5)} = x^{(5)}$. A full step of length 0 is taken. The Lagrange multiplier is computed, and is negative on the component corresponding to *1a*: the constraint is deactivated. $\mathcal{S}^{(6)} = \{1b, 2a\}$.

- $x^{(7)}$: $x^{(*6)}$ is the intersection of *1b* and *2a*. A full step is taken and all Lagrange multipliers are positive: the optimum is found.

Acknowledgments

This work was supported by grants from the RobotHow.cog EU CEC project, Contract No. 288533 under the 7th Research program (`www.robohow.eu`), and French FUI Project ROMEO. Many thanks to C. Benazeth, A. El Khoury, F. Keith, A. Kheddar and F. Lamiraux for their help.

References

[Antonelli et al., 2010] Antonelli, G., Arrichiello, F., and Chiaverini, S. (2010). Flocking for multi-robot systems via the null-space-based behavioral control. *Swarm Intelligence*, 4(1):37–56.

[Antonelli and Chiaverini, 1998] Antonelli, G. and Chiaverini, S. (1998). Task-priority redundancy resolution for underwater vehicle-manipulator systems. In *IEEE Int. Conf. on Robotics and Automation (ICRA'98)*, Leuven, Belgium.

[Antonelli and Chiaverini, 2006] Antonelli, G. and Chiaverini, S. (2006). Kinematic control of platoons of autonomous vehicles. *IEEE Trans. on Robotics*, 22(6):1285–1292.

[Baerlocher and Boulic, 2004] Baerlocher, P. and Boulic, R. (2004). An inverse kinematic architecture enforcing an arbitrary number of strict priority levels. *The Visual Computer*, 6(20):402–417.

[Behringer, 1977] Behringer, F. (1977). Lexicographic quasiconcave multiobjective programming. *Zeitschrift für Operations Research*, pages 103–116.

[Berenson et al., 2011] Berenson, D., Srinivasa, S., and Kuffner, J. (2011). Task space regions: A framework for pose-constrained manipulation planning. *Int. Journal of Robotics Research*, 30(12):1435–1460.

[Björck, 1996] Björck, A. (1996). *Numerical Methods for Least Squares Problems*. SIAM.

[Bouyarmane and Kheddar, 2011] Bouyarmane, K. and Kheddar, A. (2011). Multi-contact stances planning for multiple agents. In *IEEE Int. Conf. on Robotics and Automation (ICRA'11)*, Shangai, China.

[Boyd and Vandenberghe, 2004] Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

[Chang and Dubey, 1995] Chang, T. and Dubey, R. (1995). A weighted least-norm solution based scheme for avoiding joints limits for redundant manipulators. *IEEE Trans. on Robotics and Automation*, 11(2):286–292.

[Chaumette and Marchand, 2001] Chaumette, F. and Marchand, E. (2001). A redundancy-based iterative scheme for avoiding joint limits: Application to visual servoing. *IEEE Trans. on Robotics and Automation*, 17(5):719–730.

[Chiaverini, 1997] Chiaverini, S. (1997). Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. *IEEE Trans. on Robotics and Automation*, 13(3):398–410.

[Chiaverini et al., 2008] Chiaverini, S., Oriolo, G., and Walker, I. (2008). Kinematically redundant manipulators. In Siciliano, B. and Khatib, O., editors, *Handbook of Robotics*, page 245268. Springer-Verlag.

[Collette et al., 2007] Collette, C., Micaelli, A., Andriot, C., and Lemerle., P. (2007). Dynamic balance control of humanoids for multiple grasps and noncoplanar frictional contacts. In *IEEE-RAS Int. Conf. on Humanoid Robots (Humanoid'07)*, Pittsburgh, USA.

[Dantzig and Thapa, 2003] Dantzig, G. and Thapa, M. (2003). *Linear programming: theory and extensions*. Springer Verlag, 2nd edition.

[De Lasa et al., 2010] De Lasa, M., Mordatch, I., and Hertzmann, A. (2010). Feature-based locomotion controllers. In *ACM SIGGRAPH'10*.

[De Schutter and Van Brussel, 1988] De Schutter, J. and Van Brussel, H. (1988). Compliant robot motion i. a formalism for specifying compliant motion tasks. *Int. Journal of Robotics Research*, 7(4):3–17.

[Decré et al., 2009] Decré, W., Smits, R., Bruyninckx, H., and De Schutter, J. (2009). Extending itasc to support inequality constraints and non-instantaneous task specification. In *IEEE Int. Conf. on Robotics and Automation (ICRA'09)*, Kobe, Japan.

[Deo and Walker, 1992] Deo, A. and Walker, I. (1992). Robot subtask performance with singularity robustness using optimal damped least squares. In *IEEE Int. Conf. on Robotics and Automation (ICRA'92)*, pages 434–441, Nice, France.

[Escande et al., 2010] Escande, A., Mansard, N., and Wieber, P.-B. (2010). Fast resolution of hierarchized inverse kinematics with inequality constraints. In *IEEE Int. Conf. on Robotics and Automation (ICRA'10)*, Anchorage, USA. (preliminary version of this journal paper).

[Escande et al., 2012] Escande, A., Mansard, N., and Wieber, P.-B. (2012). Hierarchical quadratic programming (part 2). *Submitted*.

[Garcia-Aracil et al., 2005] Garcia-Aracil, N., Malis, E., Aracil-Santonja, R., and Perez-Vidal, C. (2005). Continuous visual servoing despite the changes of visibility in image features. *IEEE Trans. on Robotics*, 21(6):415–421.

[Gienger et al., 2006] Gienger, M., Janben, H., and Goerick, C. (2006). Exploiting task intervals for whole body robot control. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'06)*, Beijin, China.

[Golub and Van Loan, 1996] Golub, G. and Van Loan, C. (1996). *Matrix computations*, chapter 5.5: The rank-deficient LS problem. John Hopkins University Press, 3rd edition.

[Hanafusa et al., 1981] Hanafusa, H., Yoshikawa, T., and Nakamura, Y. (1981). Analysis and control of articulated robot with redundancy. In *IFAC, 8th Triennal World Congress*, volume 4, pages 1927–1932, Kyoto, Japan.

[Herdt et al., 2010] Herdt, A., Diedam, H., Wieber, P., Dimitrov, D., Mombaur, K., and Diehl, M. (2010). Online walking motion generation with automatic footstep placement. *Advanced Robotics*, 24(5-6):719–737.

[Hofmann et al., 2009] Hofmann, A., Popovic, M., and Herr, H. (2009). Exploiting angular momentum to enhance bipedal center-of-mass control. In *IEEE Int. Conf. on Robotics and Automation (ICRA'09)*, Kobe, Japan.

[Isermann, 1982] Isermann, H. (1982). Linear lexicographic optimization. *Operations Research Spektrum*, 4:223–228.

[Kajita et al., 2003] Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Harada, K., Yokoi, K., and Hirukawa,

H. (2003). Biped walking pattern generation by using preview control of zero-moment point. In *IEEE Int. Conf. on Robotics and Automation (ICRA'03)*, Taipei, Taiwan.

[Kanoun et al., 2011] Kanoun, O., Lamiraux, F., and Wieber, P.-B. (2011). Kinematic control of redundant manipulators: generalizing the task priority framework to inequality tasks. *IEEE Trans. on Robotics*, 27(4):785–792.

[Khatib, 1986] Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *Int. Journal of Robotics Research*, 5(1):90–98.

[Khatib, 1987] Khatib, O. (1987). A unified approach for motion and force control of robot manipulators: The operational space formulation. *International Journal of Robotics Research*, 3(1):43–53.

[Khatib et al., 2008] Khatib, O., Sentis, L., and Park, J. (2008). A unified framework for whole-body humanoid robot control with multiple constraints and contacts. In *European Robotics Symposium*, pages 303–312, Prague, Czech Republic.

[Khatib et al., 1996] Khatib, O., Yokoi, K., Chang, K., Ruspini, D., Holmberg, R., and Casal, A. (1996). Vehicle/arm coordination and multiple mobile manipulator decentralized cooperation. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'96)*, Osaka, Japan.

[Li et al., 2012] Li, T., Kermorgant, O., and Krupa, A. (2012). Maintaining visibility constraints during telechography with ultrasound visual servoing. In *IEEE Int. Conf. on Robotics and Automation (ICRA'12)*, Saint Paul, USA.

[Liégeois, 1977] Liégeois, A. (1977). Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Trans. on Systems, Man and Cybernetics*, 7(12):868–871.

[Mansard and Chaumette, 2007a] Mansard, N. and Chaumette, F. (2007a). Directional redundancy. *IEEE Trans. on Automatic Control*, 54(6):1179–1192.

[Mansard and Chaumette, 2007b] Mansard, N. and Chaumette, F. (2007b). Task sequencing for sensor-based control. *IEEE Trans. on Robotics*, 23(1):60–72.

[Mansard et al., 2007] Mansard, N., Stasse, O., Chaumette, F., and Yokoi, K. (2007). Visually-guided grasping while walking on a humanoid robot. In *IEEE Int. Conf. on Robotics and Automation (ICRA'07)*, Roma, Italia.

[Mansard et al., 2009] Mansard, N., Stasse, O., Evrard, P., and Kheddar, A. (2009). A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks. In *IEEE Int. Conf. on Advanced Robotics (ICAR'09)*, Munich, Germany.

[Marchand and Hager, 1998] Marchand, E. and Hager, G. (1998). Dynamic sensor planning in visual servoing. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'98)*, Leuven, Belgium.

[Mordatch et al., 2012] Mordatch, I., Todorov, E., and Popović, Z. (2012). Discovery of complex behaviors through contact-invariant optimization. In *ACM SIGGRAPH'12*, Los Angeles, USA.

[Nelson and Khosla, 1995] Nelson, B. and Khosla, P. (1995). Strategies for increasing the tracking region of an eye-in-hand system by singularity and joint limits avoidance. *Int. Journal of Robotics Research*, 14(3):255–269.

[Nemirovski, 1994] Nemirovski, A. (1994). *Efficient methods in convex programming.* Technion, the Israel institute of technology. Fall Semester 1994/95.

[Nenchev, 1989] Nenchev, D. (1989). Redundancy resolution through local optimization: A review. *Journal of Robotic Systems*, 6(6):769–798.

[Padois et al., 2007] Padois, V., Fourquet, J.-Y., and Chiron, P. (2007). Kinematic and dynamic model-based control of wheeled mobile manipulators: a unified framework for reactive approaches. *Robotica, Cambridge University Press*, 25(2):157–173.

[Park and Khatib, 2006] Park, J. and Khatib, O. (2006). Contact consistent control framework for humanoid robots. In *IEEE Int. Conf. on Robotics and Automation (ICRA'06)*, Orlando, USA.

[Pham and Nakamura, 2012] Pham, Q. and Nakamura, Y. (2012). Affine trajectory deformation for redundant manipulators. In *Robotics: Science and Systems (RSS'12)*, Sydney, Australia. Best Paper Award.

[Raunhardt and Boulic, 2007] Raunhardt, D. and Boulic, R. (2007). Progressive clamping. In *IEEE Int. Conf. on Robotics and Automation (ICRA'07)*, Roma, Italy.

[Remazeilles et al., 2006] Remazeilles, A., Mansard, N., and Chaumette, F. (2006). Qualitative visual servoing: application to the visibility constraint. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'06)*, Beijing, China.

[Salini et al., 2009] Salini, J., Barthélemy, S., and Bidaud, P. (2009). Lqp controller design for generic whole body motion. In *IEEE Int. Conf. on Robotics and Automation (ICRA'09)*, Kobe, Japan.

[Samson et al., 1991] Samson, C., Le Borgne, M., and Espiau, B. (1991). *Robot Control: the Task Function Approach*. Clarendon Press, Oxford, United Kingdom.

[Sentis, 2007] Sentis, L. (2007). *Synthesis and Control of Whole-Body Behaviors in Humanoid Systems*. PhD thesis, Stanford University, USA.

[Sian et al., 2005] Sian, N., Yokoi, K., Kajita, S., Kanehiro, F., and Tanie, K. (2005). A switching command-based whole-body operation method for humanoid robots. *IEEE/ASME Transactions on Mechatronics*, 10(5):546–559.

[Siciliano and Slotine, 1991] Siciliano, B. and Slotine, J.-J. (1991). A general framework for managing multiple tasks in highly redundant robotic systems. In *IEEE Int. Conf. on Advanced Robotics (ICAR'91)*, Pisa, Italy.

[Stasse et al., 2008a] Stasse, O., Escande, A., Mansard, N., Miossec, S., Evrard, P., and Kheddar, A. (2008a). Real-time (self)-collision avoidance task on a HRP-2 humanoid robot. In *IEEE Int. Conf. on Robotics and Automation (ICRA'08)*, Pasadena, USA.

[Stasse et al., 2008b] Stasse, O., Verrelst, B., Wieber, P.-B., Vanderborght, B., Evrard, P., Kheddar, A., and Yokoi, K. (2008b). Modular architecture for humanoid walking pattern prototyping and experiments. *Advanced Robotics, Special Issue on Middleware for Robotics –Software and Hardware Module in Robotics System*, 22(6):589–611.

[Sugihara, 2011] Sugihara, T. (2011). Solvability-unconcerned inverse kinematics by the levenberg-marquardt method. *IEEE Trans. on Robotics*, 27(5):984–991.

[Sung et al., 1996] Sung, Y., Cho, D., and Chung, M. (1996). A constrained optimization approach to resolving manipulator redundancy. *Journal of robotic systems*, 13(5):275–288.

[Wampler, 1986] Wampler, C. (1986). Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):93–101.

[Whitney, 1972] Whitney, D. (1972). The mathematics of coordinated control of prosthetic arms and manipulators. *Trans. ASME Journal of Dynamic System, Measures and Control*, 94:303–309.

[Yoshikawa, 1985] Yoshikawa, T. (1985). Manipulability of robotic mechanisms. *Int. Journal of Robotics Research*, 4(2):3–9.

Hierarchical Quadratic Programming
Part 2: Implementation and Experimentations

Adrien Escande
JRL-CNRS/AIST
Tsukuba, Japan

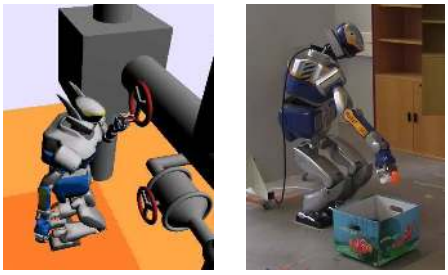
Nicolas Mansard
LAAS-CNRS, Univ. Toulouse
Toulouse, France

Pierre-Brice Wieber
INRIA Grenoble
St Ismier, France

Abstract

Hierarchical least-square optimization is often used in robotics to inverse a direct function when multiple incompatible objectives are involved. Typical examples are inverse kinematics or dynamics. The objectives can be given as equalities to be satisfied (e.g. point-to-point task) or as areas of satisfaction (e.g. the joint range). This two-part paper proposes a complete solution to resolve multiple least-square quadratic problems of both equality and inequality constraints ordered into a strict hierarchy. Our method is able to solve a hierarchy of only equalities ten time faster than the classical method and can consider inequalities at any level while running at the typical control frequency on whole-body size problems. This generic solver is used to resolve the redundancy of humanoid robots while generating complex movements in constrained environment.

In the first part of the paper, we proposed a dedicated numerical solver to solve equality-only and inequality hierarchical least-square quadratic problems. In this second part, we detail the implementation of the solver and its application to inverse kinematics. In particular, we explicit the solver complexity and prove the continuity and the stability of the resulting control laws. Finally, we experimentally demonstrate its efficiency in the context of generating robot motions.



(a) (b)

Fig. 1: *Various situations of inequality and equality constraints. (a) Obstacle avoidance, joint limits and support polygon. (b) Reaching an object behind an obstacle while ensuring the robot balance. The visibility is only ensured if possible.*

1 Introduction

Dealing with multiple contradictory objectives is a classical situation with redundant robots. In that case, imposing a hierarchy between the objectives can be an elegant solution to handle the conflicts while ensuring a safe behavior, by setting important objectives at the top of the hierarchy. A typical situtation is the design of a control law on a humanoid robot: manipulation tasks should be perfomed only within the stability region of the robot, while postural or observation tasks can be only partially realized (see Fig. 1).

Hierarchy of constraints expressed as linear equalities has been addressed early [Liégeois, 1977,

Hanafusa et al., 1981, Siciliano and Slotine, 1991]. Inequality-based objectives were first taken into account at the lowest-priority level using potential fields [Khatib, 1986]. However, some top-priority objectives in robotics are often expressed as inequalities (e.g. joint limits or obstacles avoidance). In [Kanoun et al., 2011], it was proposed to build a hierarchy of equalities and inequalities using a cascade of least-square quadratic problems (QP). However, the computation cost of the proposed method prevented any application for on-board robot control. A cheaper solution also based on QP cascades was proposed in [De Lasa et al., 2010] in a simplified case: inequalities are considered only at the first priority level, which is restrictive for simple case (e.g. Fig. 1-(a), the gaze is directly expressed as an inequality and set up as a third-priority objectives).

In the first part of the paper [Escande et al., 2012], we proposed a dedicated method to solve hierarchical problems without using a cascade of QP. For this purpose, we first proposed a matrix decomposition, named HCOD, dedicated to the hierarchical structure of the problem. Using this decomposition, the hierarchical problem with only equality constraints is solved in a single inversion. Based on the resolution of the equality problem, a dedicated active-search algorithm was proposed, that solves the general problem while avoiding the large number of iterations typically encoutered when using a cascade of solvers. Moreover, the intermediary theoretical study emphasizes the underlying hierarchical structure of the problem and allows a better understanding that could be used for automatic supervision or automatic task sequencing.

The contributions and organization of this second part are the followings:

- we give the method to compute the HCOD and to modify it when the active set increases or decreases in Sec. 3.
- we compute the complexity of the hierachical solvers and prove the continuity of the solver optimum and the stability of the subsequent inverse-kinematics based control law in Sec. 4.
- we experimentally show in Sec. 5 the solver efficiency and its use in the robotics context to generate whole-body motions of a humanoid robot.

2 Background

In the first part of the paper, we proposed a solution to solve the following hierarchical problem:

$$\underset{x,w_1 \dots w_p}{\operatorname{lex\,min}} \ \{\|w_1\|, \|w_2\|, \dots, \|w_p\|\}\,. \tag{1}$$

$$\text{subject to} \quad \forall k = 1..p,\ A_k x \leq b_k + w_k$$

where p is the number of levels of the hierarchy, and for any $k = 1..p$, A_k and b_k define the linear objectives to be satisfied at best in the least-square sense. This problem is formally defined by a cascade of p QP:

$$\min_{x_k, w_k} \ \|w_k\| \tag{2}$$

$$\text{subject to} \quad A_k x_k \leq b_k + w_k \tag{3}$$

$$\underline{A}_{k-1} x_k \leq \underline{b}_{k-1} + \underline{w}^*_{k-1} \tag{4}$$

where $\underline{A}_{k-1}$, $\underline{b}_{k-1}$ and $\underline{w}^*_{k-1}$ are the matrix and vectors composed of the stacked quantities of levels 1 to $k-1$ (by convention, they are empty matrix and vectors for $k-1=0$). $\underline{w}^*_{k-1}$ is the fixed value obtained from the QP of level $k-1$[1]. To solve this hierarchical least-square quadratic problem (HQP), it was previously proposed in [Kanoun et al., 2011, De Lasa et al., 2010] to use the cascade formulation by iteratively calling p times a QP solver. Such a cascade execution has some bad properties that are experimentally highlighted in Sec. 5. Instead, we proposed in the first part a hierarchical solver that is quickly recalled below.

2.1 Equality hierachical quadratic program

First, a subproblem of the HQP composed only of equality (eHQP) is solved (see the details

[1]For clarity purpose and without loss of generality, the problem is formulated with upper-bound constraints only. Efficient implementations would make explicit lower bounds, double bounds and equality (twin bounds) constraints. See App. D for details.

in [Escande et al., 2012], Sec. 3). The resolution of an eHQP is classical in inverse kinematics and dynamics, and was first formalized in [Siciliano and Slotine, 1991]. In the first part, we proposed a theoretical study that emphasizes the underlying mathematical structures and accelerates the resolution of this problem. We also gave a solution to compute the optimum of the dual problem. This method is based on the hierarchical complete orthogonal decomposition of the matrix $\underline{A}_k$ (see [Escande et al., 2012], Sec. 3.3):

$$\begin{bmatrix}\underline{A}_{k-1}\\ A_k\end{bmatrix} = \begin{bmatrix}\underline{W}_{k-1} & 0 & 0\\ 0 & V_k & U_k\end{bmatrix} \begin{bmatrix}\underline{H}_{k-1} & 0 & 0\\ N_k & 0 & 0\\ M_k & L_k & 0\end{bmatrix} [\underline{Y}_{k-1}\, Y_k\, Z_k]^T$$
$$= \underline{W}_k \begin{bmatrix}\underline{H}_k & 0\end{bmatrix} \begin{bmatrix}\underline{Y}_k & Z_k\end{bmatrix}^T = \underline{W}_k\ \underline{H}_k\ \underline{Y}_k^T \qquad (5)$$

where $W_k = \begin{bmatrix}V_k & U_k\end{bmatrix}$, Z_k is a null-space basis of $\underline{A}_k$, L_k is full rank lower triangular and $W_k, L_k, \begin{bmatrix}Y_k & Z_k\end{bmatrix}$ is the complete orthogonal decomposition of $A_k Z_{k-1}$:

$$A_k Z_{k-1} Z_{k-1}^T = W_k \begin{bmatrix}0 & 0\\ L_k & 0\end{bmatrix} \begin{bmatrix}Y_k & Z_k\end{bmatrix}^T \qquad (6)$$

The number of rows of A_k is denoted by m_k. The rank of $A_k Z_{k-1}$ is denoted $r_k \leq m_k$. The rank of $\underline{A}_k$ is denoted $\underline{r}_k = \sum_{j=1}^{k} r_j$.

Based on this decomposition, the hierarchical inverse of $\underline{A}_k$ is defined by (see [Escande et al., 2012], Sec. 3.4):

$$\underline{A}_p^\ddagger = \underline{Y}_p \underline{H}_k^\ddagger \underline{W}_p^T \qquad (7)$$

where $\underline{H}_k^\ddagger$ is defined by the following recurrence:

$$\underline{H}_k^\ddagger = \underline{Y}_p \begin{bmatrix}\underline{H}_{k-1}^\ddagger & 0 & 0\\ -L_k^{-1} M_k \underline{H}_{k-1}^\ddagger & 0 & L_k^{-1}\end{bmatrix} \underline{W}_p^T \qquad (8)$$

The optimum of the eHQP is then simply:

$$x_p^* = \underline{A}_p^\ddagger \underline{b}_p \qquad (9)$$

Or, using again the HCOD, $x_p^* = \underline{Y}_p \underline{y}_p^*$, with

$$\underline{y}_k^* = \begin{bmatrix}\underline{y}_{k-1}^*\\ y_k^*\end{bmatrix} \qquad (10)$$

and $y_k^* = L_k^{-1}(U_k^T b_k - M_k \underline{y}_{k-1}^*)$. The associated optimal w_k^* can be simply computed using the difference $w_k^* = A_k x_p^* - b_k$. A less expensive reformulation can also be expressed using the HCOD. The computation of the primal optimum is recalled in Alg. 1 of the first part of the paper[2].

For a given level k, the Lagrange multiplier associated with (4) is denoted by $\underline{\lambda}_k$. The multiplier associated to (3) is simply w_k. Then, $\underline{\lambda}_k$ can be computed using the hierarchical inverse by (see [Escande et al., 2012], Sec. 3.5):

$$\underline{\lambda}_k = -\underline{A}_{k-1}^{\ddagger T} A_k^T w_k^* \qquad (11)$$

Once more, a cheaper reformulation can be obtained using the HCOD and is summarized in Alg. 2.

All the multipliers can be summarized under a pseudo matrix structure:

$$\Lambda_p = \begin{bmatrix} w_1^* & {}^1\underline{\lambda}_2 & \cdots & {}^1\underline{\lambda}_{p-1} & {}^1\underline{\lambda}_p \\ & w_2^* & \cdots & {}^2\underline{\lambda}_{p-1} & {}^2\underline{\lambda}_p \\ & & \ddots & \vdots & \vdots \\ & & & w_{p-1}^* & {}^{p-1}\underline{\lambda}_p \\ & & & & w_p^* \end{bmatrix} \qquad (12)$$

where ${}^j\underline{\lambda}_k$ ($j < k$) denotes the components of the multipliers $\underline{\lambda}_k$ of the level k for the constraints of level j, and the empty spaces for $j > k$ express the absence of multipliers on above levels. On the contrary to the primal variable x_p^*, there is no direct formulation to compute the whole dual variable Λ_p. Consequently, the multipliers of each level have to be computed iteratively.

2.2 Hierarchical active search

The inequality hierarchical program (iHQP by opposition to the eHQP) defined in (1) is solved using an active-search algorithm: if we know the set of inequality constraints that hold as equalities at the optimum, then the resolution of the iHQP simply involves the resolution of the eHQP composed of the active set of constraints. The active-search algorithm iteratively searches the optimal active set. It starts with an arbitrary initial guess (an empty set, if no better guess

[2] All the algorithms being given in the first part of the paper, they will be simply referenced by their number.

is available). At each iteration, it computes the optima of the primal and dual eHQP associated to the current active set, and deduces one constraint to be added or removed to/from the active set.

In the first part of the paper ([Escande et al., 2012], Sec. 4.3), we proposed in Alg. 5 an adaptation of the classical active-search algorithm that solves the whole hierarchical problem inside a unified loop by searching for the active set of all the levels of the hierarchy at the same time.

3 Decomposition computation and modification

In this section, we present a complete solution to compute the HCOD of a matrix $\underline{A}$. This solution is based on iterative orthogonal transformations, the Householder reflections and the Givens rotations (recalled in App. B). The algorithm presented in the previous sections uses the HCOD to compute the primal x and dual λ optima at each active-search iteration.

However, the decomposition is costly and should be avoided if possible. It has to be computed once for the initial active set at the first iteration. At the next iterations when one element of the active set is added or removed, the initial HCOD only needs to be updated to fit with the current active set. While the computation of the HCOD is cubic in the matrix size, the update is quadratic and keeps a low computation cost for the active search even when several iterations are needed. A complete solution to apply such an update is presented in the two last subsections.

3.1 Computation of the HCOD

3.1.1 Right basis Y

The HCOD is computed iteratively for each level k. We suppose that the expression of A_k in the basis $\begin{bmatrix}\underline{Y}_{k-1} & Z_{k-1}\end{bmatrix}$ is computed. A rank-revealing LQ decomposition of $A_k Z_{k-1}$ is first performed: the row of largest norm of $A_k Z_{k-1}$ is selected and permuted at the top of the matrix by a permutation matrix Π_1. The Householder reflection Q_1 nullifying the tail of this row is built using (47) and applied to the $m_k - 1$ last rows of $A_k Z_{k-1}$ using (46). The result of this first reflection can be written:

$$A_k Z_{k-1} = \Pi_1 \begin{bmatrix} l_1 & 0 & \dots & 0 \\ \times & & & \\ \vdots & & K_1 & \\ \times & & & \end{bmatrix} Q_1^T \qquad (13)$$

The reflection Q_1 is also applied to the levels A_{k+1} to A_p. The same process is iteratively applied to the remaining matrix K_1, until all the rows of the remaining matrix are null. The result can be written:

$$A_k Z_{k-1} = \Pi \begin{bmatrix} & & 0 & \dots & 0 \\ & L^R & \vdots & & \vdots \\ & & & & \\ L^N & & & & \\ & & 0 & \dots & 0 \end{bmatrix} Q^T \qquad (14)$$

with $\Pi = \prod_{i=1}^{r_k} \Pi_i$, $Q = \prod_{i=1}^{r_k} Q_i$ and r_k the rank of $A_k Z_{k-1}$ (i.e. the size of L^R). This decomposition of $A_k Z_{k-1}$ is available in many classical linear algebra software by applying a column-pivoting QR decomposition on the transpose (e.g. QR in matlab, xGEQPF in LAPACK, ColPivHouseholderQr in Eigen). Q is a transformation of Z_{k-1} that reveals the rank of level k and gives the component Y_k of the right basis:

$$\begin{bmatrix} Y_k & Z_k \end{bmatrix} = Z_{k-1} Q \qquad (15)$$

The matrix Q is not expressed explicitly but as a structured product of r_k Householder reflections Q_i, whose vectors v_i are of decreasing support size. Finally, the obtained basis $Y = \begin{bmatrix}\underline{Y}_p & Z_p\end{bmatrix}$ is expressed as a structured matrix of $\underline{r}_p$ Householder reflections. Each reflection i corresponds to a vector v whose support size is $n - i$ and does not modify the i first columns of $\underline{A}_p$. If needed, the coefficients of the matrix Y can be explicitly computed by applying the reflections on the identity matrix I_n.

3.1.2 Left bases W_k

For each stage k, the matrix L^N has now to be nullified by applying a transformation on the left side of the decomposition. We can suppose that L^N has

only one row (the same procedure is then applied for each rows of L^N). Each component i of the row L^N starting from the last one is nullified by applying a Givens rotation pivoting around the row i of L^R. The result of the i^{th} rotation can be written:

$$\Pi\ W^{(i)} \begin{bmatrix} \times & & & & 0 & 0 \\ \vdots & \ddots & & & & \\ \times & \cdots & l_{ri} & & \vdots & \vdots \\ \vdots & & & \ddots & & \\ \times & \cdots & \times & \cdots & \times & \\ \boxed{\times} & \boxed{\cdots} & \boxed{l_{ni}} & \boxed{0} & \boxed{\cdots} & 0 \cdots 0 \end{bmatrix} Q^T$$

where the double arrow represent the next Givens rotation to be applied. The matrix W_k is built recursively using the following series of Givens rotations:

$$W^{(i)} = W^{(i-1)} G[r_k - i, r_k + 1, \theta_i] \tag{16}$$

where θ_i is chosen to nullify l_{ni}, *i.e.* the i^{th} column starting from the right of the matrix ${W^{(i-1)}}^T \begin{bmatrix} L^R \\ L^N \end{bmatrix}$. The recursion starts with $W^{(0)} = \Pi$ and stops with $W_k = W^{(r_k)}$.

The left part $A_k \underline{Y}_k$ has been already computed when applying the Householder reflections Q_i on A_k. The matrices M_k and N_k are computed by applying the r_k Givens rotations to $A_k \underline{Y}_k$. Finally, the rows of the matrix corresponding to L^R and L^N can be exchanged by applying a last permutation. The decomposition at level k is then written:

$$A_k = W_k \begin{bmatrix} N_k & 0 & 0 \\ M_k & L_k & 0 \end{bmatrix} Y \tag{17}$$

with W_k a product of Givens rotations and permutations and Y a product of Householder reflections.

3.2 Decomposition update

We now present the HCOD update, i.e. how to add one constraint at level k. The next section will present the removal of a constraint.

3.2.1 Update properties

We consider the HCOD $\underline{W}_p$,$\underline{H}_p$,Y of the matrix $\underline{A}_p$. One row A_{up} is added in the level k. The first part of the update consists in finding a proper decomposition of this matrix for level k, mainly by applying a series of Givens rotations Y_{up} on the right. The second part consists in propagating Y_{up} on the levels $j > k$.

The stack of A_k and A_{up} can be written:

$$\begin{bmatrix} A_k \\ A_{up} \end{bmatrix} = \begin{bmatrix} W_k & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \boxed{N_k} & 0 & 0 \cdots 0 & \cdots & 0 \\ \boxed{M_k} & \searrow L_k \swarrow & & & \\ \underline{\quad} & A_{up}Y & \underline{\quad\quad} & 0 \cdots & 0 \end{bmatrix} Y^T \tag{18}$$

The update procedure depends on the number of 0 of the tail of the new row.

The insertion of A_{up} in A_k can increase the rank of the HCOD (*i.e.* $\underline{r}_p$) of 1 or let the rank unchanged. Similarly, the rank of level k can increase of 1 or stay unchanged. Finally, the ranks of the upper levels can only decrease or stay unchanged since they are not augmented by any rows. In conclusion, at most one matrix from $k+1$ to p can lose at most one rank, all the other rank being kept unchanged. The level that loses one rank is denoted $\hat{k}$. If the updated matrix k does not change its rank, all the following rank will stay constant too. This case can be seen as $\hat{k} = k$.

The rank loss means that A_{up} is linked with $\underline{A}_{\hat{k}}$ (*i.e.* A_{up} is a linear combination of the rows of $\underline{A}_{\hat{k}}$). Consequently, $A_{up}Y^T$ is null on the columns corresponding to levels $\hat{k}$ to p and its support equal to $A_{up}\underline{Y}_{\hat{k}}$. The row of level $\hat{k}$ that has the same number of nonzero elements (abusively named the *rank* of the decomposition row) as $A_{up}Y$ is denoted by $\hat{r}$.

3.2.2 Case 1. A_{up} is linked with $\underline{A}_k$

The simplest case to update is if there is at least as much 0 in the tail of the new row than on the last row of H_k:

$$\begin{bmatrix} A_k \\ A_{up} \end{bmatrix} = \begin{bmatrix} W_k & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \boxed{N_k} & 0 & 0 & 0 \\ \boxed{M_k} & \searrow L_k \swarrow & 0 & 0 \\ \underline{\quad} & A_{up}Y \ \underline{\quad} & 0 & 0 \end{bmatrix} Y^T \tag{19}$$

This means that the new row is linked with $\underline{A}_k$. The decomposition is completed by nullifying the part of $A_{up}\underline{Y}_{\hat{k}}$ below L_k using a left basis W_{up} as explained in Section 3.1.2. As a result, the decomposition of level k can be written:

$$\begin{bmatrix} A_k \\ A_{up} \end{bmatrix} = \begin{bmatrix} W_k & 0 \\ 0 & 1 \end{bmatrix} W_{up} \begin{bmatrix} N_k & 0 & 0 & 0 \\ M_k & L_k & 0 & 0 \\ N_{up} & 0 & 0 & 0 \end{bmatrix} Y^T \tag{20}$$

with $N_{up} = W_{up}^T A_{up} Y$ and W_{up} computed with (16). The update is completed for level k by reordering the rows of H_k and the corresponding columns of W_k. Since there is no effect on the level $j > k$, the update of the whole HCOD is completed.

3.2.3 Case 2: A_{up} is not linked with $\underline{A}_k$

The first step of the update is to turn $A_{up}Y$ into a new row of H_k. This can be done directly by applying a set of Givens rotations on the right to nullify the elements starting from the last nonzero one using the previous element as a pivot.

$$Y_{up}^{(i)} = Y_{up}^{(i-1)} G[\hat{r} - i - 1, \hat{r} - i, \theta_i] \tag{21}$$

where θ_i is chosen to nullify the corresponding element of $A_{up} Y Y_{up}^{(i-1)}$. The recursion starts with $Y_{up}^{(i)} = I$ and stops with $Y_{up} = Y_{up}^{(\hat{r} - \underline{r}_k)}$. The updated matrix at level k can then be written:

$$\begin{bmatrix} A_k \\ A_{up} \end{bmatrix} = \begin{bmatrix} W_k & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} N_k & 0 & 0 & 0 \\ M_k & L_k & 0 & 0 \\ A_{up}Y & & X & 0 \end{bmatrix} Y_{up}^T Y^T \tag{22}$$

where X is the *edge* of the new triangle formed by L_k and the added line. The left part of the new row (*i.e.* $A_{up}\underline{Y}_{\hat{k}-1}$) is not modified by Y_{up}. X is nonzero (otherwise, it corresponds to the case 1). This decomposition then gives the update for level k (*i.e.* of W_k, M_k, N_k and L_k). The update can now be propagated to the next levels.

3.2.4 Propagation of Y_{up}

The propagation of Y_{up} and the consequent modifications of the levels $k+1$ to p are applied iteratively from level $k+1$. For each row with a 0 tail $h = [\times \cdots \times \ 0 \ \ 0 \cdots 0\ ]$, the multiplication with Y_{up} keeps all the 0 of the tail, except the first one, due to (21):

$$hY_{up} = [\times \cdots \times X' \ 0 \cdots 0\ ] \tag{23}$$

where X' is at the position of the first 0 of the tail of h. X' is nonzero in the general case but is zero if the rank of h is greater than $\hat{r}$, once more due to (21). Consequently, the shape of the decomposition at level i after application of Y_{up} is:

$$A_i = W_i \begin{bmatrix} N_i & \nu_1 & 0 & 0 \\ & \vdots & 0 & 0 \\ & \nu_{m-r} & 0 & 0 \\ M_i & L_i \ (d_1, \ldots, d_r) & \vdots & \vdots \\ & & 0 & 0 \end{bmatrix} Y_{up}^T Y^T$$

with $(d_1, ..., d_r)$ a set of r_i component forming a upper bidiagonal on L_i and $(\nu_1, ..., \nu_{m-r})$ a column vector of $m_i - r_i$ components. For all lines j with a rank greater or equal to $\hat{r}$, d_j is zero.

It is possible to distinguish three cases, whether the rank loss happens before, after or at the current level i.

3.2.5 Case 2.1: rank loss after the level i

In the first case, the HCOD for levels $i-1$ and i looks like:

$$\begin{bmatrix} N_{i-1} & 0 & & \\ M_{i-1} & L_{i-1} & 0 & 0 \\ & \nu_1 & 0 & 0 \\ N_i & \vdots & 0 & 0 \\ & \nu_{m-r} & 0 & 0 \\ M_i & L_i \ (d_1, \ldots, d_r) & \vdots & \vdots \\ & & 0 & 0 \end{bmatrix} \tag{24}$$

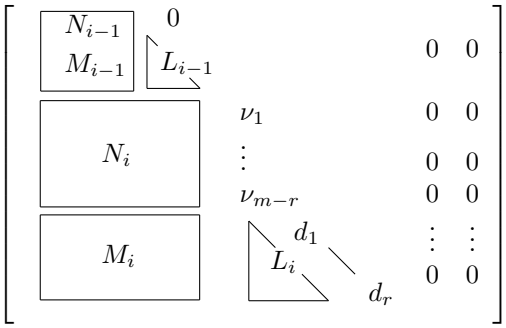
with all the d_j nonzero. From this structure, the update of the decomposition of level i is directly obtained.

3.2.6 Case 2.2: rank loss at the current level i

The structure of the decomposition is similar to the previous one, but there is l zero components in the begining of d_j: the row l of H_i has the same support size as $A_{up}Y$. By reordering L_i to put the row l at the bottom row and reordering accordingly W_i, the decomposition can be put under the form (19). Following, the last row can be nullified by the set of Givens rotations W_{up}, as described in Sec. 3.2.2.

3.2.7 Case 2.3: rank loss before level i

In that case, the total rank of the HCOD at the previous level $\underline{r}_{i-1}$ did not change, and the levels $i-1$ and i have the following structure:



(25)

However, since the rank loss occurred in one of the previous levels, the Y_{up} does not modify the columns corresponding to L_i. Following, the ν_j and d_j are zero, and the given decomposition is directly the update of the current level.

3.3 Decomposition downdate

When one constraint is deactivated, the corresponding row of $\underline{A}_p$ is removed. The modification of the HCOD to correspond to the new $\underline{A}_p$ is called *downdate* (by opposition to the update).

3.3.1 Removing one row of H_k

Without loss of generality, we can consider the case where the last row of the level k should be removed (a permutation is first applied otherwise). The general idea is to find a rotation W_{dw} so that W_kW_{dw} is unitary on the row to be removed:

$$W_kW_{dw} = \begin{bmatrix} & & & 0 \\ & W_A & & \vdots \\ & & & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix} \tag{26}$$

The rotation W_{dw} can be built as a product of Givens rotations:

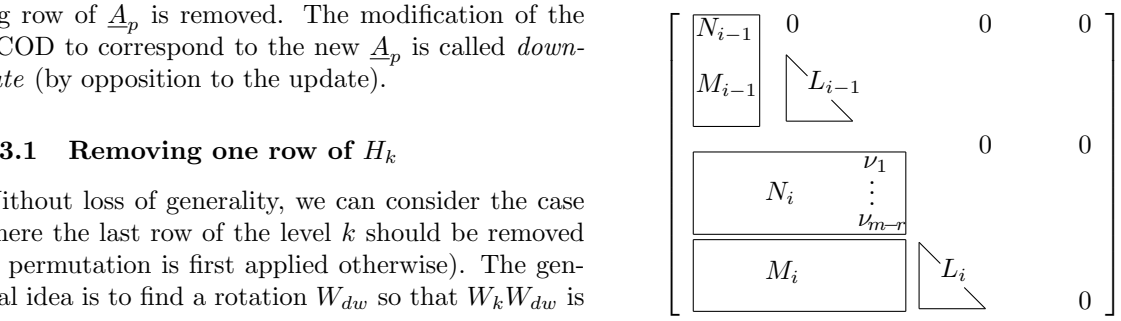
$$W_{dw} = \prod_{i=1}^{m_k-1} G[i+1, i, \theta_i] \tag{27}$$

where θ_i is chosen to nullify the i^{th} component of the row of W to be removed. The result on H_k can be written:

The last row of H_k and the last row and column of W_k are then removed. If all the d_1 ... d_r are nonzero, then the obtained form is a proper HCOD. Otherwise, the resulting L_k is upper Hessenberg and can be corrected by using a serie Y_{dw} of Givens rotation applied on the right side, as in the update case.

3.3.2 Propagation of Y_{dw}

We now consider the level $i > k$. After application of Y_{dw}, the HCOD of levels $i-1$ and i looks like:



(28)

• *Case 2.1:* If all the $\nu_1 \dots \nu_{m-r}$ are null, then the triangle part of H_i is upper Hessenberg. The process already explained for H_k is applied on H_i and the obtained Y_{dw} is propagated to H_{i+1}.

• *Case 2.2:* Otherwise, we can suppose without loss of generality that ν_{m-r} is nonzero (the corresponding row is linked to the row removed from level k). The rank of level i is increased by one. The decomposition is finalized by simply nullifying the nonzero $\nu_1 \dots \nu_{m-r-1}$ by applying Givens rotations on the left of H_i.

• *Case 2.3:* after Case 2.2, the propagation of Y_{dw} on the levels $i+1$ and further will not modify the corresponding L matrix, which concludes the HCOD modification.

A complete implementation of the decomposition and the updates, along with HQP resolutions, is proposed in App. D.

3.4 Structured versus explicit matrix

The cost of a single elementary transformation is much lower when it is kept as a structured matrix. Then, the explicit computation Y is not needed. However, when n Householder reflections are involved, the cost of a multiplication by Y is the same for an explicit or a structured expression. This is typically the case after a full HCOD, since the problem is often fully constrained and thus involves n reflections. Then, after any update or downdate, Y also implies a set of Givens rotations. To avoid an increase of the cost of Y along the active-set iterations, we transform in practice Y to an explicit matrix at the end of the first HCOD.

Similarly, the W_k are given as a set of Givens rotation. However, some coefficients of W_k need to be explicitly computed during a downdate (in (26)). For simplicity, we also compute the explicit form of W_k, except when the level k is full rank (W_k is then the identity).

4 Implementation considerations

In this section, we give some results and techniques about the effective implementation of the solver. We first characterize the computation cost of both the eHQP and iHQP solvers. When the solver is used to compute the robot control, one instance of the problem is solved at each control cycle. In that case, we prove under some conditions the continuity of the control law when some constraints become active. This continuity can be used to reduce the computation cost. Finally, we prove the overall stability of the control scheme when the solver is used to execute an inverse kinematics.

4.1 Complexity

We roughly characterize the cost of the HCOD decomposition, of the eHQP resolution knowing the HCOD and of the whole iHQP algorithm.

4.1.1 Cost of a HCOD

We first consider a matrix A of size $m \times n$. The LQ decomposition of $A = LQ$ requires $\sum_{i=1}^{\min(m,n)}(n-i) \approx nm$ Givens rotations, each of them being applied on A and Q. The cost to obtain L is then nm^2 elementary operations, while the cost for Q is mn^2.

A COD $\begin{bmatrix} V & U \end{bmatrix} \begin{bmatrix} 0 & 0 \\ L & 0 \end{bmatrix} \begin{bmatrix} Y & Z \end{bmatrix}^T$ starts with a LQ before applying a QL on the obtained AY. The cost of the QL is $(m-r)r^2$ to obtain L and $(m-r)m^2$ to obtain $W = \begin{bmatrix} V & U \end{bmatrix}$. The total cost of the COD is roughly $n^2m + nm^2 + (m-r)m^2$. When m is much smaller than n (typically for A_1), the cost is in $O(mn^2)$ while when m and n have the same magnitude order (typically for $\underline{A}_p$), the cost is in $O(2n^3)$.

We now consider the set of p matrices A_k, each of them being of size $m_k \times n$, and of rank in the hierarchy $r_k = \underline{r}_k - \underline{r}_{k-1}$. The size of $\underline{A}_p$ is $m = \sum m_k$. For simplicity, we consider that all the decompositions are computed using a series of Givens rotations, and that all the basis are computed explicitly into a dense format.

The HCOD can be computed by a serie of COD. For each COD, the coefficients of the intermediary bases Z_k are not necessary: only the $A_{k+1}Z_k$ are needed. The Y_k and Z_k are kept as product of elementary transformations. The number of operations to get the right part of the decomposition Y is the same as for getting the right part of decomposition of the COD. The QL decompositions W_k are then made for each level k independently, and the cost is each time the same as for the QL of the COD of A_k. The total cost is then:

$$c_{\text{HCOD}} = n^2 m + nm^2 + \sum_{k=1}^{p} (m_k - r_k) m_k^2 \qquad (29)$$

The cost is less than the COD of $\underline{A}_p$ but more than its QR. Of course, if all the matrices are full-row rank, the three costs are the same.

4.1.2 Cost of the eHQP

We consider now the two algorithms 1 and 2. Since $\underline{w}_p^*$ can be computed in both, we remove the redundant computations from Alg. 1.

The computation involved by the primal optimum amounts to the forward substitution of the stacked $U_k b_k$ by the triangular matrix formed of the M_k and L_k. The cost is $\sum m_k r_k$ for computing the $U_k b_k$ and then $\underline{r}_p^2$ for the forward substitution. The total cost is roughly $\underline{r}_p^2$.

The computation of the multipliers of level k is similar: it is the backward substitution of the $\underline{\rho}$, whose own cost is negligible with respect to the cost of the substitution. The cost is then $\underline{r}_k^2$. The cost to compute Λ_p is then $\sum \underline{r}_k^2 \approx \underline{r}_p^3$ which is similar to the cost of the whole HCOD.

In conclusion, if the proposed method is used to compute only the primal optimum of a eHQP, the cost is similar to the cost of the inversion of a full-rank system of the same size using a QR decomposition and a forward substitution.

If both primal and dual optima are needed (for example in the active search), the computation of the multipliers Λ_p is costly and should be parsimoniously realized.

4.1.3 Cost of the HQP active search

Two supplementary operations are executed during the active search: the computation of the step length and the HCOD update.

Consider first the update. A new row is added at level k, with $\hat{r}$ nonzero elements in $A_{up}\underline{Y}_p$. There is then $\hat{r} - \underline{r}_k$ Givens rotation to execute and propagate, first on the least-priority levels $H_{k+1} \dots H_p$ (for a cost of $(\hat{r} - \underline{r}_k) \sum_{j=k+1}^{p} m_j$) and, second on the basis Y (for a cost of $(\hat{r} - \underline{r}_k)n$). If $\hat{r} < \underline{r}_p$, there is also one level j that becomes rank-deficient and for which an update of W_j is needed, for a cost of $2r_j^2$. The total cost is then in $O(n^2 + mn)$.

Using the same reasoning, the cost of a downdate is also $O(n^2 + mn)$.

Finally, denoting by $s_k \geq m_k$ the total number of constraints of level k (*i.e.* active and inactive), the step length computation for each level requires the multiplication of the rows of A_k for the $s_k - m_k$ inactive constraints. The cost is $\sum_k n(s_k - m_k) = n(s - m)$ with $s = \sum_k s_k$.

The cost of the active search depends of course on the number of iterations in the inside loop (the number of iterations of the outside loop being bounded). If during an iteration a constraint is activated, the iteration implies the computation of the primal, of the step and an update. If the iteration is concluded with a deactivation, it implies the primal, the step, the dual and the update. Finally, due to the outer loop of Alg. 5 there are p iterations that do not activate nor deactivate any constraints. We denote by N_U and N_D the number of activations and deactivations of the algorithm. The total cost is easily deduced from the above list and Table 1.

If we suppose additionally that the hierarchical problem constraints the whole parameter space ($m = n$) and if we smooth the differences between the level for the dual computation and the updates (approximate costs are given on the third column of Table 1), the total cost is:

$$c_{\text{asearch}} = 2n^3 + (3N_U + 4N_D + 2p)n^2 + (N_U + N_D + p)n(s - n)$$

In particular, the minimum cost of the active search

operation	cost	approx.
QR	$nm^2 + n^2m$	$2n^3$
COD	$n^2m + nm^2 + (m-r)m^2$	$2n^3$
HCOD	$n^2m+nm^2+\sum(m_k-r_k)m_k^2$	$2n^3$
Primal x_p^*	$\underline{r}_p^2$	n^2
Dual $w_k^*, \underline{\lambda}_k$	$\underline{r}_k^2$	n^2
Dual Λ_p	$\sum_{k=1}^{p} \underline{r}_k^2$	pn^2
Update	$n^2 + mn$	$2n^2$
Step τ	$n(s-m)$	$n(s-n)$

Table 1: *Computation cost for each operation of the active search. The approximations are given for $n = m$ and $k = p$*

is obtained for $N_U = N_D = 0$:

$$c_{\text{asearch}} = 2n^3 + np(n+s)$$

In that last cost, compared to a standard QP, the hierarchy brings the p factor in the last term. This is mainly due to the bidimensional structure of the multipliers Λ_p.

Without surprise, the cost of the active-search strongly depends on the number of activations and deactivations $N_U + N_D$ that are needed to reach the optimal active set. If a good approximation of the optimal active set is known, then the total cost of the active search can be significantly decreased. In particular, if a similar problem has already been solved, then the obtained optimal active set can be reused for the current problem. This reuse of a part of the solution of a slightly different problem is called *warm start* and is detailed in the following three sections. We first define this notion of *similar* problem.

4.2 Parametric optimization

We now consider that HQP, that varies continuously with respect to a given parameter t:

$$\underset{x,w_1 \dots w_p}{\operatorname{lex\,min}} \ \{\|w_1\|, \|w_2\|, \dots, \|w_p\|\}. \qquad (30)$$

$$\text{subject to} \quad \forall k = 1:p, \ A_k(t)x \leq b_k(t) + w_j$$

with $A_k(t)$ and $b_k(t)$ continuous function of a real parameter $t \in \mathbb{R}$. This problem is denoted by HQP(t).

We study the continuity of the two functions:

$$\mathcal{O} : t \to \mathcal{O}(t) = \{x, \forall x', \underline{A}_p x \preccurlyeq \underline{A}_p x'\} \qquad (31)$$

$$x^* : t \to x^*(t) = \min_{x \in \mathcal{O}(t)} \{\|x\|\} \qquad (32)$$

where $b_1 \preccurlyeq b_2$ denotes the lexicographic order corresponding to the hierarchy. $\mathcal{O}(t)$ is the optimal set of the HQP(t) and $x^*(t)$ is the least-square element of $\mathcal{O}(t)$ (unique since $\mathcal{O}(t)$ is convex).

4.3 Continuity of the parametric optimum

First, the eHQP optimum is known to be continuous outside of the singular regions. This is formalized by the following result. The functions giving the rank of the projected matrices are denoted by:

$$r_k : t \to r_k(t) = \operatorname{rank}(\underline{A}_k(t)) - \operatorname{rank}(\underline{A}_{k-1}(t)) \qquad (33)$$

Theorem 4.1. *Consider a eHQP of the form* (30)*. For a given t_0, if the maps $r_1 ... r_p$ are constant on a neighborhood of t_0, then $\mathcal{O}(t)$ and $x^*(t)$ are continuous at t_0.*

Proof. The result is straightforward using the fact that the pseudo-inverse map $A \to A^+$ is continuous inside the set of constant rank of A [Ben-Israel and Greville, 2003]. □

When the $r_k(t)$ are not constant, the solution maps are not continuous in general. The only case where the continuity is not ensured is when passing inside a singular region and only at the region border. Inside the singularity, the optimum remains continuous.

Consider now the full problem (30) with inequalities. It is possible to show that, apart from these discontinuities due to the singularity of the eHQP, the continuity of the optimum is ensured. $\mathcal{S}^*(t)$ denotes the optimal active set of HQP(t) corresponding to $x^*(t)$. For a given active set $\mathcal{S}$, the eHQP associated to (30) is denoted by eHQP($t, \mathcal{S}$) and the optimum of this eHQP is denoted by $x^*_{\mathcal{S}}(t)$. We then have the following result:

Theorem 4.2. *At a given t_0, if $t \to x^*_{\mathcal{S}^*_{t_0}}(t)$ is continuous, then $x^*(t)$ is continuous.*

Proof. If the optimal active set is constant in a neighborhood of t_0, the result is straightforward using Th. 4.1. We then consider the case where the active set changes on t_0: $\forall t > t_0, \mathcal{S}(t) \neq \mathcal{S}(t_0)$. We suppose that the active set of level k is increased by one constraint $A_{up}x \leq b_{up}$ in $t > t_0$ (the case with multiple activations or deactivations follows naturally). Since (A_{up}, b_{up}) is active for any $t > t_0$ of a neighborhood, it means the optimum lies on this constraint at t_0: $A_{up}x^*_p(t_0) = b_{up}$. Both active sets $\mathcal{S}(t_0)$ and $\mathcal{S}_+ = \mathcal{S}(t_0) \bigcup \{(A_{up}, b_{up})\}$ give the same x^*_p. Using the continuity of the eHQP associated to $\mathcal{S}_+$ the continuity of $x^*(t)$ toward $x^*(t_0)$ is straightforward. □

The continuity of the solution before and after the update can be understood by looking at the structure of $\underline{y}^*_p$ in the four cases of Sec.3.2: the HCOD update may keep r_k constant or increase it, and in the second case, may decrease the rank of level $j > k$ or increase the total rank $\underline{r}_p$.

In the first case, only the continuity of y^*_k is questionable. It is ensured by the continuity of the partitioned generalized inverse [Ben-Israel and Greville, 2003]:

$$\begin{bmatrix} A \\ C \end{bmatrix} \begin{bmatrix} b \\ d \end{bmatrix}_{\underset{d \to CA^+b}{\to}} A^+b \tag{34}$$

for any A, b, C. The continuity of the y^*_j, $j > k$ follows, using recursively the continuity of $\underline{y}^*_{j-1}$ and the continuity of the eHQP.

If both r_k and $\underline{r}_p$ increase, the update adds a new column Y_{up} in Y_k. This vector belongs to the null space of the HCOD at t_0: $Y^T_{up}\underline{Y}_p(t_0) = 0$. This case is then equivalent to considering that (A_{up}, b_{up}) was added as a new least-priority level $p+1$ of the hierarchy. The $\underline{y}^*_p$ is continuous using Th. 4.1. And y^*_{p+1} is continuously evolving from 0 using (34).

Finally, the key case is when the rank increase at level k and causes a rank loss at level $j > k$. It means that Y_{up} belongs to $Y_j(t_0)$. In that case, the DOF corresponding to Y_{up} was allocated to the level j before t_0. At t_0, the constraint $A_{up}x \leq b_{up}$ is reached. The DOF is then reallocated to the level k. The reallocation is performed when the optimum is exactly on the constraint, thus causing no discontinuity. This case is depicted in Fig. 2.

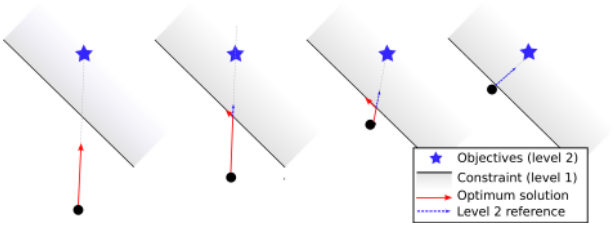


Fig. 2: *Continuity and stability: when the system approaches a boundary, the optimum continuously changes to a pure motion along the boundary, until it finally stably stops on the limit. The optimum does not fight against the reference required by the equality constraint of level 2.*

4.4 Warm start and real-time implementation

On-board a robot, the solver is used at discretized times t,t+1,... Thanks to the continuity, the optimum of time t nearly satisfies the feasible constraints of time $t+1$. A good initial guess for the active set of time $t+1$ is thus the optimal active set of time t. In most of the cases, this is enough to reach the optimum in only one iteration. From time to time, the active set changes: in most cases few additional activations and deactivations are sufficient. However, one cannot guarantee the number of necessary iterations to reach the optimum. In particular, a minor modification of one optimum can trigger a cascade of activations-deactivations in pathological cases.

In these pathological cases, it is possible to force the solver to stop after an arbitrary number N of iterations. It is then not ensured that the optimum is reached and three sub-optimal cases arise depending on what happened during the last iteration: if τ was less than 1 (case of Alg. 4#17), it means that all the feasible constraints are satisfied, but the optimum of the corresponding eHQP is not reached yet. If an update was triggered with $\tau = 1$ (case of Alg. 4#11), it means that some of the constraints might still be unsatisfied and inactive. In that case, the residue of these constraints could be improved by activating them using some more iterations. Lastly, if a downdate was triggered, the eHQP optimum is reached but not the optimal active set. With more iterations,

some unnecessary constraints could be relaxed, that would free some DOF to improve the solution.

Thanks to the continuity property, it is possible to show that the obtained final point $x^{(N)}$ is at a distance to the optimum x^* proportional to the sampling of the control. If the optimal active set was not reached at $t+1$, the search will restart at $t+2$ from the intermediary active set obtained after the N iterations of time $t+1$.

By using the previous active set as a warm start and bounding the number of iterations, the solver will find the optimum in real time in most of the case, and otherwise will respect the real-time constraint by spreading the search of the optimal active set over several iterations.

4.5 Stability

We consider now the practical case of the inverse kinematics: the robot velocity $\dot{q}$ is computed as the optimal solution of a HQP, where the constraints are defined by a set of p tasks:

$$\dot{e}_k = J_k \dot{q} \tag{35}$$

$$\forall k \in S_I,\ \dot{e}_k \leq b_k \tag{36}$$

$$\forall k \in S_E,\ \dot{e}_k = \dot{e}_k^* \tag{37}$$

where $S_I \bigcup S_E$ is a partition of the set $\{1 \dots p\}$ of the p first integers: S_I are the task levels that are defined by a limit b_k and S_E are the task constraints to follow a given velocity $\dot{e}_k^*$. When the task function e_k is an error between a reference and a current sensor value, the task reference $\dot{e}_k^*$ is often given to drive the error to 0:

$$\dot{e}_k^* = -\kappa_k e_k \tag{38}$$

where $\kappa_k > 0$ is a gain used to tune the convergence speed. Finally, we suppose that 0 is an acceptable solution for all the tasks of S_I, *i.e.* $\forall k \in S_i,\ b_k > 0$. The robot input $\dot{q}$ is then computed as the result of a HQP:

$$\underset{\dot{q}, w_1 \dots w_p}{\operatorname{lex\,min}} \ \{\|w_1\|, \|w_2\|, \dots, \|w_p\|\}. \tag{39}$$

subject to (35),(36), (37). In this section, we briefly prove that the use of a HQP keeps the same properties of control stability than in other classical inverse-kinematics approaches. We define by e_E the stack of all equality constraints:

$$e_E = \begin{bmatrix} e_{k_1} \\ \vdots \\ e_{k_E} \end{bmatrix} \tag{40}$$

for $S_E = \{k_1 \dots k_E\}$, and by J_E the associated jacobian.

Theorem 4.3. *The control law defined by* (39) *is stable in the sense of Lyapunov. It is asymptotically stable* iff *J_E is full row rank and none of the equality-constraint levels of the HCOD are rank deficient.*

The idea of this theorem is illustrated in Fig. 2: if the optimum is inside the bounds (36), then the close-loop inverse-kinematics eHQP is known to be stable [Antonelli, 2009]. Otherwise, the system will smoothly stop on the boundary. The inequality prevents the robot to come closer to the objective but does not imply an unstable behavior. The stability is kept even if the boundaries prevent the system to be asymptotically stable. The complete proof is given in App. C.

If the robot is not inside the confidence area (then some b_k are not positive), then the corresponding task components can be seen as equalities that bring the robot inside the area, which ensures the stability during the transition phase.

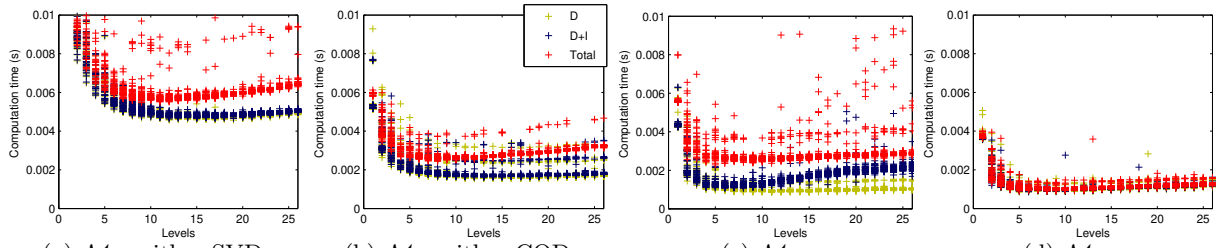
5 Experiments

We first present in Sec. 5.1 the computation times obtained with both eHQP and iHQP out of any robotics application. Then the iHQP solver is used in the following sections to perform an inverse-kinematics control scheme on a humanoid robot.

5.1 Computation time

5.1.1 Equality-only HQP

We first experimentally check the computation time needed to compute the primal optimum of a eHQP using the three following methods:



(a) $\mathcal{M}_P$ with a SVD (b) $\mathcal{M}_P$ with a COD (c) $\mathcal{M}_Z$ (d) $\mathcal{M}_H$

Fig. 3: *Comparison of the computation time of the eHQP primal optimum using four methods. (a) Using $\mathcal{M}_P$ with a SVD (b) Using $\mathcal{M}_P$ with a COD (c) Using $\mathcal{M}_Z$ with a COD (d) Using $\mathcal{M}_H$. Various problems $\underline{A}$ with $n = 100$, $m = \sum m_i = 120$, $r = \sum r_i = 80$ are resolved using varying number of levels p on a 2.5GHz PC processor. The computation time is plotted wrt p. The plots display the time measured for the decomposition of AP and AZ only (D), for the decomposition and the inversion (D+I), and for decomposition, inversion and projection (total cost). In (a) and (b) (D) is nearly equal to (D+I).In (d), the projection consists in translating y^* in the canonical basis, which is plotted but is negligible.*

- $\mathcal{M}_P$: using iterative pseudoinversion and explicit projector P_k, as proposed in [Siciliano and Slotine, 1991] (recalled in Eq. (15) of the first part)
- $\mathcal{M}_Z$: using iterative pseudoinversion and the basis of the null space Z_k (recalled in Eq. (17) of the first part)
- $\mathcal{M}_H$: using the hierarchical inverse (9).

Since the pseudoinverses and projectors of $\mathcal{M}_P$ are generally computed using a SVD [Baerlocher and Boulic, 2004] which is well known for being slower than a COD, the same $\mathcal{M}_P$ is computed twice, once with a SVD and the second time with a COD. A set of eHQP problems is resolved. These solvers are run on a set of problems whose size parameters (number of columns, of rows and total rank) are the same but whose number p of levels of the hierarchy varies. The computation costs are presented in Fig. 3.

For all methods, the cost increases for small p. The hierarchy induces very little extra cost, even for large p, and no significant cost at all when using our method. As expected, the SVD is very expensive compared to the COD. Apart from the additional cost due to the SVD, the total cost is similar for $\mathcal{M}_P$ and $\mathcal{M}_Z$ but is differently distributed: $\mathcal{M}_P$ spends more time in the decomposition while $\mathcal{M}_Z$ spends more time in computing Z. Finally, by working with y^* directly and using only one basis Y for all the decomposition, $\mathcal{M}_H$ is the most efficient. Most of the time is spent in the decomposition. Inverting L and translating $\underline{y}_p^*$ into x^* are negligible. The method is approximately six times faster than the classical SVD-based $\mathcal{M}_P$ (and up to ten times for small p).

5.1.2 Active search

Similarly, the hierarchical active search Alg. 5 is compared to the cascade of QP used in [Kanoun et al., 2011]. The results are shown in Figures 4 and 5. The computation cost increases with the number p of levels when using a cascade of QP, while it remains constant with Alg. 5 as shown in Fig. 4. This is due to an increase of the number of iterations of the active-search loops used in the cascade, as shown in Fig. 5. Indeed, the cascade activates a set of constraints at level k that may not be necessary at level $k+1$ and activates it again at level $k+2$. These back-and-forth decisions are avoided when considering all the levels at the same time.

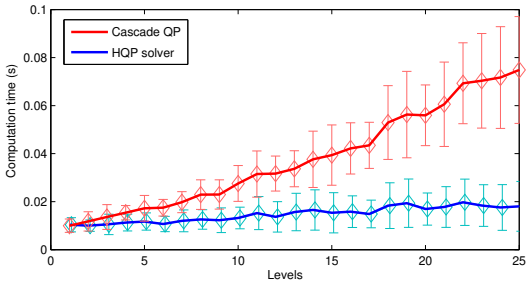


Fig. 4: *Computation cost of the iHQP resolution using a cascade of QP [Kanoun et al., 2011] or a our HQP solver (Alg. 5). A set of hierarchical problems is solved, whose size parameters are the same ($n = 100$, $m = 150$, $r = 80$) but whose number p of levels varies. The computation time is plotted wrt p. The time increases with the number of levels when using a cascade, while it is nearly constant with our solver. A little overhead appears for high p and is due to the computation of Λ_p. For $p = 1$, the cost are exactly the same.*

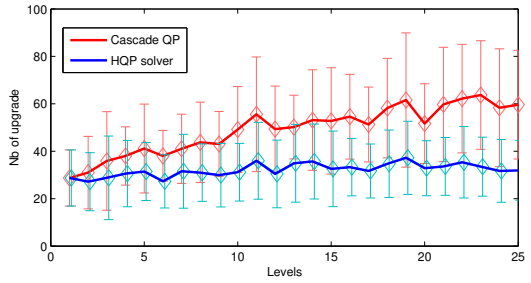


Fig. 5: *Number of activations and deactivations using a cascade of QP and a our solver for the same problems as in Fig. 4. The number increases with p for the cascade of QP and remains constant for the HQP solver.*

5.2 Robotic setup

The solver has been experimentally tested to perform an inverse kinematics with the HRP-2 robot. The robot has 36 DOF whose first six are not actuated. When considering only the kinematics, a classical solution is to replace the underactuation by a contact constraint of equivalent size, typically constraining the six parameters of position and orientation of one contacting foot. The underactuation is then resolved while the dynamic of the robot is negligible. All the QP resolved in the following have a parameter size $n = 36$. The robot is controlled at 200Hz.

Two sets of tasks are considered. The first set aims at regulating to 0 an error depending on the robot configuration. The task function is then given by:

$$e(q, \Omega) = s(q) - s^*(\Omega) \tag{41}$$

where the task function e depends on the robot configuration and other parameters of the universe Ω as an error between a current measurement $s(q)$ and a desired value of it s^*. The functions used in the following are the end-effectors position and orientation of the right hand e_{rh}, left hand e_{lh} and feet e_{rf}, e_{lf}. Both feet are controlled together on the ground by stacking the two last tasks $e_{feet} = (e_{rf}, e_{lf})$. The center of mass (COM) is controlled to a given point by e_{com}. The orientation of one vector attached to the robot (for example, having the hand around a stick but able to rotate around the stick axis) is controlled by regulating the cross product:

$$e_\theta = u(q) \times u^*(\Omega) \tag{42}$$

where u is the vector attached to the robot to be placed on u^*. More details about these classical task functions can be found in [Sentis, 2007, Kanoun, 2009]. The regulation of the error is realized by the proportional correction (38). The linear constraint is finally:

$$J\dot{q} = \dot{e}^* - \frac{\partial e}{\partial \Omega}\dot{\Omega} \tag{43}$$

where $J = \frac{\partial e}{\partial q}$ is the robot task Jacobian.

The second set of tasks defines a bound on a function of the robot configuration:

$$e^l(\Omega) \leq e(q) \leq e^u(\Omega) \tag{44}$$

This constraint is homogeneous to the configuration. The linear constraint is obtained by the first-order Taylor approximation:

$$\frac{\kappa}{\Delta t}(e^l(\Omega) - e(q)) \leq J\dot{q} \leq \frac{\kappa}{\Delta t}(e^u(\Omega) - e(q)) \tag{45}$$

where κ is the time horizon (in number of control iteration) used as a gain modifier [Faverjon and Tournassoud, 1987].

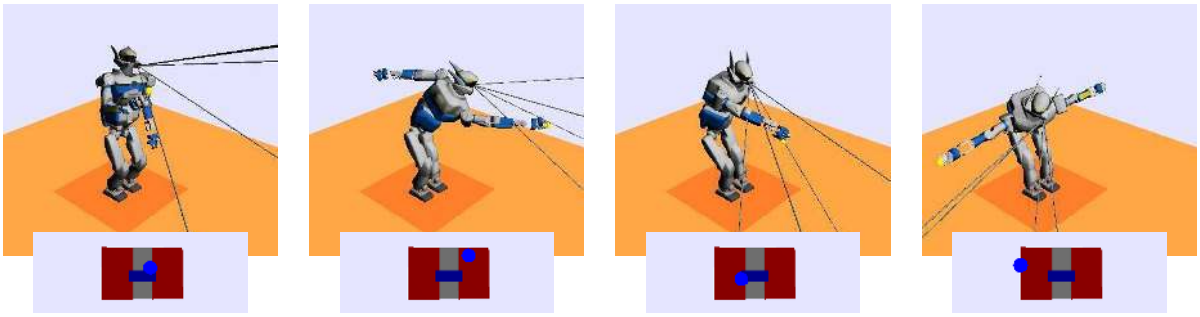
Several task functions e can be used to obtain various behaviors. The joint-limit task e_{jl} bounds the robot configuration q by a set of fixed value. The task e_{supp} keeps the COM in the support polygon. The field-of-view (FOV) task e_{fov} considers the projection of an object on the image plane and bounds it by the image border. The collision avoidance is enforced by the task e_{coll} by imposing the distance between a body of the robot and an object to be positive. In that case, the pair of bodies and object to check has to be specified (no systematic collision checking was performed here, it should be considered in the future [Stasse et al., 2008]). Once more, details about these classical functions can be found in [Sentis, 2007, Kanoun, 2009].

5.3 Experiment A: grasping using conditional visual guidance

The robot has to grasp a point object while keeping its balance and respecting its joint limits. During the task, the robot tries to keep the object in its FOV and, if possible to keep its COM into a small 2cm-wide band inside its support polygon to obtain a well-balanced posture (e_{bal}). The task order is then $e_{jl} \prec e_{feet} \prec e_{supp} \prec e_{rh} \prec e_{fov} \prec e_{bal}$. The robot motion is summarized by Figures 6 to 14.

The ball is moved in four different places as shown in Fig. 6: first in front of the robot, in an easily reachable position; then far in the left, so that the robot has to bend to reach it; the ball is put back to the initial position before putting it to a far right position that is not reachable while keeping the COM inside the support polygon. At the first ball position, the two last tasks are respected: the ball is inside the FOV and the COM is in the central band. The second ball position is further away and requires to the robot to importantly bend to reach it. The COM cannot be kept inside the central band while satisfying all the other tasks. Relaxing the least-priority COM task enables to satisfy e_{fov} and e_{rh}. The higher-priority COM task is respected, that ensures the robot balance. The ball is then placed back in front of the robot: the COM comes back to the central band while all the other tasks are kept satisfied. The last position is unreachable while keeping the COM in the support polygon. All the tasks from the least-priority one are successively relaxed until the minimal distance to the ball is finally reached: at the final position, the COM is outside of the central band, on the border of the support polygon, and the ball is outside the FOV. This is a typical case of interest of the hierarchy: a proper behavior is ensured by the tasks having priority (balance, joint limits) while the optional objectives are satisfied at best.

The task sequence is given in Fig. 7. In the beginning of each motion sequence (when the ball is just moved), the visibility constraint (44) might be violated without the FOV task (45) being violated: the control is simply bringing the ball inside the FOV boundaries according to the task definition. The task becomes violated when (45) cannot be fulfilled. Details about the COM and FOV satisfaction are given in Figures 8 and 9. At the beginning of the third motion sequence, the COM is outside of the central band. It is brought back to this zone after 0.2 seconds. Similarly, at the beginning of each sequence, the ball is outside of the FOV and is quickly brought back. At time $T = 4s$, the COM is at the central-band limit when several joints reach their limits (see Fig. 10). This reduces the available DOF for the grasp task, and following for e_{bal}, which has to be relaxed: the COM leaves the central band. Similarly, at $T = 9s$, the COM is on the border of the band. The activation of some joint limits once more drives the COM outside of the central band. At $T = 9.5s$, some DOF of the grasp task e_{rh} collapse because of a kinematic singularity. The reallocation of the DOF used by the least-priority tasks leads first the X coordinate of the COM to leave the central band, then the ball to leave the FOV. The COM quickly escapes the central band, until it finally reaches the second COM bound imposed by e_{supp}. The limitation of the COM causes the violation of e_{rh}: the robot then stops as close as possible to the ball. Some typical trajectories of the joints are shown in Fig. 10: the limits are always respected. The number of active constraints for all levels together (*i.e.* the size of the optimal active set) is displayed in Fig. 11.



(t=2s) Ball in front (t=5s) Far left (t=7s) Front (t=10.4s) Far right

Fig. 6: *Top row: snapshots of the robot motion. Bottom row: corresponding COM projection (blue point) in the support polygon (brown rectangles depict the feet, the blue rectangle shows the area for e_{bal}). Each snapshot is captured at the end of a motion sequence. The FOV is displayed by the 4 lines passing by the center of the image projection.*

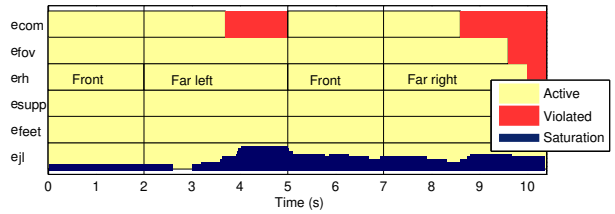


Fig. 7: *Experiment A: task sequence, listed by priority from bottom to top. The tasks are specifically marked when they become violated. The hierarchy appears through the violation order: the least-priority tasks are relaxed first in case of conflicts. The number of saturated joint limits is displayed in the e_{jl} row.*

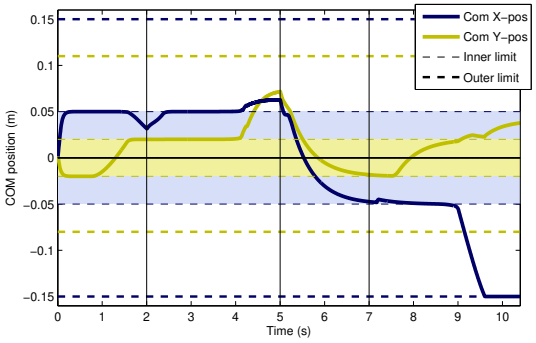


Fig. 8: *Experiment A: position of the COM wrt the inner and outer limits. The COM has to remain into the outer limits to ensure the balance of the robot, and should be kept if possible inside the central band (inner limits) to obtain a balanced robot posture.*

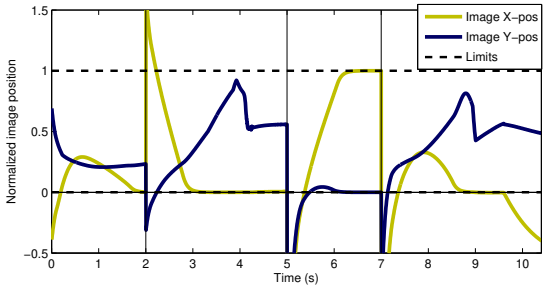


Fig. 9: *Experiment A: position of the object projection in the image plane wrt the FOV limits. When the ball is moved outside the FOV, e_{fov} brings it back into the FOV limits. At $T = 0s, 2s, 5s$ and $7s$, the ball is artificially moved out of the FOV and the robot brings it back following the task reference. The robot loses the ball at $T = 9.8s$ due to a conflict with the tasks having priority.*

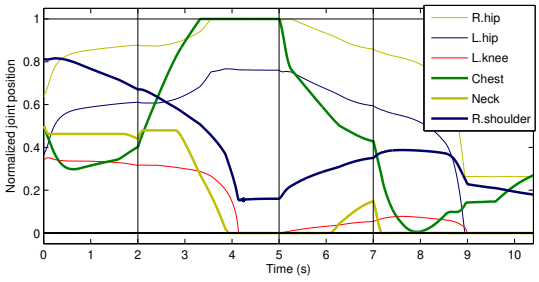


Fig. 10: *Experiment A: normalized joint positions.*

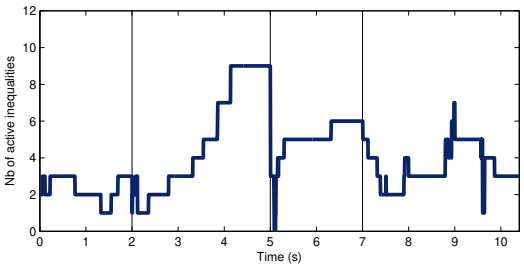


Fig. 11: *Experiment A: number of active inequalities at each control cycle.*

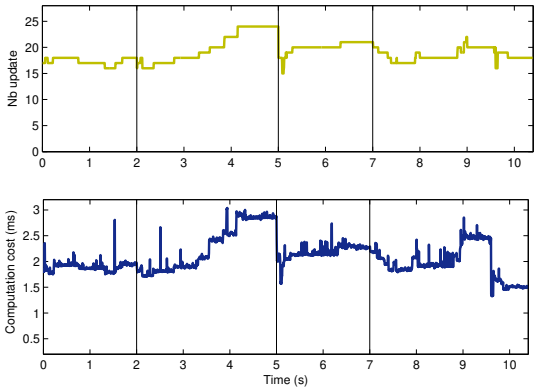


Fig. 12: *Experiment A: Number of algorithm iterations and computation time when using a cascade of QP [Kanoun et al., 2011].*

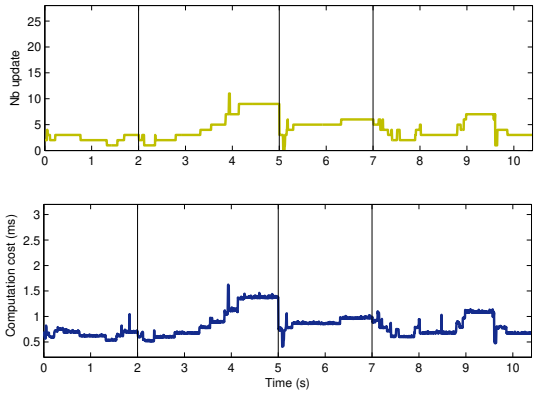


Fig. 13: *Experiment A: Number of algorithm iterations and computation time when no initial guess is used.*

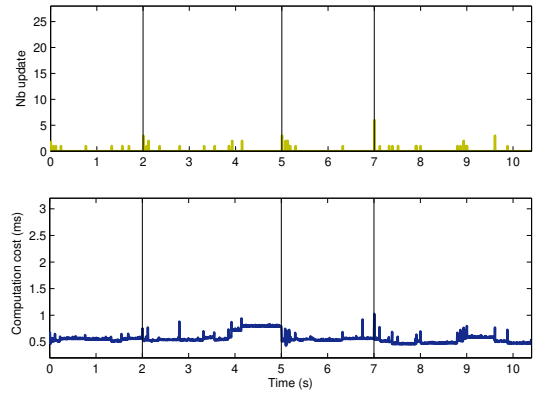


Fig. 14: *Experiment A: Number of algorithm iterations and computation time when using a warm start based on the previous control cycle.*

Finally, the computation cost is experimentally studied. We compare the HQP resolution using a cascade of QP as proposed in [Kanoun et al., 2011] (Fig. 12), with our method using an empty initial guess (Fig. 13) and using a warm start as proposed in Sec. 4.4 (Fig. 14). First, the number of iterations in the active search loop is much higher with a cascade of QP than using the proposed HQP solver. The number of iterations in the active search is very similar to the number of active constraints at the optimum, as can be seen by comparing Fig. 11 to Fig. 13-(top). Our solver barely needs any deactivation to find the optimal active set. This is not the case when using a cascade of QP: Fig. 12-(top) shows that approximately twice as many iterations are need to reach the optimal active set. As expected, the number of iterations is even lower using a proper warm start: in that case, the active search only iterates

when a new boundary is reached. The maximal number of iterations is 6 (at the first iteration after the change of the ball position at T=7), the mean number is 0.03 and in 97.6% of the case, the active search converges without any update. As shown in Sec. 4.1, the computation time depends on the number of active constraints and of active-search iterations. Since there is nearly no iteration, Fig. 14-(bottom) depends only on Fig. 11 and has the same shape. In Figures 12-(bottom) and 13-(bottom), the influence of the number of iterations is more important, and the time graph shape is similar to the graph of number of iterations. Using the HQP and the warm start, an average of 0.56ms of computation is needed.

5.4 Experiment B: opening a valve

The robot has to open a valve by manipulating a wheel. The motion is composed of two parts: the robot first manipulates the wheel using one hand, then rotates the wheel using both hands with successive re-grasps. During the motion, the robot has to avoid a block located on its left, and to keep its COM inside the support polygon. When grasping the wheel, the robot has to look at it; when rotating the wheel, it has to look at a pression meter located on its left. All the inequalities are considered at the top priority levels so that the HQP behavior can be compared to the solution proposed in [De Lasa et al., 2010]. The first movement (left-arm manipulation) is summarized in Figures 15 to 18. The second movement (both-arm manipulation) is summarized in Figures 19 to 24. The comparison of the computation times are given in Figures 25 to 27.

5.4.1 First movement

Snapshots of the first motion are shown in Fig. 15. The task sequence is detailed in Fig. 16. The constraints are the joint limits, the support polygon, the FOV and the distance of the left elbow and shoulder to the left obstacle. The left-hand task is divided into the translation e_{Tlh} and rotation e_{Rlh} components. During the approach, both the left-hand rotation and translation are controlled. When the robot rotates the wheel, the rotation of the hand

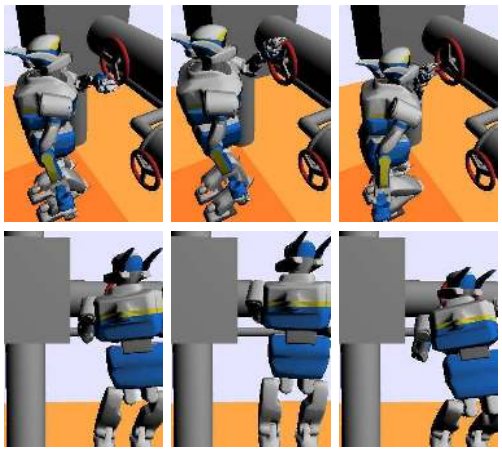


Fig. 15: *Experiment B-1: Snapshots of the first movement: the robot uses only its left hand to manipulate the wheel. The three snapshots are captured during the wheel rotation for three angles of* 0, $\frac{2\pi}{3}$ *and* $\frac{4\pi}{3}$.

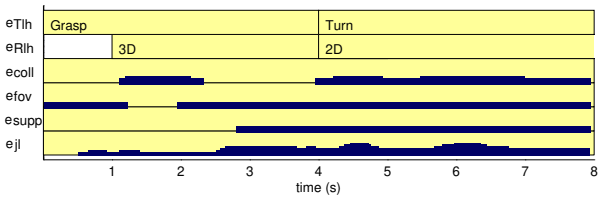


Fig. 16: *Experiment B-1: Task sequences of the first movement. The wheel is rotated of two complete loops. For the four inequality tasks, the number of active constraints during the motion are plotted for each levels.*

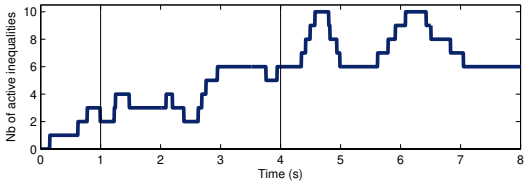


Fig. 17: *Experiment B-1: Number of active inequalities during the first movement.*

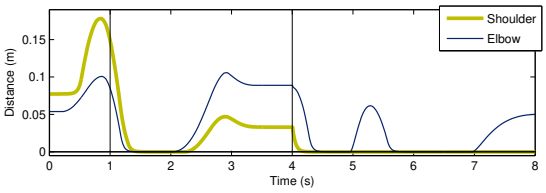


Fig. 18: *Experiment B-1: distance to the obstacle during the first movement. The obstacle is on the way between the left-arm initial position and the wheel: the two constraints of the shoulder and the elbow become active at* $T = 1.1s$ *and* $T = 1.2s$*. They are deactivated later during the grasping phase, at* $T = 2s$ *and* $T = 2.1s$*. At each loop of the wheel, the left arm comes close to the obstacle, between* $T = 4s$ *and* $T = 5s$*, and a second time between* $T = 5.7s$ *and* $T = 7s$*.*

around the wheel axis is let free and only two degrees of rotation are controlled. The hierarchy is $e_{jl} \prec e_{supp} \prec e_{fov} \prec e_{coll} \prec e_{feet} \prec e_{Tlh} \prec e_{Rlh}$. The number of active constraints for the four first levels is also shown in Fig. 16. The total number of active inequalities is given in Fig. 17. When the robot is on the left part of the wheel, the obstacle strongly constraints the robot, which causes an increase of the number of active constraints: two peaks appear for each loop of the wheel. The distances of the shoulder and elbow to the obstacle is given in Fig. 18. The arm comes close to collision when the robot approaches the wheel: the constraints are saturated to prevent it. The constraints are then deactivated when the robot goes away from the obstacle. When the robot starts to rotate the wheel, the constraints become once more active. Since the motion is not holonomic [Zanchettin and Rocco, 2012] (in particular, there is no posture task), the motion realized for each loop is different: during the second loop, the elbow constraint remains saturated.

5.4.2 Second movement

The robot then uses the second arm to ease the manipulation of the wheel. The motion is then more constrained since both hands are bound to the wheel. Snapshots of the motion are given in Fig. 19. The task sequence is given in Fig. 20: each hand first reaches an arbitrary pre-grasp position before grasping the wheel. During the approach, the three rotations are controlled, while the rotation along the tangential axis to the wheel is left free during the manipulation. The distance to the obstacle is plotted in Fig. 21. The constraint becomes active at the end of the motion. The joint position with respect to the limit is shown in Fig. 22. Contrary to the previous experiment, the joints do not systematically remain on the exact limits since the robot is moving to follow the rotation of the wheel. The number of active constraints is given in Fig. 23. The number of active constraints increases with the complexity of the task. It reaches its maximum just before the robot starts to move the wheel (see Fig. 19-left, the robot is bent on the right with many apparent saturations). It then decreases when the robot stands straight (see Fig. 19-middle), and increases again at the end of the motion (see Fig. 19-right). Finally, the conditioning number of the left- and right-hand tasks is shown in Fig. 24. The conditioning number evolves both continuously with the changes in the Jacobians and discreetly at each constraint activation.

5.4.3 Computation times

Finally, the computation cost of the overall motion is studied. The computation time and corresponding number of iterations of the active set are plotted in Fig. 25. In 97.5% of the cases, the active search loop converges without any update. The mean number of iterations is 0.035 and the maximum is 9. The mean of computation time is 0.9ms. The peaks of number of iteration correspond to peaks of computation time. When 9 iterations are needed, the algorithm takes 3.7ms to converges. This effect emphasizes the fact that the active search is not real time: it is not possible to predict how many iterations are needed to reach the optimum and, even if the mean is below 1ms, a control frequency of 1kHz is not possible.

As proposed in Sec. 4.4, it is possible to arbitrarily limit the number of iterations to enforce a deterministic convergence time. In Fig. 26, the algorithm can do only one update at each control cycle. When a second update is requested, the algorithm quits without even

Fig. 19: *Experiment B-2: Snapshots of the second movement: the robot uses both hands to manipulate the wheel. For the sake of clarity, the environment is not displayed.*

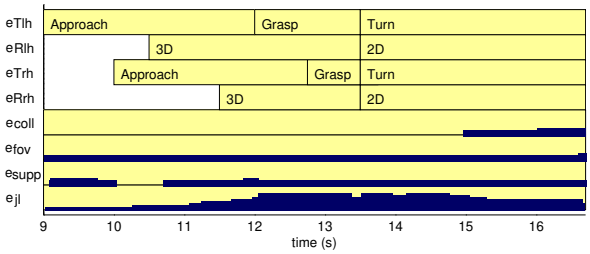


Fig. 20: *Experiment B-2: Task sequence of the second movement: the plots show the grasping phase (from $T = 9s$ to $T = 13.5s$) and a rotation of $\frac{2\pi}{3}$. For the four inequality tasks, the number of active constraints during the motion is plotted for each levels.*

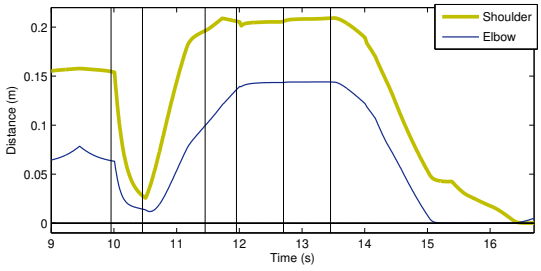


Fig. 21: *Experiment B-2: Distance to the obstacle.*

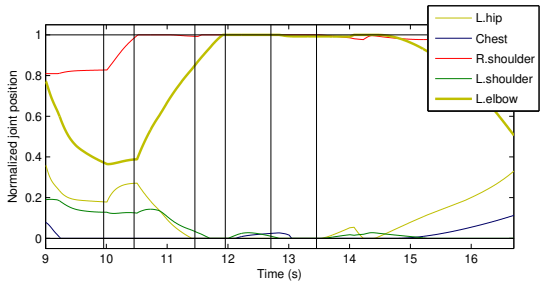


Fig. 22: *Experiment B-2: Normalized joint positions during the second movement.*

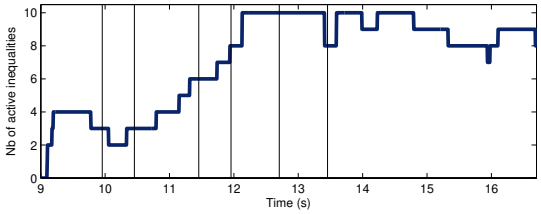


Fig. 23: *Experiment B-2: Number of active inequalities during the second movement.*

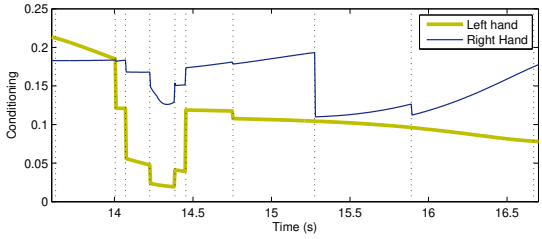


Fig. 24: *Experiment B-2: Conditioning number of the hand tasks when both hands are moving the wheel. The vertical dot lines show the inequality activations. The conditioning number evolves both continuously with the changes in the Jacobians and discreetly at each constraint activation.*

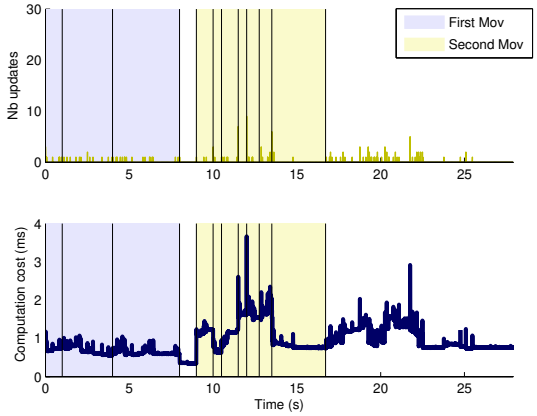


Fig. 25: *Experiment B: Number of algorithm iterations and computation time when using a warm start based on the previous control cycle.*

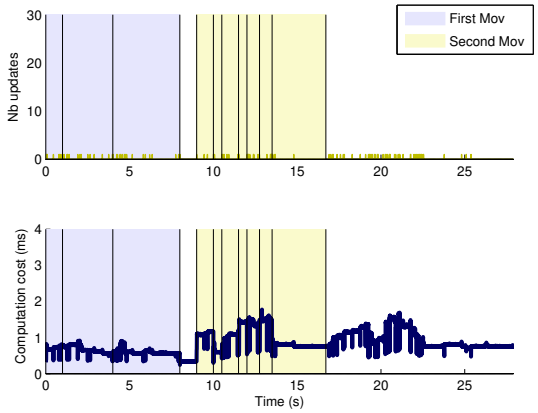


Fig. 26: *Experiment B: Number of algorithm iterations and computation time when using a warm start and a limitation of active-set iterations.*

computing the dual optimum. The peaks of computation disappear. In exchange, the number of control cycles without update decreases to 96.6%. There is no perceptible changes in the robot behavior, since, due to the continuity properties, the obtained sub-optimum is very close to the solution.

Since the only inequalities are at the first levels of the hierarchy and are always compatible, the solver proposed in [De Lasa et al., 2010] can be used to generate the motion. The computation time and corresponding number of iterations of the active-set loops are plotted in Fig. 27. As already noticed with the cascade of QP, the active search needs many iterations to find the optimal active set (up to 30, with a mean at 4.5). However, since each QP solved in the cascade is very small, the number of iterations only slightly impacts the computation cost. In most of the cases, the convergence is slower than in Fig. 25 (the mean is 1.3ms), but there is no peak of computation like with our solver (the maximum is 2.6ms). The computation time is always higher than in Fig. 26.

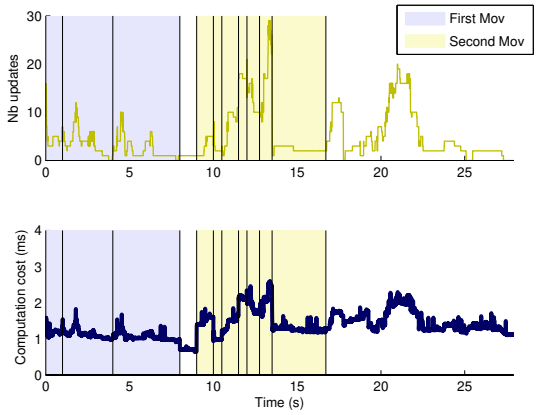


Fig. 27: *Experiment B: Number of algorithm iterations and computation time when using the projected cascade described in [De Lasa et al., 2010].*

5.5 Experiment C: grasping with posture constraints

This experiment is executed by the real HRP-2 robot. The robot has to grasp a point object while looking at it and avoiding its joint limits and the collisions with the environment. Three tasks are set at the least-priority levels to enforce the use of the upper limbs of the robot: the task e_{legs} is regulating the joint positions of the legs to the reference initial position; the task e_{chest} is regulating the orientation of the chest to keep it vertical, using (42). Finally, the last task e_{up} is blocking the upper part of the robot (chest, arms and neck). This kind of behavior using only the necessary limbs to perform an action was proposed

in [Ee et al., 2007] using dedicated geometrical computations. The hierarchy is $e_{jl} \prec e_{coll} \prec e_{supp} \prec e_{rh} \prec e_{fov} \prec e_{legs} \prec e_{chest} \prec e_{up}$. The motion is summarized by Figures 28 to 33.

The ball is moved in three different positions, as shown in Fig. 28: first close in front of the robot, then at the height of the waist and finally on the ground behind a small box. The grasping task is finally removed when the last position is reached. The task sequence is shown in Fig. 29. When the ball is close enough, only the least-priority task e_{up} is violated, and the robot is grasping the ball using only its right arm. The neck and the left arm are marginally used, respectively for the FOV task and support task. When the ball is placed at the second position, it is out of the reach of the arm alone. The task e_{legs} is violated at the end of the grasping motion. When the ball is on the ground, it is not possible to grasp it with the chest being vertical. The task e_{chest} is violated at the end of the motion to reach the ball. Finally, the task e_{rh} is removed. The three tasks e_{legs}, e_{chest} and e_{up} are then feasible, and the robot goes back naturally to its initial position.

The errors of the four tasks are given in Fig. 30 and illustrate very well the hierarchical order: the task e_{rh} has priority over the three other ones, and is always accomplished: the error exponentially converges as imposed. The task e_{up} is violated first, and its error is the most important. The task e_{legs} is violated then, while the task e_{chest} is violated at the end, and keeps the lowest error value.

The activation of the limits is synthesized on Fig. 29. Some examples of activations are given in Figures 31 and 32. The left hand is moving backward to ensure the robot balance, and is quickly blocked by the task preventing the collision with the wall situated behind the robot. The right hand avoids a collision with the box when going to the third target position. Both collision constraints are deactivated when the robot moves back to the initial pose. Some joints limits of the legs are saturated when the robot goes for the second target position, and many limits are saturated when reaching the third target.

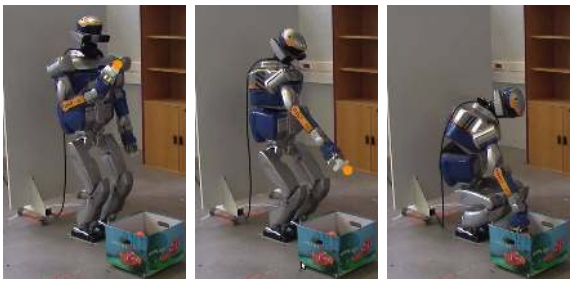
Finally, the number of active inequality constraints, and the number of iterations of the active-search loop are given in Fig. 33. As previously, the active set of the previous control cycle is used as warm start and reduces the number of iterations of the loop. The maximum number of iterations is 7 and is reached when the task e_{rh} is removed from the hierarchy. In average, 0.035 iterations and 1.1ms are needed at each control cycle, and 97.8% of the cycles are resolved without extra iteration.

6 Conclusion

In this paper, we have proposed a generic solution to resolve a hierarchical least-square quadratic program defined by equality or inequality constraints. When only equalities are set, the method comes back to the classical stat-of-the-art solutions [Hanafusa et al., 1981, Siciliano and Slotine, 1991, Baerlocher and Boulic, 2004] but is up to ten times faster. When the problem encompasses inequality constraints, a true hierarchy is solved with inequalities at any level. The resolution loop keeps a low number of iterations by using a unified active-search algorithm, in contrast to the cascades of QP used in [Kanoun et al., 2011, De Lasa et al., 2010]. Using a proper construction of the active-search loop, the resolution can be performed in real-time at 200Hz using a classical personal computer and no specific hardware tuning.

The solver is generic and can be applied to many problems in robotics and beyond. We have proposed to apply it to compute the control law of a redundant robot in the inverse-kinematics context. In that case, we have shown that the control is continuous for a given hierarchy, and stable. The method was used to generate several complex motions with the humanoid robot HRP-2, in presence of realistic constraints such as joint limits, obstacle or field of view. For most of the control cycles (97% in average in the presented experiments), the active-search loop does not need any iterations, which implies that the solver deals with inequalities as if they were equalities. The presented experiments validated that the proposed method is the first complete solution to handle inequalities at any level inside a hierarchy of tasks to generate robot motions in real time.

In the future, we will try to reduce the computation



(t=2.1s) (t=4.2s) (t=7.3s)

Fig. 28: *Experiment C: Snapshots of the robot movement.*

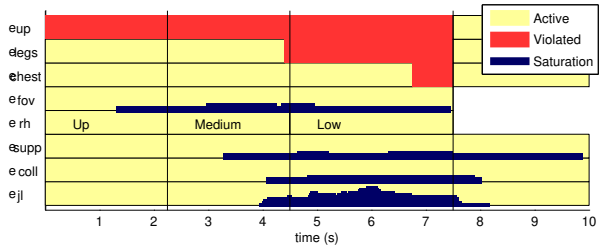


Fig. 29: *Experiment C: Task sequence.*

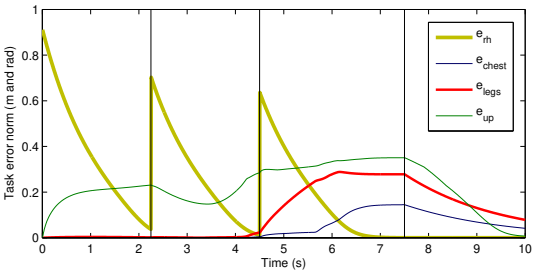


Fig. 30: *Experiment C: Norm of the task errors.*

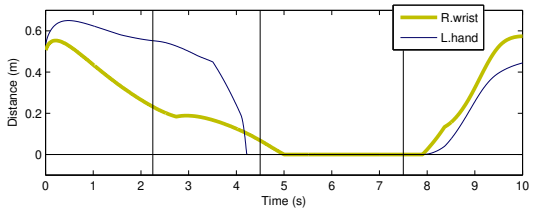


Fig. 31: *Experiment C: Distance to the obstacle.*

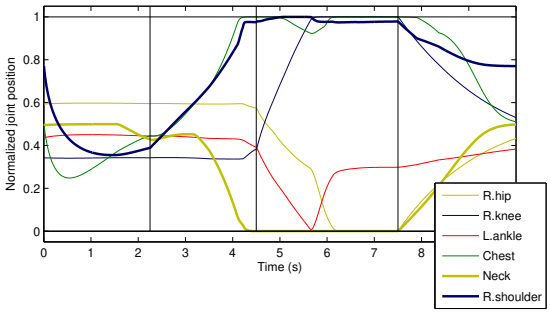


Fig. 32: *Experiment C: Normalized joint positions.*

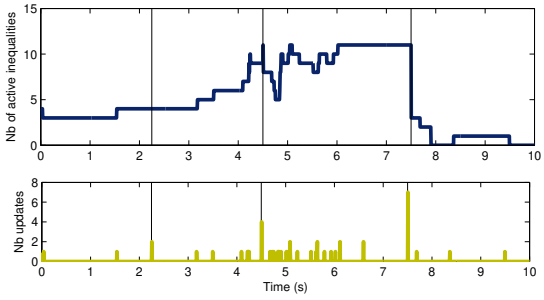


Fig. 33: *Experiment C: Number of active inequalities and updates.*

cost by predicting the future constraint activations. The solver was only applied to inverse the robot kinematics, but the dynamics could be handled as well. Only a simple obstacle-avoidance scheme was set up, and a proper link with a complete collision checker should be studied. Finally, the continuity of the control scheme is not ensured when adding or removing a task from the hierarchy, which is needed before being able to apply it as a basic solution on our robots. Openings to other kind of problems, such as those solved by a walking pattern generator, is also an important perspective.

A Index to Multimedia Extensions

Ext.	Type	Description
1	Video	The movement sequences presented in the experimental sections.
2	Code	MATLAB code described in App. D.

B Elementary transformation

This appendix recalls the basis on Householder reflections and Givens rotations.

B.1 Householder reflections

Householder reflections are orthogonal transformations built from any vector v:

$$Q = I - \frac{2}{v^T v} v v^T \tag{46}$$

If v is chosen as $v = (0, \dots, 0, a_1 - \|a\|, a_2, \dots, a_n)$, then:

$$Q \begin{bmatrix} \times \\ \vdots \\ \times \\ | \\ a \\ | \end{bmatrix} = \begin{bmatrix} \times \\ \vdots \\ \times \\ \|a\| \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{47}$$

Householder reflections are rank-1 modifications of the identity. They are very useful to zero the tail of a vector or of a matrix column. They do not affect the head of the vector they are applied on. The explicit shape of Q should not be computed, but only v should be stored. See [Golub and Van Loan, 1996] for more details.

B.2 Givens rotations

Givens rotations of any dimension d are defined by:

$$G[i, j, \theta] = I_d - S_{ij}(I_2 - R_\theta)S_{ij}^T \tag{48}$$

with I_d and I_2 the identity matrices of dimension d and 2 respectively, $R_\theta = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$ the 2D rotation of angle θ and $S_{ij} = \begin{bmatrix} e_i & e_j \end{bmatrix}$ with e_i and e_j the canonical vectors (zero everywhere except on the corresponding row). A right multiplication with a Givens rotation only modify the two rows i and j. It can be used to introduce a zero inside a vector or a matrix:

$$G[i, j, \theta]^T \begin{bmatrix} \vdots \\ x_i \\ \vdots \\ x_j \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \|x_i, x_j\| \\ \vdots \\ 0 \\ \vdots \end{bmatrix} \tag{49}$$

when $\theta = \tan^{-1}(x_j, x_i)$. Givens rotations are rank-2 modifications of the identity. See [Golub and Van Loan, 1996] for more details. In this paper, Householder and Givens matrices are referred to as elementary transformations.

C Proof of Theorem 4.3

We define the Lyapunov energy function by $V = \frac{1}{2} {e_E}^T e_E$. Then

$$\dot{V} = {e_E}^T \dot{e}_E = {e_E}^T J_E \dot{q} \tag{50}$$

where $\dot{q}$ is computed from the eHQP of the current optimal active search:

$$\dot{q} = Y \underline{H}_p^{\ddagger} \underline{W}_p^T \underline{b}_p \tag{51}$$

Suppose that the active set of $\mathcal{S}$ is built incrementally from the initial guess S_E, using the update procedure of Sec. 3.2. The initial decomposition involves only J_E, with the following notations:

$$J_E = \underline{W}_E \underline{H}_E Y_E^T \tag{52}$$

For each active component of the inequality levels S_I, there are two options: it corresponds to a column Y_i of Y_E (case 2.2 of the update) or it is decoupled. In the second case, the component can be equivalently activated at the least-priority level. The optimum is then equivalently written:

$$\underline{y}^* = (y_1^*, \ldots, y_p^*, y_{p+1}^*) \tag{53}$$

with y_{p+1}^* the equivalent level where all the decoupled constraints of $\mathcal{S}$ are set and all the active inequalities of levels 1 to p are coupled with J_E: both bases Y and Y_E are the same with a permutation Π on the column order: $Y = Y_E \Pi$.

The optimum $\underline{y}_p^*$ of the full iHQP can be computed from the optimum of the eHQP corresponding to J_E, denoted by $\underline{y}_E^*$:

$$\underline{y}_p^* = \begin{bmatrix} y_1^* \\ \vdots \\ y_p^* \end{bmatrix} = \Pi^T \begin{bmatrix} (1-\tau_1) y_{E1}^* \\ \vdots \\ (1-\tau_p) y_{Ep}^* \end{bmatrix} = \Pi^T \, \Delta_\tau \, \underline{y}_E^* \tag{54}$$

where the τ_k are the step lengths met during the update phases and Δ_τ is the diagonal matrix of the $1 - \tau_k$.

Introducing this last form into (50):

$$\dot{V} = {e_E}^T \underline{W}_E \underline{H}_E \Delta_\tau \underline{y}_E^* \tag{55}$$

$$= {e_E}^T \underline{W}_E \underline{H}_E \Delta_\tau \, \underline{H}_E^\ddagger \underline{W}_E^T e_E \tag{56}$$

which is non-negative since Δ_τ is positive and $\underline{H}_E^\ddagger$ respects the three first conditions of Moore-Penrose.

If J_E is full rank, all the N of $\underline{H}_k$ are empty. Moreover, if none of the equality levels of the iHQP are rank deficient, then it is as if all the inequalities had been activated at a least-priority level p. Then:

$$\dot{V} = {e_E}^T e_E > 0 \tag{57}$$

which ensures the asymptotic stability. □

D MATLAB code

The main algorithms proposed in the paper are summarized by a short Matlab solver attached to it. The software solves the following HQP:

$$\underset{x, w_1 \ldots w_p}{\operatorname{lex\,min}} \; \{\|w_1\|, \|w_2\|, \ldots, \|w_p\|\}\,. \tag{58}$$

$$\begin{aligned} \text{subject to} \quad \forall k = 1..p \quad A_k^E x &= b_k^E && (59)\\ b_k^I &\le A_k^I x && (60)\\ A_k^S x &\le b_k^S && (61)\\ b_k^l \le A_k^{lu} x &\le b_k^u && (62) \end{aligned}$$

For each constraint, b is given by a pair of floats and an integer identifying the constraint to one of the four constraint types specified above.

The problem specifications $\underline{A}_p, \underline{b}_p$, the current active set and the corresponding HCOD are stored in a dedicated array h: each cell of the array corresponds to one level of the hierarchy and stores A_k, b_k, W_k, H_k and the active constraints. The right basis Y is independently stored. The software is organized in four main functions split in the same number of files:

- `hcod` computes the HCOD from the problem specification $\underline{A}_p, \underline{b}_p$ and an initial guess of the active set. The function returns h and Y that are manipulated by the other function during the active-search loop.
- `eHQP_primal` computes the primal optimum of the eHQP corresponding to the current active set. The function returns the optimum x^* and the corresponding expression in the Y basis y^*. This corresponds to Alg. 1
- `eHQP_dual` computes the multiplier $\underline{\lambda}_k$ of one level k of the hierarchy, as proposed in Alg. 2.
- `active_search` is the main function: it computes the HCOD and then runs the search loop. It returns the primal and dual optima and the HCOD. It uses the fuctions `step_length` (see Alg. 4), `check_multipliers` and `freeze` (to test the lexicographic positivity of the multipliers), and `up` and `down` (to add and remove one row of the HCOD).

To simplify the code, the solver computes explicitly the left and right bases of the HCOD instead of using structured matrices. It is given as a detailed example to guide the implementation of a hierarchical QP solver, but should not be used for reactivity tests. An optimized version of the solver coded in C++ using the Eigen library and structured-matrix bases was used for the experiments.

Acknowledgments

This work was supported by grants from the RobotHow.cog EU project, Contract No. 288533 under the 7^{th} research program (`www.robohow.eu`), and French FUI Project ROMEO. Many thanks to C. Benazeth, A. El Khoury, F. Keith, A. Kheddar and F. Lamiraux for their help.

References

[Antonelli, 2009] Antonelli, G. (2009). Stability analysis for prioritized closed-loop inverse kinematic algorithms for redundant robotic systems. *IEEE Trans. on Robotics*, 25(5):985–994.

[Baerlocher and Boulic, 2004] Baerlocher, P. and Boulic, R. (2004). An inverse kinematic architecture enforcing an arbitrary number of strict priority levels. *The Visual Computer*, 6(20):402–417.

[Ben-Israel and Greville, 2003] Ben-Israel, A. and Greville, T. (2003). *Generalized inverses: theory and applications*. Springer, 2nd edition.

[De Lasa et al., 2010] De Lasa, M., Mordatch, I., and Hertzmann, A. (2010). Feature-based locomotion controllers. In *ACM SIGGRAPH'10*.

[Ee et al., 2007] Ee, N., Yokoi, K., Kajita, S., and Tanie, K. (2007). Whole-body motion generation integrating operator's intention and robot's autonomy in controlling humanoid robots. *IEEE Trans. on Robotics*, 23(4):763–775.

[Escande et al., 2012] Escande, A., Mansard, N., and Wieber, P.-B. (2012). Hierarchical quadratic programming (part 1). *Submitted*.

[Faverjon and Tournassoud, 1987] Faverjon, B. and Tournassoud, P. (1987). A local based approach for path planning of manipulators with a high number of degrees of freedom. In *IEEE Int. Conf. on Robotics and Automation (ICRA'87)*, volume 4, pages 1152–1159, Atlanta, USA.

[Golub and Van Loan, 1996] Golub, G. and Van Loan, C. (1996). *Matrix computations*, chapter 5.1: Householder and Givens Matrices. John Hopkins University Press, 3rd edition.

[Hanafusa et al., 1981] Hanafusa, H., Yoshikawa, T., and Nakamura, Y. (1981). Analysis and control of articulated robot with redundancy. In *IFAC, 8th Triennal World Congress*, volume 4, pages 1927–1932, Kyoto, Japan.

[Kanoun, 2009] Kanoun, O. (2009). *Contribution à la plannification de mouvements pour robots humanoïdes (in English)*. PhD thesis, Univ. Toulouse, France.

[Kanoun et al., 2011] Kanoun, O., Lamiraux, F., and Wieber, P.-B. (2011). Kinematic control of redundant manipulators: generalizing the task priority framework to inequality tasks. *IEEE Trans. on Robotics*, 27(4):785–792.

[Khatib, 1986] Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *Int. Journal of Robotics Research*, 5(1):90–98.

[Liégeois, 1977] Liégeois, A. (1977). Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Trans. on Systems, Man and Cybernetics*, 7(12):868–871.

[Sentis, 2007] Sentis, L. (2007). *Synthesis and Control of Whole-Body Behaviors in Humanoid Systems*. PhD thesis, Stanford University, USA.

[Siciliano and Slotine, 1991] Siciliano, B. and Slotine, J.-J. (1991). A general framework for managing multiple tasks in highly redundant robotic systems. In *IEEE Int. Conf. on Advanced Robotics (ICAR'91)*, Pisa, Italy.

[Stasse et al., 2008] Stasse, O., Escande, A., Mansard, N., Miossec, S., Evrard, P., and Kheddar, A. (2008). Real-time (self)-collision avoidance task on a HRP-2 humanoid robot. In *IEEE Int. Conf. on Robotics and Automation (ICRA'08)*, Pasadena, USA.

[Zanchettin and Rocco, 2012] Zanchettin, A. and Rocco, P. (2012). A general user-oriented framework for holonomic redundancy resolution in robotic manipulators using task augmentation. *IEEE Trans. on Robotics*, 28(2):514–521.