AN OPTIMIZED LAGRANGIAN-MULTIPLIER APPROACH FOR INTERACTIVE MULTIBODY SIMULATION IN KINEMATIC AND DYNAMICAL DIGITAL PROTOTYPING

A. Tasora

Politecnico di Milano
Dipartimento di Ingegneria Strutturale (DIS) - Sezione Trasporti e Movimentazione
Piazza Leonardo Da Vinci, 32, 20133 Milano, Italy
e-mail: tasora@mech.polimi.it , web page: http://mech.polimi.it

Keywords: multibody, lagrangian multipliers, quaternions, sparse systems, interactive dynamics.

Abstract. The dynamical simulation of nonlinear mechanical systems, in a context of real time interaction between human designer and computer, requires a fast and general-purpose solution scheme. Despite the approach based on lagrangian multipliers implies the introduction of many redundant variables, hence penalizing computational speed if compared to other methods, it also offers the most universal and versatile way to describe articulated systems. Therefore, by using specific formalisms borrowed from quaternion algebra, we implemented some expedients which make the lagrangian approach as fast as possible, allowing custom factorization of sparse matrices and enhanced constraint stabilization. Meanwhile this suggests a solution scheme which can handle, with a minimal set of formalisms, some problems implicated by realtime user-interaction with mechanisms, such as intermitting contacts, collisions and ill-posed constraints.

1. Introduction

The evolution of the computational resources in modern computers encourages a growing interest for virtual prototyping of mechanical devices within 3d graphical interfaces.

This means that complex mechanisms, such as prosthetic devices or robotic arms, can be sketched, assembled and moved in a CAD-like environment, thus allowing the designer to test the kinematic and dynamical behavior by mouse interaction.

Usually there are two distinct approaches to the simulation of mechanical systems: the method of 'joint coordinates' (or 'recursive' or 'reduced' coordinates') and the method of 'cartesian' or 'maximal' coordinates. The former approach introduces the least possible number of free coordinates to describe the state of the system (usually the rotations in joints), while the latter introduces an huge amount of coordinates as if rigid bodies were not constrained, and some algebraic equations are be added in order to represent the constraints between the parts by means of the so-called "lagrangian multipliers".

While the 'reduced coordinates' methods are inherently faster -given their lower dimensionality-, they must challenge the topological problem of parametrizing the system's degrees of freedom, that may be a difficult task, especially when dealing with closed-loop kinematical chains.

On the other hand, lagrangian methods offer the most general and versatile way to model mechanical systems since each entity (rigid bodies, forces,

constraints) can be seen as black boxes, and arbitrary relations can be imposed to variables. This inspires a modular design of the solution method, which fits well into an object-oriented approach and allows an easier implementation of complex constraints (such as intermitting contacts, non holonomic constraints, etc).

The big drawback of lagrangian methods is the fact that they imply the solution of systems with a large number of equations in many variables, hence leaning toward a $O(n^3)$ solution time for n rigid bodies, while most reduced-coordinates methods (for example, the optimal Featherstone $O(n)$ algorithm) have nearly linear dependence to the number of parts, at least for open-loop mechanisms.

However, a recent research (Baraff, 1996) showed that, even in a lagrangian framework, one can exploit the sparsity of acyclic constraint systems in order to develop a direct, non iterative solution scheme with exact $O(n)$ linear time.

Encouraged by Baraff's result, we stuck to the lagrangian approach, and developed special techniques which make the lagrangian method competitive, in terms of speed, with reduced-coordinates schemes.

2. Quaternion kinematics

In order to develop an efficient method for the solution of dynamical and kinematic problem, we must develop some specific formulas for rotation kinematics. We decide to use quaternions as coordinates for the rotation of references not only because this avoids singularities in transformations, but also because quaternion algebra offers a powerful, yet straightforward, way of handling the constraint equations. In fact our simulation software computes constraint jacobians with a fast analytical approach, thank to quaternion algebra. The so called "lock formulation" method, which derives the constraint jacobians analytically for 50 holonomic and rheonomic constraints, is described in (Tasora,1999).

2.1 Basic quaternion algebra

Here are some basic properties of quaternions, applied to kinematic problems.

A quaternion is a tetra-dimensional hyper-complex number $\mathbf{q} \in \{\Re^1, \Im^3\}$ with the following properties:

$$\mathbf{q} = \mathrm{e}_0 + \mathbf{i}\cdot\mathrm{e}_1 + \mathbf{j}\cdot\mathrm{e}_2 + \mathbf{k}\cdot\mathrm{e}_3 \quad (1)$$

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1 \quad (2)$$

where non-commutative multiplication holds between terms: **ij=k**, **ji=-k**, **jk=i**, **kj=-i**, **ki=j**, **ik=-j**.

Separating the vector imaginary part **v** and scalar real part *s*, as in **q**=(*s*,**v**), multiplication between quaternions can be expressed as a Grassman vector product, that is:

$$\mathbf{q}_1\mathbf{q}_2 = \left(s_1 s_2 - \mathbf{v}_1 \circ \mathbf{v}_2 \ , s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2\right) \quad (3)$$

The dot product $\mathbf{q}_1 \circ \mathbf{q}_2$, the sum and the product between quaternion behave as the corresponding traditional vector operations.

The conjugate of a quaternion **q** is $\mathbf{q}^*$, in eq.4, and the euclidean norm is defined in eq.5:

$$\mathbf{q} = (s\ , \mathbf{v}) \qquad \mathbf{q}^* = (s\ , -\mathbf{v}) \quad (4)$$

$$\|\mathbf{q}\| = \mathbf{q}^* \circ \mathbf{q} = (e_0^2 + e_1^2 + e_2^2 + e_3^2) \quad (5)$$

We noticed that the Grassman quaternion product (eq.3) can be written also in terms of linear algebra, using the following symbols:

$$\mathbf{q}_1\mathbf{q}_2 = \left[\overset{+}{\mathbf{q}}_1\right]\mathbf{q}_2 \qquad \left[\overset{+}{\mathbf{q}}_1\right] = \begin{bmatrix} +e_0 & -e_1 & -e_2 & -e_3 \\ +e_1 & +e_0 & -e_3 & +e_2 \\ +e_2 & +e_3 & +e_0 & -e_1 \\ +e_3 & -e_2 & +e_1 & +e_0 \end{bmatrix} \quad (6)$$

as well as:

$$\mathbf{q}_1\mathbf{q}_2 = \left[\overset{+}{\mathbf{q}}_2\right]\mathbf{q}_1 \qquad \left[\overset{+}{\mathbf{q}}_2\right] = \begin{bmatrix} +e_0 & -e_1 & -e_2 & -e_3 \\ +e_1 & +e_0 & +e_3 & -e_2 \\ +e_2 & -e_3 & +e_0 & +e_1 \\ +e_3 & +e_2 & -e_1 & +e_0 \end{bmatrix} \quad (7)$$

2.2 Quaternions for frame kinematics

Only unimodular quaternions ||**q**||=1 describe rigid rotations. Direct and inverse relations between unimodular quaternions and alternative angle representations can be found in many references (Tasora,1999, Shabana1989, Wehage 1984).

Rotation matrices [Λ] can be expressed in terms of quaternions [Λ]= [Λ(**q**)] as follows:

$$[\Lambda] = \begin{bmatrix} 2\left[(e_0)^2 + (e_1)^2\right] - 1 & 2(e_1e_2 - e_0e_3) & 2(e_1e_3 + e_0e_2) \\ 2(e_1e_2 + e_0e_3) & 2\left[(e_0)^2 + (e_2)^2\right] - 1 & 2(e_2e_3 - e_0e_1) \\ 2(e_1e_3 - e_0e_2) & 2(e_2e_3 + e_0e_1) & 2\left[(e_0)^2 + (e_3)^2\right] - 1 \end{bmatrix} \quad (8)$$

Rotation of a point $\mathbf{p}_a$ into $\mathbf{p}_b$ by a quaternion $\mathbf{q}_{a,b}$ can be written (Watt, 1995) with double Grassman products, where we introduce the *pure-imaginary* quaternions $\underline{\mathbf{p}}=(0,\mathbf{p})$, that is:

$$\mathbf{p}_b = \left[\Lambda(\mathbf{q}_{b,a})\right]\mathbf{p}_a \quad \rightarrow \quad \underline{\mathbf{p}_b} = \mathbf{q}_{b,a}\,\underline{\mathbf{p}_a}\,\mathbf{q}^*_{b,a} \tag{9a}$$

Since rotation quaternions are unimodular, remembering eq.4 and eq.5, we get the inverse of eq.9a by conjugating the rotation quaternions:

$$\underline{\mathbf{p}_a} = \mathbf{q}^*_{b,a}\,\underline{\mathbf{p}_a}\,\mathbf{q}_{b,a} \tag{9b}$$

An easy relation between angular speed (as vector $\boldsymbol{\omega}_l$ in frame local coordinate system) and the time-derivative of rotation quaternion $d\mathbf{q}/dt$ can be easily deduced (Wehage, 1984, Shoemake, 1995), again by using pure quaternions $\underline{\mathbf{w}}_l = (0, \boldsymbol{\omega}_l)$:

$$\dot{\mathbf{q}}_{l,w} = \tfrac{1}{2}\mathbf{q}_{l,w}\,\underline{\mathbf{w}^*_l} \tag{10}$$

For angular speed $\boldsymbol{\omega}_w$, in world coordinate system, we use eq.9b to write:

$$\underline{\mathbf{w}_l} = \mathbf{q}^*_{l,w}\,\underline{\mathbf{w}_w}\,\mathbf{q}_{l,w} \tag{11}$$

then, substituting eq.11 into eq.10, and observing that, for unimodular quaternions, we can simplify and cut away $\mathbf{q}^*\mathbf{q} = \mathbf{q}\,\mathbf{q}^* = \mathbf{q}_{\Re e}$, we get:

$$\dot{\mathbf{q}}_{l,w} = \tfrac{1}{2}\underline{\mathbf{w}^*_w}\,\mathbf{q}_{l,w} \tag{12}$$

Also, we can time-differentiate eq.10 and eq.12 once in order to obtain the relation between acceleration-quaternions and angular acceleration vectors ($\boldsymbol{\alpha}_l$ and $\boldsymbol{\alpha}_w$, in local or world-reference):

$$\ddot{\mathbf{q}}_{l,w} = \tfrac{1}{2}\mathbf{q}_{l,w}\,\underline{\boldsymbol{\alpha}^*_l} + \tfrac{1}{2}\dot{\mathbf{q}}_{l,w}\,\underline{\mathbf{w}^*_l} \tag{13}$$

$$\ddot{\mathbf{q}}_{l,w} = \tfrac{1}{2}\underline{\boldsymbol{\alpha}^*_w}\,\mathbf{q}_{l,w} + \tfrac{1}{2}\underline{\mathbf{w}^*_w}\,\dot{\mathbf{q}}_{l,w} \tag{14}$$

As well as we got the formulas to get time-derived quaternions as functions of angular speed and angular accelerations, we can get also the inverse relations. In detail, by pre-multiplication of eq.10 by $\mathbf{q}^*_{l,w}$ or by post-multiplication of eq.12 by $\mathbf{q}_{l,w}$, remembering that $\mathbf{q}^*\mathbf{q} = \mathbf{q}\,\mathbf{q}^* = \mathbf{q}_{\Re e}$ can be simplified in multiplications, and recalling def.(4), we get:

$$\underline{\mathbf{w}^*_l} = 2\,\mathbf{q}^*_{l,w}\dot{\mathbf{q}}_{l,w} \qquad \underline{\mathbf{w}_l} = -2\,\mathbf{q}^*_{l,w}\dot{\mathbf{q}}_{l,w} \tag{15}$$

$$\underline{\mathbf{w}^*_w} = 2\,\dot{\mathbf{q}}_{l,w}\mathbf{q}^*_{l,w} \qquad \underline{\mathbf{w}_w} = -2\,\dot{\mathbf{q}}_{l,w}\mathbf{q}^*_{l,w} \tag{16}$$

By differentiation of eq.15 and 16, we get also the inverse relations for angular accelerations:

$$\underline{\boldsymbol{\alpha}_l} = -2(\mathbf{q}^*_{l,w}\ddot{\mathbf{q}}_{l,w} + \dot{\mathbf{q}}^*_{l,w}\dot{\mathbf{q}}_{l,w}) \tag{17}$$

$$\underline{\boldsymbol{\alpha}_w} = -2(\ddot{\mathbf{q}}_{l,w}\mathbf{q}^*_{l,w} + \dot{\mathbf{q}}_{l,w}\dot{\mathbf{q}}^*_{l,w}) \tag{18}$$

Note that equations 15,16,17,18 can be written also with linear algebra (by the way this would help to rearrange the formalisms in order to avoid some unnecessary computations, since the real part of quaternions $\underline{\mathbf{w}}$ and $\underline{\boldsymbol{\alpha}}$ is always null).

In fact, recalling the matrix-form of the Grassman product (eq.6 and 7), we can use two 3x4 non-square matrices $[G(\mathbf{q})]$ defined as:

$$\left[G_l(q)\right] = 2\begin{bmatrix} -e_1 & +e_0 & +e_3 & -e_2 \\ -e_2 & -e_3 & +e_0 & +e_1 \\ -e_3 & +e_2 & -e_1 & +e_0 \end{bmatrix} \tag{19}$$

$$\left[G_w(q)\right] = 2\begin{bmatrix} -e_1 & +e_0 & -e_3 & +e_2 \\ -e_2 & +e_3 & +e_0 & -e_1 \\ -e_3 & -e_2 & +e_1 & +e_0 \end{bmatrix} \tag{20}$$

in order to rewrite our kinematic relations for angular speeds (eq.10,12,15,16) as:

$$\boldsymbol{\omega}_l = \left[G_l(\mathbf{q}_{l,w})\right]\dot{\mathbf{q}}_{l,w} \tag{21}$$

$$\boldsymbol{\omega}_w = \left[G_w(\mathbf{q}_{l,w})\right]\dot{\mathbf{q}}_{l,w} \tag{22}$$

$$\dot{\mathbf{q}}_{l,w} = \tfrac{1}{4}\left[G_l(\mathbf{q}_{l,w})\right]^T\,\boldsymbol{\omega}_l \tag{21}$$

$$\dot{\mathbf{q}}_{l,w} = \tfrac{1}{4}\left[G_w(\mathbf{q}_{l,w})\right]^T\boldsymbol{\omega}_w \tag{22}$$

For angular accelerations, either by differentiation of the above equations, or by manipulating eq.13,14,17,18, we get the same results:

$$\boldsymbol{\alpha}_l = \left[G_l(\mathbf{q}_{l,w})\right]\ddot{\mathbf{q}}_{l,w} + \left[G_l(\dot{\mathbf{q}}_{l,w})\right]\dot{\mathbf{q}}_{l,w} \tag{23}$$

$$\boldsymbol{\alpha}_w = \left[G_w(\mathbf{q}_{l,w})\right]\ddot{\mathbf{q}}_{l,w} + \left[G_w(\dot{\mathbf{q}}_{l,w})\right]\dot{\mathbf{q}}_{l,w} \tag{24}$$

$$\ddot{\mathbf{q}}_{l,w} = \tfrac{1}{4}\left[G_l(\mathbf{q}_{l,w})\right]^T\boldsymbol{\alpha}_l + \tfrac{1}{4}\left[G_l(\dot{\mathbf{q}}_{l,w})\right]^T\boldsymbol{\omega}_l \tag{25}$$

$$\ddot{\mathbf{q}}_{l,w} = \tfrac{1}{4}\left[G_w(\mathbf{q}_{l,w})\right]^T\boldsymbol{\alpha}_w + \tfrac{1}{4}\left[G_w(\dot{\mathbf{q}}_{l,w})\right]^T\boldsymbol{\omega}_w \tag{26}$$

Interesting enough, we observed that, under the assumption of unimodularity of quaternions, some terms of these equations vanish, in particular the second addenda of eq.23 and eq.24. These become simply $\boldsymbol{\alpha}_l=[G_l]\,\ddot{\mathbf{q}}$ and $\boldsymbol{\alpha}_w=[G_w]\,\ddot{\mathbf{q}}$. In a similar fashion, also the second addenda of eq.17 and eq.18 can be neglected, thus simplifying into $\underline{\boldsymbol{\alpha}_l} = -2\,\mathbf{q}^*_{l,w}\ddot{\mathbf{q}}_{l,w}$ and $\underline{\boldsymbol{\alpha}_w} = -2\,\ddot{\mathbf{q}}_{l,w}\mathbf{q}^*_{l,w}$.

3. Lagrangian system dynamics

Let introduce the vector of coordinates **x**, built from all the coordinates of rigid bodies:

$$\mathbf{x} = \left\{ \mathbf{x}_{(1)}^{\mathrm{T}}, \ \dots \ , \mathbf{x}_{(n)}^{\mathrm{T}} \right\}^{\mathrm{T}}$$

where $\mathbf{x}_{(i)}=\{\mathbf{p}_{(i)}{}^{\mathrm{T}},\mathbf{q}_{(i)}{}^{\mathrm{T}}\}^{\mathrm{T}}$ is the coordinate of the *i*-th body, that is carthesian position **p** and rotation quaternion **q**.

Using this maximal set of variables, the integrals of system's equations of motion must reside on the constraint manifolds, hence posing a *differential-algebraic* (DAE) problem of the type:

$$\begin{cases} \left\{ \dfrac{\mathrm{d}}{\mathrm{dt}} \left[\dfrac{\partial \mathrm{E_c}}{\partial \dot{\mathbf{x}}} \right] \right\}^{\mathrm{T}} - \left[\dfrac{\partial \mathrm{E_c}}{\partial \mathbf{x}} \right]^{\mathrm{T}} + [\mathrm{C_x}]^{\mathrm{T}} \boldsymbol{\lambda} = \hat{\mathbf{Q}} \\ \mathbf{C}(\mathbf{x}, \mathrm{t}) = \mathbf{0} \end{cases} \tag{27}$$

where Lagrange's equations (with multipliers **λ**) are coupled to constraint equations **C**=**0**, functions of time and coordinates **x**. Since DAE-oriented implicit solvers may be slow and complex, a common way to solve eq.27 is to reduce it to an explicit ODE (*ordinary differential*) system of the type:

$$\begin{bmatrix} [\mathrm{M}] & [\mathrm{C_x}]^{\mathrm{T}} \\ [\mathrm{C_x}] & [0] \end{bmatrix} \cdot \begin{Bmatrix} \ddot{\mathbf{x}} \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} \hat{\mathbf{Q}} + \mathbf{Q}_{\mathrm{m}} \\ \mathbf{Q}_{\mathrm{c}} \end{Bmatrix} \tag{28}$$

where constraint equations are inserted in eq.28 in their time-differentiated form (eq.31):

$$\mathbf{C} = \mathbf{C}(\mathbf{x}, \mathrm{t}) = \mathbf{0} \tag{29a}$$

$$\dot{\mathbf{C}} = [\mathrm{C_x}]\dot{\mathbf{x}} + \mathbf{C}_{\mathrm{t}} = \mathbf{0} \tag{29b}$$

$$\ddot{\mathbf{C}} = [\mathrm{C_x}]\ddot{\mathbf{x}} - \mathbf{Q}_{\mathrm{c}} = \mathbf{0} \tag{29c}$$

It is important to remark that some terms of eq.29a,29b,29c usually entail complex calculations, expecially the jacobian $[\mathrm{C_x}]$ and the $\mathbf{Q}_{\mathrm{c}}$ vector which are needed for eq.28, but we have recently developed a compact and efficient method, called *lock formulation*, which allows a fast and unified way to compute these jacobians for a wide class of holonomic and rheonomic constraints, thank to quaternion algebra (Tasora,1999).

Note that straightforward integration of accelerations from eq.28 generally doesn't suffice, since only the differentiated form of constraint equations eq.29c is satisfied, and this doesn't guarantee also eq.29a and 29b to be true as time integration goes on: this means that some constraint drifting in position and speed may take place. Then, to avoid this side effect, fast 'Baumgarte' constraint stabilization techniques could be used (Bae,1991).

However we experienced the best results with an exact elimination of constraint violation like in the scheme recently suggested by W.Blajer (Blajer,1997).

In detail, after each integration step relying on accelerations of eq.28, we correct the system's state by moving positions and velocities back to their manifolds, **C** and $\dot{\mathbf{C}}$ respectively, by means of small corrections which are orthogonal to the manifolds.

The correction of position constraint violation must be performed first, and may be repeated iteratively since **C** is non linear in **x** (but often just one or two iterations are sufficient to drive the correction Δ**x** within tolerance). We worked out what can be seen as a 'custom' Newton Raphson solution of eq.29a:

$$\begin{bmatrix} [\mathrm{M}] & [\mathrm{C_x}]^{\mathrm{T}} \\ [\mathrm{C_x}] & [0] \end{bmatrix} \cdot \begin{Bmatrix} \Delta\mathbf{x}^{(\mathrm{j})} \\ \boldsymbol{\tau} \end{Bmatrix} = \begin{Bmatrix} 0 \\ -\mathbf{C}^{(\mathrm{j})} \end{Bmatrix} \tag{30}$$

$$\mathbf{x}^{(\mathrm{j+1})} = \mathbf{x}^{(\mathrm{j})} + \Delta\mathbf{x}^{(\mathrm{j})}$$

One can see that the Δ**x** obtained above is the only one that, among the infinite which can satisfy eq.29a, also minimizes $\{\tfrac{1}{2}\Delta\mathbf{x}^{\mathrm{T}}[\mathrm{M}]\Delta\mathbf{x} + \Delta\mathbf{x}[\mathrm{C_x}]^{\mathrm{T}}\boldsymbol{\tau}\}$.

The correction of speed constraint-violation can be faced in the following way, where no iterations are necessary:

$$\begin{bmatrix} [\mathrm{M}] & [\mathrm{C_x}]^{\mathrm{T}} \\ [\mathrm{C_x}] & [0] \end{bmatrix} \cdot \begin{Bmatrix} \Delta\dot{\mathbf{x}} \\ \boldsymbol{\mu} \end{Bmatrix} = \begin{Bmatrix} 0 \\ -\dot{\mathbf{C}}^{(\mathrm{old})} \end{Bmatrix} \tag{31}$$

$$\dot{\mathbf{x}}^{(\mathrm{new})} = \dot{\mathbf{x}}^{(\mathrm{old})} + \Delta\dot{\mathbf{x}}$$

Note that eq.31 can be see as a Newton Raphson solution of eq.29b in a single step. This speed correction can be seen as the one which, among the infinite which respect eq.29b, also minimizes $\{\tfrac{1}{2}\Delta\dot{\mathbf{x}}^{\mathrm{T}}[\mathrm{M}]\Delta\dot{\mathbf{x}} + \Delta\dot{\mathbf{x}}[\mathrm{C_x}]^{\mathrm{T}}\boldsymbol{\mu}\}$.

We stress a remarkable fact: few algebraic manipulations reveal that eq.30 and eq.31 are completely equivalent to the formulation of Blajer, therefore the same considerations can be applied, in particular the fact that Δ**x** and Δ$\dot{\mathbf{x}}$ corrections obtained in this way do not change the positions and the speeds of the system in the *null space* of $\mathbf{C}^{\mathbf{m}}$, that is the *n*-dimensional subspace $\mathbf{D}^{\mathbf{n}}$ which complements $\mathbf{C}^{\mathbf{m}}$ in $\mathbf{E}^{\mathbf{s}}$, i.e. $\mathbf{C}^{\mathbf{n}} \cap \mathbf{D}^{\mathbf{m}} = \mathbf{0}$ $\mathbf{C}^{\mathbf{m}} \cup \mathbf{D}^{\mathbf{m}} = \mathbf{E}^{\mathbf{s}}$ for *m* constraints in a system of *s* coordinates (Blajer,1997).

Despite the more compact formulas developed by Blajer do not introduce auxiliary multipliers **τ** and **μ** as in systems eq.30 and eq.31, in our scheme the coefficient matrices of eq.28, eq.30 and eq.31 are the same, and their high degree of sparsity suggests a

custom and fast way of solving the linear systems.

4. System factorization

The linear systems of eq.28, 30 and 31 must be solved many times during integration: in case of many coordinates the solution time can be the bottleneck of the entire simulation code, since required time for direct solution schemes is generally $O(n^3)$, for n variables. Since our method is based on lagrangian multipliers with a maximal set of variables, the value of n can be high, and the only way to achieve fast performances as in reduced coordinate methods consists in implementing $O(n^h)$ solution schemes, with h tending to unity: this is possible if one exploits the extreme sparsity of these matrices.

4.1 Matrix structure

The coefficient matrix in eq.28, 30 and 31, referenced as $[A_{mc}]$ from now on, has the structure of fig.1 (for a simple example with 3 bodies and two constraints A and B, the former between body 1 and 2, the second between 2 and 3).

Note that the mass matrix is diagonal-dominant (being built from masses and 4x4 inertia tensors $[M_{qq}]$ in quaternion coordinates), and the jacobian $[C_x]$ is built from blocks corresponding to single joints. Also, the jacobian has additional k rows for k bodies, since the auxiliary constraints on unimodularity of quaternions must be taken into account (the condition $e_0^2+e_1^2+e_2^2+e_3^2=0$ is differentiated once and twice in time, so C_t, Q_c and 1x4 jacobians $[C_x]_{quat\ n}$ are obtained just like for all other constraints $\mathbf{C}(\mathbf{x},t)$, as in eq.29a, 29b, 29c).

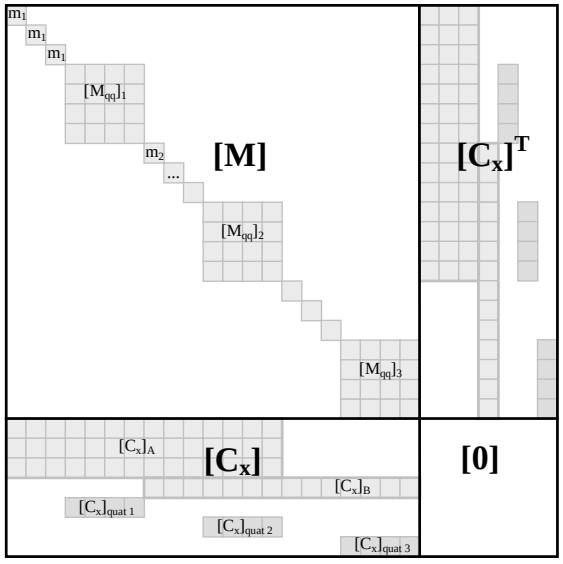

Fig.1: Structure of system matrix $[A_{mc}]$ (example)

4.2 Projection in α-ω space

The solution of linear systems of the type $[A_{mc}]x=b$ as in eq.28,30,31 can be faced by Gauss forward-backward scheme, or by LU decomposition. However, these standard methods are acceptable only for few variables, since their complexity is strictly $O(n^3)$.

A little improvement would consist in using LU or Gauss method modified for sparse matrices. Yet this poses a big problem: row pivoting destroys symmetry, hence deteriorates sparsity and almost nullify this advantage after few row reductions.

Therefore, one could consider a sparse Cholesky decomposition LL^T, which retains symmetry. But this won't work, since $[A_{mc}]$ is not always positive definite (because of jacobian blocks), even if well conditioned.

The only solution would be the adoption of a LDL^T decomposition method, which is similar to Cholesky, but does not require $[A_{mc}]$ to be positive definite (no square roots are performed).

However, straight LDL^T decomposition of $[A_{mc}]$ would soon or later run into a null pivot, since the diagonal 4x4 block matrices $[M_{qq}]_i$ have rank-3 (coming from the projection of the inertia tensors $[J_{\omega\omega}]_i$ in 4-dimensional quaternion space, as $[M_{qq}]_i=[G_l]_i^T[J_{\omega\omega}]_i[G_l]_i$). Then, diagonal pivoting could take place, but we found that diagonal pivoting for this kind of $[A_{mc}]$ matrix often leads to a bad conditioning of the numerical process.

Finally we found that the best way to go is to use LDL^T factorization on a subspace-form of the $[A_{mc}]$ matrix: in detail, reducing the rotation coordinates from 4 (the quaternions components, constrained by unimodularity) to 3 unconstrained, which would cause 3x3 full-rank blocks on the diagonal, and would also avoid the small additional $[C_x]_{quat}$ jacobians.

Here comes the use of the quaternion formulae we developed heretofore. We can see that, under unimodularity of $\mathbf{q}$ in eq.19, we have:

$$\tfrac{1}{4}\left[G_l(\mathbf{q})\right]^T\left[G_l(\mathbf{q})\right]=\left[I\right] \qquad (32)$$

then we can pre-multiply all quaternions of vector $\ddot{\mathbf{x}}$ in eq.28 by the quantity $¼[G_l]^T[G_l]$.

Then, remembering that, by eq.23, $\boldsymbol{\alpha}_l=[G_l]\,\dot{\mathbf{q}}$, we can write eq.28 in the form which uses $\boldsymbol{\alpha}$ instead of $\ddot{\mathbf{q}}$ as unknowns:

$$\begin{bmatrix} [M] & [C_x]^T \\ [C_x] & [0] \end{bmatrix} \begin{bmatrix} [T_q] & \\ & [I] \end{bmatrix} . \begin{Bmatrix} \ddot{\mathbf{x}}_{\alpha} \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} \hat{\mathbf{Q}} + \mathbf{Q}_m \\ \mathbf{Q}_c \end{Bmatrix} \tag{33}$$

where $\ddot{\mathbf{x}}_{\alpha} = \{\ddot{\mathbf{p}}_{(1)}, \boldsymbol{\alpha}_{(1)}, ..., \ddot{\mathbf{p}}_{(n)}, \boldsymbol{\alpha}_{(n)}\}$ and

$$[T_q] = \begin{bmatrix} [I] & & & \\ & \frac{1}{4}[G_1]^T_{(1)} & & \\ & & ... & \\ & & & [I] \\ & & & & \frac{1}{4}[G_1]^T_{(n)} \end{bmatrix} \tag{34}$$

To keep $[A_{mc}]$ symmetric and square, one can pre-multiply both terms of eq.33 by the matrix $[T_q]^T$, thus obtaining:

$$\begin{bmatrix} [T_q]^T & \\ & [I] \end{bmatrix} \begin{bmatrix} [M] & [C_x]^T \\ [C_x] & [0] \end{bmatrix} \begin{bmatrix} [T_q] & \\ & [I] \end{bmatrix} . \begin{Bmatrix} \ddot{\mathbf{x}}_{\alpha} \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{bmatrix} [T_q]^T & \\ & [I] \end{bmatrix} \begin{Bmatrix} \hat{\mathbf{Q}} + \mathbf{Q}_m \\ \mathbf{Q}_c \end{Bmatrix} \tag{35}$$

From eq.35 the matrix of coefficients, now called $[A_{mc}]_{\alpha}$, becomes smaller than $[A_{mc}]$ and looses the $[C_x]_{quat}$ blocks, see fig.2.

This reworked version of the coefficient matrix works very well with sparse LDL^T factorization, then one can efficiently solve system of eq.35 for $\ddot{\mathbf{x}}_{\alpha}$. Later, the vector $\ddot{\mathbf{x}}_{\alpha}$ (with rotation coordinates in α-space) can be transformed in the original vector $\ddot{\mathbf{x}}$, which uses quaternions as rotation coordinates, simply by applying eq.25 for each body:

$$\ddot{\mathbf{q}}_{l,w(i)} = \tfrac{1}{4}\left[G_l(\mathbf{q}_{l,w(i)})\right]^T \boldsymbol{\alpha}_{l(i)} + \tfrac{1}{4}\left[G_l(\dot{\mathbf{q}}_{l,w(i)})\right]^T \boldsymbol{\omega}_{l(i)}$$

An interesting remark about the transformation of eq.35: efficient multiplication with the $[T_q]$ matrix can exploit sparsity. That is, one just has to post-multiply some parts of jacobians $[C_x]$ by matrices $¼[G_l]^T$, and use $[J_{\omega\omega}]$ instead of $[M_{qq}]$.

In fact in eq.35 the inertia matrices are transformed as $¼[G_l][M_{qq}][G_l]^T¼$, that is $¼[G_l][G_l]^T[J_{\omega\omega}][G_l][G_l]^T¼ = [J_{\omega\omega}]$, so these simplify into 3x3 inertia (constant) tensors as in Newton-Euler equations.

Note also that there is an optimal ordering of constraint equations in eq.35 (the one which produces the less number of 'fill-ins' while decomposing matrix $[A_{mc}]_{\alpha}$). See preordering methods in (Duff 1996).

Similarly to eq.35, also eq.30 can be projected in different coordinates. In detail, using the same method, equation 30 is transformed into the following system, where $\Delta\mathbf{x}_{\omega} = \{\Delta\mathbf{p}_{(1)}, \Delta\boldsymbol{\beta}_{(1)}, ..., \Delta\mathbf{p}_{(n)}, \Delta\boldsymbol{\beta}_{(n)}\}$:

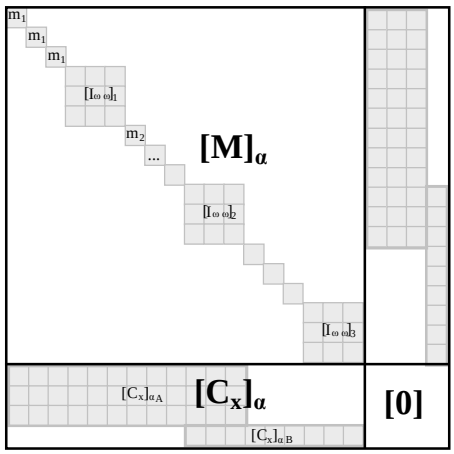

Fig.2: Structure of system matrix $[A_{mc}]_{\alpha}$ (example)

$$\begin{bmatrix} [T_q]^T & \\ & [I] \end{bmatrix} \begin{bmatrix} [M] & [C_x]^T \\ [C_x] & [0] \end{bmatrix} \begin{bmatrix} [T_q] & \\ & [I] \end{bmatrix} . \begin{Bmatrix} \Delta\mathbf{x}_{\omega}^{(j)} \\ \boldsymbol{\tau} \end{Bmatrix} = \begin{bmatrix} [T_q]^T & \\ & [I] \end{bmatrix} \begin{Bmatrix} 0 \\ -\mathbf{C}^{(j)} \end{Bmatrix} \tag{36}$$

It is easy to see that, after solving eq.36, one can recover $\Delta\mathbf{x}$ from $\Delta\mathbf{x}_{\omega}$ simply using, for each body:

$$\Delta\mathbf{q}_{l,w(i)} = \tfrac{1}{4}\left[G_l(\mathbf{q}_{l,w(i)})\right]^T \Delta\boldsymbol{\beta}_{l(i)} \tag{36b}$$

In a like manner, from eq.31 one gets the following system, where $\Delta\mathbf{x}_{\omega} = \{\Delta\mathbf{p}_{(1)}, \Delta\boldsymbol{\omega}_{(1)}, ..., \Delta\mathbf{p}_{(n)}, \Delta\boldsymbol{\omega}_{(n)}\}$:

$$\begin{bmatrix} [T_q]^T & \\ & [I] \end{bmatrix} \begin{bmatrix} [M] & [C_x]^T \\ [C_x] & [0] \end{bmatrix} \begin{bmatrix} [T_q] & \\ & [I] \end{bmatrix} . \begin{Bmatrix} \Delta\dot{\mathbf{x}}_{\omega} \\ \boldsymbol{\mu} \end{Bmatrix} = \begin{bmatrix} [T_q]^T & \\ & [I] \end{bmatrix} \begin{Bmatrix} 0 \\ -\dot{\mathbf{C}}_{old} \end{Bmatrix} \tag{37}$$

Again, after solving the linear system 37, one gets the original $\Delta\dot{\mathbf{x}}$ from $\Delta\dot{\mathbf{x}}_{\omega}$ simply applying eq.21 for each body:

$$\Delta\dot{\mathbf{q}}_{l,w(i)} = \tfrac{1}{4}\left[G_l(\mathbf{q}_{l,w(i)})\right]^T \Delta\boldsymbol{\omega}_{l(i)} \tag{37b}$$

Note that this transition to quaternions to three rotation coordinates happens only for the solution of systems 28,30,31, and it *not* equivalent to simply adopting from the beginning three angles (ex. HPB or Cardano angles) as body coordinates, since this would *not* offer all advantages of quaternions in terms of the advanced algebra, the singularity-avoidance and, most important, the possibility of adopting our *lock-formulation* for fast computation of constraints.

5. Sparse LDL^T factorization

Systems 35, 36 and 37 must be solved many times during integration. However, some efforts can be saved because the coefficient matrices are the same, that is $[A_{mc}]_{\alpha}$ can be factorized in LDL^T form just for the solution of system of eq.35, while only the backward substitution is performed for eq.36 and 37 since known terms change, but $[A_{mc}]_{\alpha}$ won't change (someone may argue that eq.36 is an iterative N-R solution scheme, where also $[A_{mc}]_{\alpha}$ may change a bit, but we observed that usually one or two iterations are sufficient to correct small residuals, and $[A_{mc}]_{\alpha}$ can be freely considered constant).

In order to perform an efficient LDL^T factorization of the $[A_{mc}]_{\alpha}$ matrix, one must use a *sparse matrix representation*.

We found an efficient storage method by using linked lists of *non-structural-zeros* elements, one for each row of the matrix as in figure 3 (the lower part isn't stored since symmetry is assumed).

The LDL^T algorithm has been rewritten, taking advantage of this linked list representation. Forward decomposition works "in place" (L^T is written over the upper part of $[A_{mc}]_{\alpha}$, D is written on the diagonal of $[A_{mc}]_{\alpha}$, lower L is not written, L^T diagonal is assumed unitary)

Here's the pseudocode for the LDL^T factorization.

```
for (k=1; k < rows; k++)
  pivot = GetElement((k-1),(k-1));
  [handle diagonal pivoting here, if needed..]
  [if null pivot anyway, force pivot to 10e34]
  for (i=k; i < rows; i++)
    leader = GetElement((k-1), i);
    if (leader)
      r= (leader / pivot);
        sub = GetFirstRowElement(i);
        row = GetFirstRowElement(k-1);
        for (NULL;row!= NULL;row=row->next)
          if (row->col >= i)
            mval = row->val;
            sub=GetOptEl(i, row->col, &subval, sub);
            newval = subval - r * mval;
               sub=SetOptEl(i, row->col, newval, sub);
          SetElement((k-1),i, r);
```

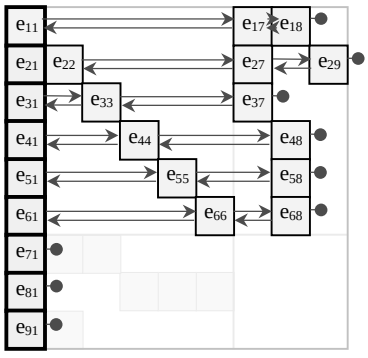

Fig.3: sparse storage of matrix $[A_{mc}]_{\alpha}$ (small example)

Note: in the pseudocode above, most fetch/store is done via the speed-optimized functions `GetOptEl()` and `SetOptEl()` which use a 'guess' for addressing the element in linked lists, with the following syntax:

```
[elem*]suggested_next_guess= GetOptEl
       (row,col,[double*]fetched_value,[elem*]guess);
[elem*]suggested_next_guess= SetOptEl
       (row,col,[double*]stored_value,[elem*]guess);
```

For backward substitution, given the factorization $[A_{mc}]_{\alpha}=[L][D][L]^T$ and a vector of known terms **b**, one can solve $[A_{mc}]_{\alpha}\,\mathbf{x} = \mathbf{b}$ with three easy steps: $\mathbf{u}=[L]^{-1}\mathbf{b}$, $\mathbf{v}=[D]^{-1}\mathbf{u}$, $\mathbf{x}=([L]^T)^{-1}\mathbf{v}$.

This is done with the following (unoptimized) pseudocode, where `x` is the unknown vector, and `pivarray[]` an array of pivots, if diagonal pivot occurred in factorization:

```
for (k=1; k<rows; k++)        // BACKWARD substitution - L
    sum = 0;
    for (j=0; j<k; j++)
      sum+=(GetElement(j,k))*(X->GetElement(pivarray[j],0));
    x = (B->GetElement(pivarray[k],0) - sum);
    X->SetElement(pivarray[k],0, x);
for (k=0; k<rows; k++)        // BACKWARD substitution - D
    x = X->GetElement(pivarray[k],0) / GetElement(k,k);
    X->SetElement(pivarray[k],0,x);
for (k=(rows-2); k>= 0; k--) // BACKWARD substitution - L'
    sum = 0;
    for (j=(k+1); j< rows; j++)
      sum+=(GetElement(k,j))*(X->GetElement(pivarray[j],0));
    x = (X->GetElement(pivarray[k],0) - sum);
    X->SetElement(pivarray[k],0, x);
```

Mechanisms with ill-conditioned constraints and redundant joints may cause over-constrained systems: this often happens in devices with intermitting contacts, impacts and variable topology in general.. In these situations the factorization scheme cannot avoid zero pivots during forward LDL^T decomposition (after all possible pivotings). Then, we found that a straightforward artifice is to force that pivot to a very high value (ex.10^{34}) and then proceed as usual with decomposition. In this way, the corresponding lagrangian multiplier will be nearly zero when doing the backward substitution, and its equation won't affect anything (later, ill placed constraints can be removed).

6. Examples

As examples of application of the solution scheme, we propose the virtual prototyping of a prosthetic arm (fig.4) and the dynamical simulation of a man which rides a bike on a uneven terrain (fig.5), for hazard estimation.

The prosthetic arm has been modeled with 7 rigid bodies and 8 links, for a total of 82 variables (42

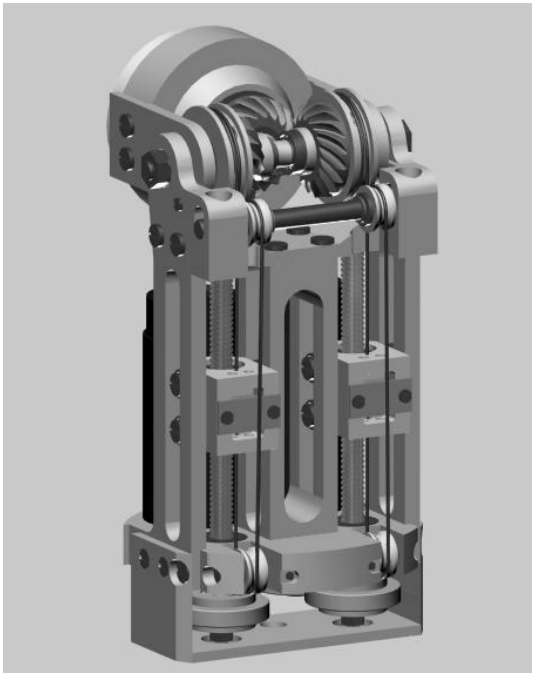

Fig.4: Prototype of prosthetic arm (detail)

coordinates, 40 scalar constraints). The designer can test the kinematics and the dynamics of the device by mouse interaction over the three dimensional model, since the simulation runs fast enough to allow real-time rates (more than 150 dynamical simulation steps per second, on a 700 MHz AMD CPU, which means about 6 ms per simulation step).

Such prosthetic arm is built on the principle of the differential gear: two DC motors are mounted into the arm shield and rotate the two side gears which are coaxial to the second axis of the Cardano's suspension of the shoulder (since the frontal pinion is locked to the breast, this causes a 2-DOF spherical motion).

The transmission of the torques passes from motors to differential gears by means of two ball-screw reducers and two pulleys with iron wires, for reasons of compactness, efficiency and noise. Otherwise, worm-screw reducers can be used, as in fig.4.

All these components are taken into account during both the dynamical and kinematic analysis.

The simulation of the multibody model can be a valuable help for studying which torques are needed for given motion tasks, hence assisting the choice of the proper transmission and motor components. In detail, specific motions can be assigned to the wrist of the complete arm (such as: 'take a book', 'lift a bottle',

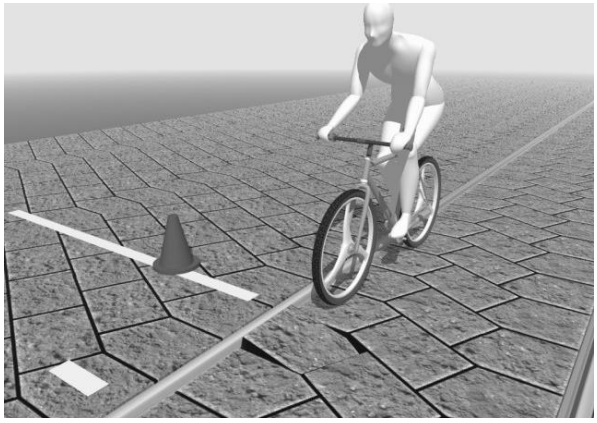

Fig.5:Simulation of a bike on uneven terrain

specified as cartesian Nurbs envelopes) and the graphs of the torques for the two shoulder motors can be suddenly obtained. This is useful to test different features and components, in order to find the best tradeoff between speed, cost, weight, efficienty and noise.

Of course, this multibody simulation can be extended also to the study of the complete robotized arm (motor torques and joint reactions can be obtained for all the parts of the device, including elbow and robotized hand/wrist). Given to the enhanced complexity of this 9-body full model, with elbow and wrist, the system DAE has 104 variables (54 coordinates, 50 scalar constraints) and the simulation speed is a bit lower: 8-9 ms of CPU time per step.

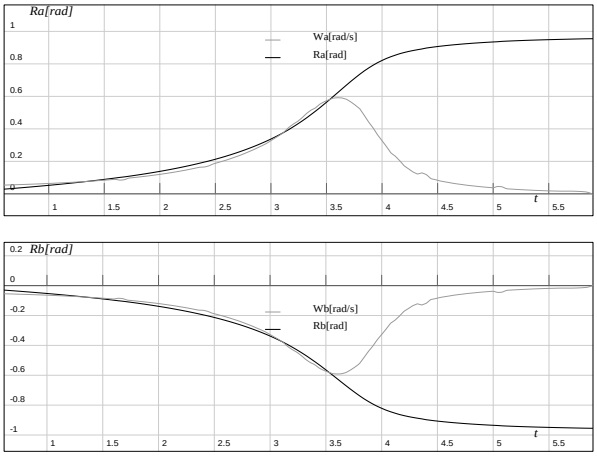

Fig.6: Example: rotation and angular speed of the two gears of the prosthetic arm, for a slow and rectilinear motion of the hand in cartesian space.

It is interesting to remark that this kind of prosthetic shoulder cannot be controlled simply by letting the user switch on-off the two engines with mio-electric sensors, since the effect would be highly impredictable and even trivial trajectories could require a cooperative and coordinated motion of both the engines, moreover with non-linear laws. However this complexity can be overridden by a real-time controller which lets the user choose the motion of the hand in plain cartesian space (that is in an intuitive way, like 'upper', 'lower', 'more on the left', 'more on the right', 'nearer', 'farther') while the inverse kinematics is automatically performed to remap the motion in the joint space of the motors.

A prototype of this robotized prosthetic shoulder is under development in our department, and will be soon assembled and tested in real environments.

The dynamical simulation of the man riding the bike consists of a very complex multibody model: 17 bodies, 18 links, for a total of 200 variables (102 coordinates, 92 scalar constraints). Note that, despite the high dimensionality of the problem, the dynamic solution runs so fast that the true bottleneck is rather the visualization and the tire model, not the multibody.

The man model consists of a simplified skeleton of rigid bodies (anthropometric data comes from the Winter man model), and the bike model is based on a standard commercial "city bike", with 28-622 700 x 28C tires. The mass moments of inertia of the parts have been obtained from Gauss quadrature of the B-rep geometry, in the 3D modeler environment.

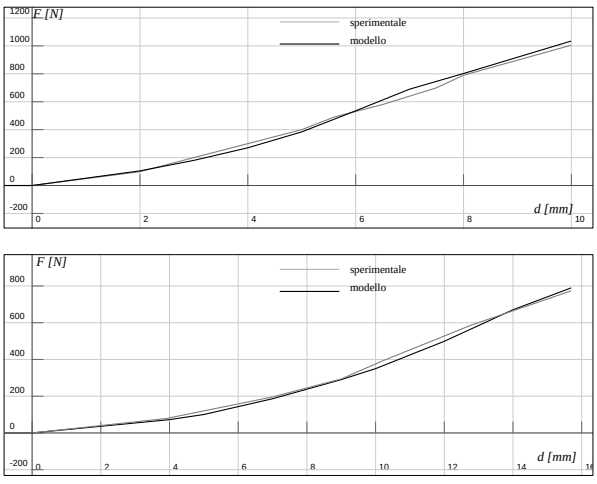

Fig. 6: Experimental and simulated vertical tire stiffness, for city bike (above) and low-inflated mountain bike (below).

A three dimensional deformable tire model has been implemented (points of contact with the ground are sampled with 3-6 mm resolution via a ray-tracing algorithm, with octree optimization), and experimental tests have been performed in order to tune and validate this model, as can be seen in fig.6.

Also, we accomplished some experimental tests to get the exact coefficient of friction for the specific road conditions of this simulation: main results are reported in table 1.

	Static friction	Kinetic friction
Asphalt – clean	0.97	0.86
Asphalt – wet	0.90	0.88
Stone – clean	0.68	0.65
Stone – wet	0.62	0.61
Steel rail – clean	0.45	0.42
Steel rail – wet, but clean	0.38	0.38
Steel rail – wet and dirty	<0.35	<0.34

Table 1 : Friction coefficients of the 28-622 bike tire.

Since the stiffness of the contact between seat and man plays a relevant role in the dynamics of the system, we also performed some experiments to measure the behavior of such non-linear stiffness. The graph of stiffness *k* as a function of downward motion of hips is reported in figure 7.

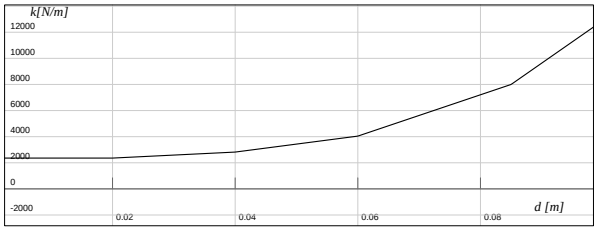

Fig. 7 : Vertical nonlinear stiffness of contact man-seat

Many simulations have been run (each lasting 3 seconds) in order to analyze the effects of the crossing of road obstacles with varying shapes. In detail, the obstacle was represented by a misplaced stone in the road pavement: our intention was to study which is the acceptable step height before injuries can occur, and how this height affects parameters such as vertical acceleration, vehicle displacement from optimal trajectory, reaction forces, and so on.

As a concluding remark, we observe that in simple mechanisms (for example, the prosthetic arm) the gain in computation speed is at least **2x..3x** respect to a

naive (full Gauss) solution of eq.28,30,31. Using custom sparse factorization on eq.35,36,37, the higher the number of bodies, the more gain in performance can be expected: in fact we experienced speedup factors of **20x**, and more, for systems with high dimensionality ($n_{variables}$>>100), like the case of the man on the bicycle.

In fact, table 2 shows that the computational effort grows almost linearly with the increasing of the number of variables in the differential-algebraic problem, i.e. $O(n^k)$ with $k \approx 1.0$, while a naive solution method would show a straight $O(n^3)$ dependence on complexity.

	DAE dimension (tot variables)	N coordinates	N constraints	Average steps per second	Average step CPU time [s]
Arm (shoulder mechanism only)	82	42	40	>150	0.006
Arm (complete, with forearm hand)	104	54	50	>125	0.008
20-pieces closed kinematic chain	180	120	60	67	0.015
40-pieces closed kinematic chain	360	240	120	24	0.042
Man with bike	200	108	92	56	0.018

Table 2 : Performance of the method (700 MHz cpu)

7. Conclusions

A fast solution scheme for the lagrangian-multiplier method of *n*-body mechanisms has been developed and successfully tested.

By applying specific formulas of quaternion algebra together with a custom factorization of sparse matrices, we obtained a very efficient method for the solution of differential-algebraic problems.

Such theoretic framework encouraged a compact way of handling problems like constraint stabilization, intermitting contacts, redundant joints and impacts, consequently allowing its application to complex problems of interactive dynamics.

References

A.Shabana, *Multibody systems*, John Wiley & Sons, New York 1989.

K.Shoemake, *Animating rotation curves*, in CG ACM, vol 19, 1995.

R.Roberson, R.Schwertassek, *Dynamics of multibody systems*, Springer Verlag, Berlin.

A.Watt, M.Watt, *Advanced animation and rendering techniques*, Addison Wesley, 1995

D.Baraff, *Linear-time dynamics using Lagrange multipliers*, Siggraph 96, New Orleans, 1996.

R.A.Wehage, *Quaternion and Euler parameters-A brief exposition*, in Computer Aided Analysis and Optimization of Mechanical Systems Dynamics, Springer Verlag, 1984.

A.Tasora, P.Righettini, *Application of quaternion algebra to the efficient computation of jacobians for holonomic-rheonomic constraints*, EUROMECH 404, Lisbon, September 1999.

D.Bae, S.Yang, *A stabilization method for kinematic and kinetic constraint equations*, Real-time integration methods for mechanical system simulation, NATO advanced research workshop, Utah, ed. Springer Verlag 1991.

W.Blajer, *A Geometric unification of Constrained System Dynamics*, Multibody System Dynamics vol.1, 3-21, 1997.

P. Amestoy, T. Davis, and I. Duff, *An approximate minimum degree ordering algorithm*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 886--905.