Mech. Sci., 4, 199–219, 2013
www.mech-sci.net/4/199/2013/
doi:10.5194/ms-4-199-2013

Mechanical Sciences
Open Access

ROBOTRAN: a powerful symbolic gnerator of multibody models

N. Docquier[1,*], A. Poncelet[1], and P. Fisette[1]

[1]Université catholique de Louvain, Center for Research in Mechatronics, B 1348 Louvain-la-Neuve, Belgium
[*]Postdoctoral Researcher of the Fonds de la Recherche Scientifique – FNRS

Correspondence to: N. Docquier (nicolas.docquier@uclouvain.be)

Received: 15 December 2012 – Revised: 19 March 2013 – Accepted: 14 April 2013 – Published: 2 May 2013

Abstract. The computational efficiency of symbolic generation was at the root of the emergence of symbolic multibody programs in the eighties. At present, it remains an attractive feature of it since the exponential increase in modern computer performances naturally provides the opportunity to investigate larger systems and more sophisticated models for which real-time computation is a real asset.

Nowadays, in the context of mechatronic multibody systems, another interesting feature of the symbolic approach appears when dealing with enlarged multibody models, i.e. including electrical actuators, hydraulic devices, pneumatic suspensions, etc. and requiring specific analyses like control and optimization. Indeed, since symbolic multibody programs clearly distinguish the modeling phase from the analysis process, extracting the symbolic model, as well as some precious ingredients like analytical sensitivities, in order to export it towards any suitable environment (for control or optimization purposes) is quite straightforward. Symbolic multibody model portability is thus very attractive for the analysis of mechatronic applications.

In this context, the main features and recent developments of the ROBOTRAN software developed at the Université catholique de Louvain (Belgium) are reviewed in this paper and illustrated via three multibody applications which highlight its capabilities for dealing with very large systems and coping with multiphysics issues.

1 Introduction

Before the appearance of efficient computer architectures for scientific numerical computations, only analytical methods were available for modeling systems. The analyses often used rather restrictive hypotheses (truncated and/or linearized models, for instance). The emergence of powerful processors and reliable and user-friendly languages and software led the scientific community to develop numerical programs able to cover a wide range of applications in a given field (e.g. structural dynamics, multibody dynamics, fluid mechanics, electronic circuits, . . .). Regarding multibody dynamics, numerous so-called multibody programs were developed all over the world as from the seventies (Schiehlen, 1990), each of them being described as a general purpose code although, in reality, faced with the huge variety of applications, they all impose some restrictions on the modeling and analysis processes.

1.1 System modeling versus system analysis

From our point of view, it is useful to distinguish modeling from analysis. The *modeling* phase is the analytical, numerical or symbolical process which, once and for all, sets up the equations of motion describing a multibody system (MBS in short) for a given set of system parameters and generalized coordinates. For instance, the direct dynamic model is illustrated in Fig. 1 as a block whose input are the system parameters δ (joints location, body masses, and inertia, etc.) and the generalized positions q and velocities $\dot{q}$ and whose outputs are the generalized accelerations $\ddot{q}$ and the Lagrange multipliers λ. The *analysis* phase denotes any numerical process which uses a *model* (or part of it) to generate the expected

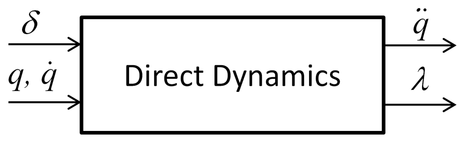


Figure 1. Direct Dynamic Model: main input/output.

results. Equilibrium solution, modal analysis, time integration, optimization are examples of analysis processes which, in case of complex MBS, must be performed numerically via specific algorithms. All of these numerical processes require a repeated evaluation of the model, often more than a million times in a time integration or optimization process. This means that, whatever the chosen formalism and the available computer resources, the computation of the model must be as efficient as possible.

1.2 Numerical versus symbolic implementation

The classical way of implementing these two phases, modeling and analysis, in a computer program is to do so in a purely numerical manner. Although this is mandatory for the analysis phase, it is not the only possibility for the modeling phase.

Before thinking about generating it automatically using a computer, the most natural way to model a system such as that of Fig. 1, is to do it *manually* – using a pen or some general purpose program – on the basis of a given formalism (Newton/Euler equations, Virtual Work Principle, etc.). While appearing to be archaic in the computer era, and not always reliable, this method is still used for simple systems , simplified models and, hopefully, for training undergraduate students dealing with Newtonian mechanics. Although it is subject to human error, manual generation often yields optimized mathematical equations, in terms of arithmetical and trigonometrical expressions: this remark is fundamental in the context of symbolic generation.

Numerical generation means that, each time a computation is required by some analysis process, the model is rebuilt by numerous calls of subroutines, each involved in some specific parts of the equations or specific vector/tensor operations (library subroutines). This multitude of subroutines is the consequence of the universal nature of multibody programs, and the multiple calls are partially responsible for the heaviness of the numerical generation, compared to manual generation. Experiments in multibody dynamics show that for most applications, many data values, e.g. components of geometrical vectors, inertia parameters, joint velocities, etc., are simply equal to zero. All these zero quantities are treated in a numerical process in the same way as the non-zero ones: a manual generation of the model would eliminate all the unnecessary arithmetical operations beforehand, yielding an optimal form of the equations of motion.

Symbolic generation of multibody models tries to take advantage of both the numerical and manual techniques. So-called symbolic multibody programs manipulate only arithmetical operators (+, ., ., /) and strings of alphanumeric characters (mi, qdi, dij, $\ldots$) to generate – in a set of files – the analytical form of the equations using a desired syntax (e.g. C, Java, Fortran, MATLAB, etc.). For a given multibody application, this symbolic generation is performed only once, as in manual generation. From the multibody modeling point of view, these symbolic generators exhibit the same level of generality as their numerical competitors in the sense that they can handle systems with any topology and containing several degrees of freedom. However, they allow drastic simplifications, from the most trivial (addition/multiplication by zero) to the most complex ones (simplification of long trigonometric expressions). Comparison tests (Samin and Fisette, 2003) showed that, all other things being equal, a symbolic multibody model performs a time simulation between five and ten times faster than a purely numerical one. This is not at all negligible. This speed enhancement motivated the development of symbolic multibody codes in the eighties since computers were at that time quite inefficient in simulating even medium-sized systems (around 10 d.o.f.). The computational superiority of the symbolic approach still remains attractive since the exponential increase in modern computer performances naturally provides the opportunity to investigate larger systems or more sophisticated models (Samin and Fisette, 2003).

More recently, another interesting aspect of the symbolic approach has emerged in the framework of the analysis of hybrid mechatronic systems involving enlarged multibody models (i.e. including electrical actuators, hydraulic devices, etc.) and requiring enlarged analysis (control, optimization, etc.). Indeed, since symbolic multibody programs clearly separate the modeling part from the analysis process, it is quite straightforward to extract the symbolic model in order to export it towards another environment (e.g. for control or optimization purposes). In other words, symbolic multibody models are portable, and this is very attractive for the analysis of multiphysics applications (Samin et al., 2007).

1.3 State-of-the-art

An interesting state-of-the-art of multibody symbolic software is proposed in (Kurz et al., 2010). This paper presents the new features of the research software Neweul-M^2 (involving Maple and MuPad algebra symbolic engines), successor of the Neweul program developed at the end of the seventies and which is one of the first symbolic computer implementation.

The common denominator that emerges when analyzing and comparing multibody symbolic software is their intrinsic versatility in terms of:

– Model type and underlying formalisms (inverse or direct models, kinematic, dynamic equations, sensitivity matrices, etc.).

– Generated languages (Fortran, C, Java, Matlab, etc.).

– Coupling with other engineering environments (Matlab, Simulink, Comsol, etc.).

In addition to this, they all take advantage of modern languages (e.g. Java) and graphical interfaces (GUI, CAD) for data introduction and results presentation. This aspect, which is naturally stressed for visibility or commercial purposes, is far from being trivial in our discipline. Indeed, the large variety of mechanical devices in the three-dimensional space (e.g. joints with multiple d.of. possibly constrained) and of force laws (e.g. issuing from friction, pneumatic pressure or even look-up table, etc.) which is specific to MBS, definitively requires a high-level reflexion to make a general purpose program both flexible, user-friendly and efficient. With respect to this, a particular attention has been paid to ROBOTRAN to preserve a good equilibrium between the user leeway and the automated processes (see Sect. 4).

Naturally guided by specific educational objectives, research topics and/or industrial projects and collaboration opportunities, symbolic programs have their own specificities and have evolved in different directions.

In short, Maplesim (successor of DynaFlexPro) focuses on graphical way of modeling, the linear graph theory being at the root of the approach (Shi and McPhee, 2000). As simulation engine, it uses Maplesoft, a world leader in mathematical and analytical software. Maplesim is able to deal with rigid and flexible MBS and, thanks to the graph approach, is intrinsically well-suited to deal with multiphysics applications.

The research software Neweul-M^2 (Kurz et al., 2010) is based on the Newton-Euler equations and on the virtual work principle to generate the differential [differential/algebraic] equations of motion of open-loop [closed-loop] systems, via the state-of-the-art computer algebra systems Maple or Mupad.

MOBILE (Kecskeméthy, 1993) was developed by A. Kecskeméthy in the nineties to model multibody mechatronic systems using an original object-oriented approach. In particular, the underlying formulation is able to provide an analytical closed-form solution for most of 3-D loop constraints, on the basis of the so-called kinematic pair approach (Kecskeméthy et al., 1997).

Fast simulation is at the root of the SD/FAST program, providing the equations as C or Fortran source code, which can be compiled and linked into any computers environment to perform real time simulation on standard computers.

Carsim (and all its vehicle companion programs) is the successor of Autosim (developed by M. Sayers) and clearly focuses on vehicle dynamics (race cars, passenger cars, trucks, etc.). It is based on a Lisp symbolic multibody program to generate the vehicle equations of motion in symbolic form for real-time simulation purpose.

MotionGenesis, the successor of the symbolic program Autolev (developed by Th. Kane), is scientist-oriented, the user being clearly involved in the model generation. The developers also emphasize the high performances of the program in terms of code compactness and time simulation.

As regards ROBOTRAN, the purpose of the present paper is to highlight the recent developments and features of the program, not only in terms of modeling features and computational performances but also by referring to the flexibility of the approach for educational, research and industrial purposes.

In Sect. 2, the ROBOTRAN underlying formalisms are briefly reviewed; more details can be found in (Samin and Fisette, 2003). Section 3 presents the symbolic capabilities which highly takes advantage of the recursive nature of the formalisms to generate the equations in compact form. In particular, the recent developments of the symbolic engine allow ROBOTRAN to generate symbolically the dynamic model and the sensitivity equations of very large MBS with closed-loop constraints. In Sect. 4, the ROBOTRAN user environment is shortly described as the latter is really part of multibody program appeals, considering the large variety of possible systems to analyses. Some recent applications will be presented in Sect. 5 before concluding.

2 ROBOTRAN formalisms

Before presenting the symbolic engine and the advanced features of the program, it is necessary to establish the notations and to summarize the underlying formalisms. This is the purpose of the present section. More details can be found in Samin and Fisette (2003).

2.1 Dynamics of tree-like multibody systems

Initially developed for Robotic applications (Maes et al., 1990)[1], the formalisms underlying ROBOTRAN are based on the use of *relative joint coordinates*. As usual in this case, equations are firstly established for a tree-like structure (i.e. with no explicit constraints). Constrained systems (i.e. containing loops of bodies or user constraints) are modeled by first restoring a tree-like structure whose dedicated formalisms are thus necessary for any kind of MBS.

2.1.1 Direct dynamics

To predict the motion of a MBS, the *direct dynamics* of MBS (see Fig. 1) is required to compute the generalized accelerations $\ddot{q}$ (joint accelerations) for a given configuration $(q,\dot{q})$ of the MBS to which forces and torques are applied.

[1]ROBOTRAN stands for “ROBOt TRANslator”.

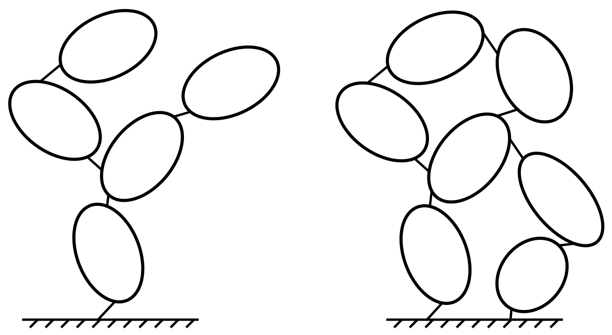


Figure 2. Tree-like versus Closed MBS.

In a synthetic form, the so-called *semi-explicit* direct dynamics model reads:

$$M(q,\delta)\ddot{q} + c(q,\dot{q},\delta,fr,tr,g) = \phi(q,\dot{q}) \tag{1}$$

or, in an *explicit* form:

$$\ddot{q} = f(q,\dot{q},fr,tr,\phi,\delta) \tag{2}$$

to be solved with respect to $\ddot{q}$. In case of the semi-explicit form (1), this can be performed via the Cholesky decomposition of mass matrix M. In the previous equations:

- M $[n \cdot n]$ is the generalized mass matrix of the system (which is symmetric and positive definite),
- c $[n \cdot 1]$ is the non linear dynamic vector which contains the gyroscopic, centrifugal effects as well as the contribution of gravity g, external resultant forces fr and torques tr,
- q $[n \cdot 1]$ denotes the relative generalized coordinates,
- δ $[10\,n \cdot 1]$ gathers together the dynamic parameters of the system (body masses, centers of mass location and the six components of body tensors inertia
- ϕ $[n \cdot 1]$ represents the generalized joint forces (torques): their explicit computation is typical in relative coordinate formulations, the reason being strongly related to robotic applications and inverse dynamics issues.

To obtain the above models, the symbolic engine of ROBOTRAN implements several formalisms[2].

The semi-explicit form (1) is obtained via the so-called Newton-Euler recursive algorithm[3] with mass matrix extraction (Samin and Fisette, 2003).

The explicit direct dynamics model (2) is symbolically generated

- either via the so-called *Order-N* formulation, inspired from (Schwertassek and Rulka, 1989), which requires three recursive steps to get the acceleration with an $O(N)$ complexity only
- or via the previous Newton-Euler recursive algorithm including a fully symbolic Cholesky decomposition of the mass matrix (Postiau et al., 2001).

In terms of equations complexity (i.e. the number of arithmetical operations versus the number of d.o.f.) to produce the generalized accelerations $\ddot{q}$, a detailed comparison between formalisms in relative coordinates has indicated the obvious superiority of the recursive formulations with respect the the non-recursive one, and a certain competition between the Order-N and the Newton/Euler recursive algorithms (Samin and Fisette, 2003). The latter is surprisingly more efficient – although having an $O(N)^2 + O(N)^3$ complexity – for most of practical applications dealing with rigid bodies.

2.1.2 Inverse dynamics

The *inverse dynamics* of a multibody system is the computation of the generalized joint forces (torques) ϕ to be applied to the joints for a given configuration $(q,\dot{q},\ddot{q})$ of the system to which external forces and torques are applied:

$$\phi = f(q,\dot{q},\ddot{q},\delta,fr,tr,g) \tag{3}$$

in which the dimension of ϕ and q are equal for a fully actuated system.

Inverse dynamic models (3) are typically used in robotics to control the actuator torques ϕ when following a desired trajectory $(q(t),\dot{q}(t),\ddot{q}(t))$, in biomechanics to predict the net torque in the human body joints when walking or jumping, etc.

Equations (3) are generated in ROBOTRAN via both non-recursive (Virtual principle) and recursive (Newton/Euler) formalisms; the latter, being implicit with respect to the accelerations $\ddot{q}$, has an $O(N)$ complexity.

2.1.3 Reaction dynamics

The *reaction model* is a particular inverse model which is also of practical use in robotics. It consists in computing the components, in the inertial frame, of the vector reaction force F^{r} and torque T^{r} at a reference point of the bed-plate of the robot. Defining the following $[6 \cdot 1]$ column vector,

$$\phi^{\mathrm{r}} = \begin{pmatrix} F^{\mathrm{r}} \\ T^{\mathrm{r}} \end{pmatrix} \tag{4}$$

the inverse reaction model reads:

$$\phi^{\mathrm{r}} = \phi^{r}(q,\dot{q},\ddot{q},\delta,fr,tr,g) \tag{5}$$

To compute it, ROBOTRAN automatically inserts 6 locked degrees of freedom (3 translations followed by 3 rotations)

[2]Although only the recursive ones are available from the web interface.

[3]For which lots of declination exist in the literature.

between the inertial base and the first body of the MBS. Then, using the Newton/Euler recursive algorithm, the dynamic equations of (only) these joints are symbolically generated to obtain (5).

The ROBOTRAN reaction model has been successfully used to identify dynamic parameters of robots, by establishing the relationship between [a set of] exciting trajectories and the reaction forces and torques: the main advantage with respect to an inverse dynamics approach relates to the absence of the joint friction forces which cancel each other out in (5) by the action/reaction principle (Chenut et al., 2002).

2.2 Dynamics of constrained multibody systems

2.2.1 Direct dynamics

Dynamics and constraints equations

In reality, most multibody applications contain loops of bodies (parallel robot, robotical orthesis, railway bogie, etc., as shown in Fig. 3) which impose the generalized joint coordinates q to satisfy algebraic constraints at any time, denoted $h_{\text{loop}}(q) = 0$. Constraints can result from other physical phenomena (e.g.: geometrically constrained motion, rolling without slipping condition, etc.): those will be referred to user constraints and denoted $h_{\text{user}}(q) = 0$. Gathering these m constraints together, we can write:

$$h(q) = \begin{pmatrix} h_{\text{loop}}(q) \\ h_{\text{user}}(q) \end{pmatrix} = 0 \quad [m \cdot 1]$$

In order to fully describe the system, these constraints and their first and second time derivatives must be added to the equations of motion[4], in which constraint forces are introduced via the well-established Lagrange multipliers technique:

$$M(q)\ddot{q} + c(q,\dot{q}, fr, tr, g) = \phi(q,\dot{q}) + J^t\lambda \tag{6}$$

$$h(q) = 0 \tag{7}$$

$$\dot{h}(q,\dot{q}) = J(q)\dot{q} = 0 \tag{8}$$

$$\ddot{h}(q,\dot{q},\ddot{q}) = J(q)\ddot{q} + \dot{J}\dot{q}(q,\dot{q}) = 0 \tag{9}$$

where:

- $J = \frac{\partial h}{\partial q^t}$ denotes the constraint Jacobian matrix (dimension: $[m \cdot n]$),
- $\dot{J}\dot{q}(q,\dot{q})$ $[m \cdot 1]$ is the quadratic term (expression in $\dot{q}^i\,\dot{q}^j$) of the constraints at acceleration level (dimension: $[m \cdot 1]$),
- λ represents the Lagrange multipliers associated with the constraints (dimension: $[m \cdot 1]$).

[4] In which, for legibility reasons, we will no longer indicate the dependence with respect to the dynamic parameters δ.

Coordinate partitioning reduction

Various methods can be used to solve system (6–9). Among these, one can opt for a full reduction of the system to a purely differential form, which can be obtained by means of the *Coordinate Partitioning technique* (Wehage and Haug, 1982). The Jacobian matrix J is assumed to have full rank m. In this case the constraints $h(q) = 0$ are independent and m generalized coordinates can be locally expressed as functions of the $(n-m)$ others. In this way, it becomes possible to reduce the original DAE[5] system (6–9) to a set of $(n-m)$ differential equations (ODE)[6] in those $(n-m)$ independent coordinates. This reduced set will represent the equations of motion of the constrained multibody system, where $(n-m)$ also corresponds to its number of degrees of freedom.

Let us summarize the steps required to obtain these equations of motion. After reordering the vector of generalized coordinates q (and the columns of the constraint Jacobian J), we can perform the following partition:

$$q = \begin{pmatrix} \mathrm{u} \\ \mathrm{v} \end{pmatrix} ; J = \begin{pmatrix} J_u & J_v \end{pmatrix} \tag{10}$$

where u denotes the subset of $(n-m)$ *independent* coordinates and v denotes the subset of *dependent* coordinates. When correctly choosing the subset v, the m by m matrix J_v will be regular. By "correctly", we mean that, to establish this partitioning for a given application, we can rely:

- on an intuitive reasoning, based on the system configuration (e.g. for a planar slider-crank mechanism, the crank rotation can be chosen as the independent variable u whatever the crank position),
- on the LU factorization of the *full* Jacobian matrix $J(q)$, with column permutation on the basis of the largest pivot. The resulting left $[m \cdot m]$ square block will be the "best" candidate.

Once the coordinate partitioning is established, the reduction method simply uses matrix permutations and operations to produce the final system. Let us first partition the generalized mass matrix M and the vector c according to the coordinate partitioning (10):

$$\begin{pmatrix} M_{uu} & M_{uv} \\ M_{vu} & M_{vv} \end{pmatrix}\begin{pmatrix} \ddot{u} \\ \ddot{v} \end{pmatrix} + \begin{pmatrix} c_u \\ c_v \end{pmatrix} = \begin{pmatrix} \phi_u \\ \phi_v \end{pmatrix} + \begin{pmatrix} J_u{}^t \\ J_v{}^t \end{pmatrix}\lambda \tag{11}$$

where $J_v{}^t$ refers to the transpose of matrix J_v. Since this matrix is regular, eliminating the unknowns λ using the lower part of system (11) produces:

$$\begin{pmatrix} M_{uu} & M_{uv} \end{pmatrix}\begin{pmatrix} \ddot{u} \\ \ddot{v} \end{pmatrix} + B_{vu}{}^t\begin{pmatrix} M_{vu} & M_{vv} \end{pmatrix}\begin{pmatrix} \ddot{u} \\ \ddot{v} \end{pmatrix} \tag{12}$$
$$+c_u + B_{vu}{}^t c_v = \phi_u + B_{vu}^t\phi_v$$

[5]For "differential algebraic equations".
[6]For "ordinary differential equations".

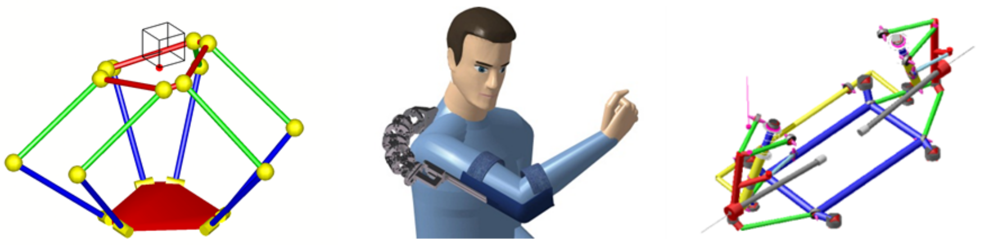


Figure 3. Closed-loop MBS (ROBOTRAN applications).

where we define the so-called coupling matrix: $B_{vu} \overset{\Delta}{=} -(J_v)^{-1} J_u$. The algebraic constraints have to be solved in order to eliminate the dependent variables v. While analytical solutions can exist for specific cases, the algebraic constraints (7) are generally nonlinear and require an iterative procedure to be solved: the Newton-Raphson algorithm – with possible relaxation – can be used for successive estimations of v:

$$v^{k+1} = v^k - (J_v)^{-1} h|_{v=v^k} \tag{13}$$

where the right hand side is evaluated for $v = v^k$ and the values of u corresponding to the instantaneous system configuration.

Using the first (Eq. 8) and second derivatives (Eq. 9) of the constraints, the generalized velocities and accelerations $\dot{v}$ and $\ddot{v}$ are respectively given by:

$$\dot{v} \quad = \quad B_{vu}\dot{u} \tag{14}$$

$$\ddot{v} \quad = \quad B_{vu}\ddot{u} + b \quad \text{with} \quad b \overset{\Delta}{=} -J_v^{-1}(\dot{J}\dot{q}) \tag{15}$$

and can also be eliminated from the differential Eq. (13). This produces the final *reduced* system:

$$\left(M_{uu} + M_{uv}B_{vu} + B_{vu}{}^t M_{vu} + B_{vu}{}^t M_{vv}B_{vu}\right)\ddot{u}$$
$$+\left(M_{uv} + B_{vu}{}^t M_{vv}\right) b + (c_u + B_{vu}{}^t c_v) - (\phi_u + B^t_{vu}\phi_v) = 0$$

which can be concisely written as:

$$\mathcal{M}(u)\ddot{u} + \mathcal{F}(\dot{u}, u) = 0 \tag{16}$$

The set of ordinary differential Eq. (16) constitutes the equations of motion of the constrained MBS described in terms of the $n-m$ independent generalized coordinates u.

- $\ddot{v}$ can be computed directly from system (15).
- As regards the Lagrange multipliers λ, the lower part of Eq. (11) can be used to recover them:

$$\lambda = (J_v{}^t)^{-1}\{M_{vu}\ddot{u} + M_{vv}\ddot{v} + c_v - \phi_v\} \tag{17}$$

2.2.2 Inverse dynamics

As for tree-like MBS, one can be interested in computing the value of the joint torques of a closed MBS for a given trajectory $(q(t), \dot{q}(t), \ddot{q}(t))$. This is the case of parallel manipulators for instance. The coordinate partitioning can also be used to reduce the inverse dynamic model (3) subject to kinematic constraints (7).

Let us denote ψ the left-hand-side of Eq. (6). The latter then reads:

$$\psi = \phi + J^t\lambda \tag{18}$$

Applying the previous coordinate partitioning $(q = u, v)$ to (18) and recalling the definition of the coupling matrix B_{vu}, we obtain:

$$\psi_u = \phi_u + B^t_{vu}(\psi_v - \phi_v) \tag{19}$$

To compute the inverse dynamics in a general case, let us first split the joint generalized force ϕ into an active component $\phi^{\rm a}$ (corresponding to actuators) and a passive component $\phi^{\rm p}$ (e.g. friction, spring-type law, etc.):

$$\phi = \phi^{\rm a} + \phi^{\rm p} \tag{20}$$

Assuming that the actuators are located on each independent joints u (which also assumes that there are as many actuators as degrees of freedom $n-m$),

$$\phi^{\rm a} = \begin{pmatrix} \phi^{\rm a}_u \\ 0 \end{pmatrix}, \tag{21}$$

the inverse dynamics (19) becomes:

$$\phi^{\rm a}_u = \psi_u - \phi^{\rm p}_u + B^t_{vu}(\phi^{\rm p}_v - \psi_v) \tag{22}$$

In practical situations, actuators are not necessary located on the independent joints u, because the $\{u, v\}$ partitioning results from a numerical requirement (matrix conditioning) and not from physical considerations. However, nothing prevents us from considering two distinct partitioning inside a unique inverse dynamic model:

- the $q = \{u, v\}$ partitioning to assemble the MBS and to solve the constraints (Eqs. 13, 14 and 15),
- a second coordinate partitioning, $q = \{q_{\rm a}, q_{\rm p}\}$, based on the actuated and non actuated joints.

Using this second partitioning, the reduced inverse dynamic model (22) simply becomes:

$$\phi^{\mathrm{a}}_{q_{\mathrm{a}}} = \psi_{q_{\mathrm{a}}} - \phi^{\mathrm{p}}_{q_{\mathrm{a}}} + B^{t}_{q_{\mathrm{p}}q_{\mathrm{a}}}(\phi^{\mathrm{p}}_{q_{\mathrm{p}}} - \psi_{q_{\mathrm{p}}}) \tag{23}$$

which requires that the constraints Jacobian sub-matrix $J_{q_{\mathrm{p}}}$ be regular.

When dealing with *overactuated* MBS for which the number of actuators (ϕ^{a}) is larger than the number of d.o.f., there are an infinite number of solutions for the inverse dynamics which becomes an underdetermined system of the form:

$$A(q)\phi^{\mathrm{a}} = b(q,\dot{q},\ddot{q}) \tag{24}$$

where A is a m by n rectangular matrix (with $m < n$). Additional criteria are thus needed to solve the system (24). One can use an optimization process to satisfy some specific criteria as in (Raison et al., 2010) to deal with human muscle overactuation, or the Moore-Penrose pseudo-inverse solution:

$$\phi^{\mathrm{a}} = A^{+}\, b \tag{25}$$

where, assuming rank$(A) = m$, $A^{+} \triangleq A^{t}(A A^{t})^{-1}$.

Equation (25) gives the solution of minimum Euclidean norm $||\phi||_2$. This approach has been successfully used for instance in (Ganovski et al., 2004) to minimize actuator torques of overactuated parallel robots following trajectories with singular configurations.

In case of *underactuated* MBS (i.e. less actuators than d.o.f.), instead of using the previous inverse models, the direct dynamics form (16) can be used by splitting the independent coordinates u into free variables u_{f} and actuated variables u_{a} whose value is constrained according to a prescribed motion or trajectory. Equation (16) then becomes:

$$\begin{pmatrix} \mathcal{M}_{u_{\mathrm{f}}u_{\mathrm{f}}} & \mathcal{M}_{u_{\mathrm{f}}u_{\mathrm{a}}} \\ \mathcal{M}_{u_{\mathrm{a}}u_{\mathrm{f}}} & \mathcal{M}_{u_{\mathrm{a}}u_{\mathrm{a}}} \end{pmatrix} \begin{pmatrix} \ddot{u}_{\mathrm{f}} \\ \ddot{u}_{\mathrm{a}} \end{pmatrix} + \begin{pmatrix} \mathcal{F}_{u_{\mathrm{f}}} \\ \mathcal{F}_{u_{\mathrm{a}}} \end{pmatrix} = \begin{pmatrix} 0 \\ \lambda \end{pmatrix} \tag{26}$$

This system can be seen as an hybrid direct/inverse model where the upper part (related to the free motion) refers to direct dynamics (unknown $\ddot{u}_{\mathrm{f}}$) and the lower part refers to the collocated inverse dynamics whose unknowns (λ) correspond to the actuated joint forces ϕ^{a}:

$$\phi^{\mathrm{a}} = \begin{pmatrix} \mathcal{M}_{u_{\mathrm{a}}u_{\mathrm{f}}} & \mathcal{M}_{u_{\mathrm{a}}u_{\mathrm{a}}} \end{pmatrix} \begin{pmatrix} \ddot{u}_{\mathrm{f}} \\ \ddot{u}_{\mathrm{a}} \end{pmatrix} + \mathcal{F}_{u_{\mathrm{a}}} \tag{27}$$

in which $\ddot{u}_{\mathrm{f}}$ must be computed in parallel via a time simulation for instance.

3 Symbolic engine

There do exist commercial general purpose symbolic computation packages like Maple and Mathematica. Why do we not use these to generate multibody equations ? There are two main reasons for this.

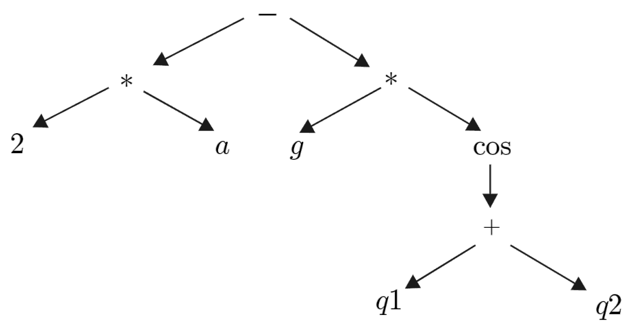


Figure 4. Tree representation of a mathematical expression.

The first and most important one relates to the amount of computer memory required to generate medium-sized and large models (up to 300 d.o.f. in our case) symbolically.

The second reason concerns the possible simplifications of the symbolic expressions. Although the simplification capabilities of commercial packages are extremely powerful, the condensation of multibody equations relies on specific rules which can be applied more easily by developing a dedicated symbolic program; this was the main motivation to develop the ROBOTRAN program. Its capabilities in terms of symbolic manipulation are briefly summarized in the following sub-sections, dealing with:

- expression simplification
- memory allocation
- advanced features
  - fully symbolic generation of constrained MBS
  - recursive differentiation of multibody models.

3.1 Expression simplification

A mathematical expression in multibody equations uses simple arithmetical operators $+, -, \cdot, /, (), =$ and functions: mainly sin() and cos(), occasionally: sqrt(), atan(),....

Let us for example consider the following expression

$$2 \cdot a - g \cdot \cos(q1 + q2) \tag{28}$$

which, when applying mathematical priority rules, is equivalent to

$$(2 \cdot a) - (g \cdot (\cos(q1 + q2))) \tag{29}$$

This expression can be represented by the tree shown in Fig. 4. For instance, the second multiplication is an expression whose nature is $\cdot$ and points towards two arguments: the expressions g and cos(). In a tree representation, the leaves represent alphanumeric symbols, e.g. $2, a, g, \ldots$ in example (28). They are also considered as expressions (or "leaf expressions") but they have no sub-expression to which they point: they simply contain the string they represent.

The tree representation and the dynamic increase in size of the expressions during the elaboration of the equations leads us to represent an expression by a C-structure, *dynamically* generated and handled via *pointers*. At the root of the program, there is the C-structure *expression* which contains – at least – the following fields:

- the *nature* of the expression ($+,-$,string, etc.),
- the address(es) (i.e. pointers) of the *argument*(s) to which the expression points: these arguments are either strings – for a leaf expression – or other expressions which have been created via a previous dynamic memory allocation. To ensure the basic level of symbolic simplification (i.e. expression $c+a+b-a$ replaced by $c+b$), two ordering rules are used.

The first one re-organizes a given expression on the basis of a so-called *multibody priority (e.g. a mass symbol has a higher priority than a force symbol but has a lesser priority than a coefficient), as illustrated in the following example:*

$$4 < m^i < d^{ij} < F^j_{ext} < \dot{q}^i < b \cdot c < (a-d) \tag{30}$$

In case of equal multibody priority, a (sub-)expression is then re-organized according to the lexicographical sequence of the ASCII table of characters, as for instance:

$$K^{ii} < m^i; \;\; 2 < 3; \;\; d^{ii} + d^{ij} < d^{ii} + d^{jk}; \;\; \text{etc.} \tag{31}$$

When dealing with a given symbolic expression (whatever its length), the symbolic engine (layer 1) of ROBOTRAN recursively uses the above priority rules to ensures that the final form of any expression will be purged of consecutive equal terms – or sub-expressions – with opposite signs.

3.2 Trigonometric simplification

A revolute joint i, with a generalized joint coordinate q^i, of a multibody system induces an elementary rotation matrix which contains the trigonometric functions $\cos(q^i)$ and $\sin(q^i)$. The evaluation of rotation matrices between any pair of bodies i and j, such that j is a descendant of i in the MBS structure, is obtained by multiplying the elementary rotation matrices associated with each revolute joint k belonging to the kinematic chain $\{i, i+1, i+2, ..., j-1, j\}$ for instance:

$$R^{j,i} = R^{j,j-1} R^{j-1,j-2} \ldots R^{i+1,i} \tag{32}$$

Thus, an optimized trigonometric engine is required to condensate trigonometric expressions like

$$\begin{aligned} &K212 \cdot qpp8 \cdot (C2 \cdot C2 \cdot C3 \cdot S4 + S2 \cdot S24 \\ &\quad -S2 \cdot S34 \cdot S5 + S2 \cdot S2 \cdot C3 \cdot S4) \end{aligned} \tag{33}$$

in which Cj, Sk and Sjk represent $\cos(q^j)$, $\sin(q^k)$ and $\sin(q^j + q^k)$ respectively.

The ROBOTRAN trigonometric simplification process, which is performed on line (i.e. it is not a post-process), has two main levels. The first and lowest one systematically detects and performs the fundamental trigonometric simplifications according to well-known formulae. For instance, let us consider a symbolic expression a:

- if $a = C1 \cdot C1 + S1 \cdot S1$, the process returns 1,
- if $a = 2 \cdot C1 \cdot S1$, trigo(a) the process creates and returns the auxiliary variable $S11$, which stands for $\sin(2q^1)$,
- if $a = C1 \cdot C2 - S1 \cdot S2$, the process creates and returns the auxiliary variable $C12$, which stands for $\cos(q^1+q^2)$,
- etc.

while the previous level is able to detect and simplify elementary trigonometric formulae, it is not able to deal with expressions like

$$\begin{aligned} &C2 \cdot C4 \cdot C56 \cdot C56 \cdot S8 + C2 \cdot C4 \cdot S56 \cdot S56 \cdot S8 \\ &\quad + C2 \cdot S4 \cdot S56 \cdot C8 + S2 \cdot C4 \cdot S56 \cdot C8 \\ &-S2 \cdot S4 \cdot C56 \cdot C56 \cdot S8 - S2 \cdot S4 \cdot S56 \cdot S56 \cdot S8 \end{aligned} \tag{34}$$

for which judicious groupings and factorings must be performed in order to make a maximum of trigonometric formulae appear, which can then be simplified. This is the purpose of the second level of the process. The following examples illustrate the power of the method. The trigonometric expressions are generated by a direct model (of a railway bogie) based on the virtual power principle (Cj, Sk and Sjk represent $\cos(q^j)$, $\sin(q^k)$ and $\sin(q^j + q^k)$ respectively):

- $C2 \cdot C4 \cdot C56 \cdot C56 \cdot S8 + C2 \cdot C4 \cdot S56 \cdot S56 \cdot S8$
  $+ C2 \cdot S4 \cdot S56 \cdot C8 + S2 \cdot C4 \cdot S56 \cdot C8$
  $- S2 \cdot S4 \cdot C56 \cdot C56 \cdot S8 - S2 \cdot S4 \cdot S56 \cdot S56 \cdot S8$
  becomes: $C24 \cdot S8 + S24 \cdot S56 \cdot C8$
- $C2 \cdot S2 \cdot C4 \cdot C56 \cdot C56 \cdot S8 + C2 \cdot S2 \cdot C4 \cdot S56 \cdot S56 \cdot S8$
  $+ C2 \cdot S2 \cdot S4 \cdot S56 \cdot C8 + S2 \cdot S2 \cdot C4 \cdot S56 \cdot C8$
  $- S2 \cdot S2 \cdot S4 \cdot C56 \cdot C56 \cdot S8 - S2 \cdot S2 \cdot S4 \cdot S56 \cdot S56 \cdot S8$
  becomes: $S2 \cdot (C24 \cdot S8 + S24 \cdot S56 \cdot C8)$

3.3 Recursive scheme condensation

In multibody dynamics, a *recursive scheme* denotes any formalism (kinematic, dynamic, direct, inverse, etc.) in relative coordinates, written as one or more algorithmic loops covering the MBS from the base body to the terminal bodies. For instance, the so-called Recursive Newton Euler formalism represents a recursive scheme consisting of two algorithmic loops: one for the forward kinematics (for $i = 1 : N^{\text{body}}$), the second for the backward dynamics (for $i = N^{\text{body}} : 1$). The so-called Order-N formalism (Schwertassek and Rulka, 1989) is also a recursive scheme which performs three recursions: forward kinematics, backward dynamics and forward kinetics to directly obtain the explicit direct dynamics (2).

In such formalisms, the relation which expresses – for instance – the angular velocity ω^3 of a given body 3 with respect to its parent body 2, ω^2, is written in vector form

$$\omega^3 = \omega^2 + \Omega^{23} \tag{35}$$

where Ω^{23} stands for the *relative* angular velocity between body 2 and 3.

A recursive ROBOTRAN implementation of the vector Eq. (35) is given hereafter for a specific MBS:

$$\begin{aligned} \mathrm{OM13} &= qd(3) + \mathrm{OM12} \\ \mathrm{OM23} &= \mathrm{OM22} \cdot C3 + \mathrm{OM32} \cdot S3 \\ \mathrm{OM33} &= -\mathrm{OM22} \cdot S3 + \mathrm{OM32} \cdot C3 \end{aligned} \tag{36}$$

where OMij denotes the i-th component of the j-th body angular velocity in body i fixed frame.

The two above equations clearly highlight the *recursive* nature of the analytic Eq. (35) on the one hand and of the corresponding symbolic expressions (36) on the other hand.

Via this very simple example, one can easily extrapolate the reasoning to a full formalism in which such a recursivity between adjacent bodies can apply to position, velocity, acceleration, forces, torques, etc. to end up in the final model (Eqs. 1 and 2 for instance). The ROBOTRAN implementation is based on this technique.

Although recursive formulations intrinsically have a compact form (when compared with *in extenso* formulations which do not exploit the above-mentioned recursivity (Samin and Fisette, 2003), they paradoxically perform superfluous evaluations: depending on the type and succession of joints of the application, some components of the vector components (in Eq. 36 for instance) are superfluous for the final scalar form. Whereas a general purpose multibody program which generates the model *numerically* is not able to detect these superfluous equations, a symbolic multibody program can do so. Thus, in addition to removing useless terms in equations (see the previous sections), *entire* equations can be detected symbolically as being superfluous (up to 30 % for direct dynamics!). The ROBOTRAN recursive condensation process is based on a linked-list (of equations) and C-pointers, and removes those equations before "engraving" the final result. The process is illustrated in Fig. 5 via an academic example which computes a given result R from data A, B, C, D via a recursive approach: symbolic equations $B1 = \ldots$, $B3 = \ldots$, $D1 = \ldots$ are clearly useless for the result and can be completely disregarded and removed from the list before printing the equations.

3.4 Memory allocation

As mentioned above, the amount of computer memory required by symbolic programs exponentially increases when manipulating large expressions or system equations: the generation can simply fail or require a very long computer time.

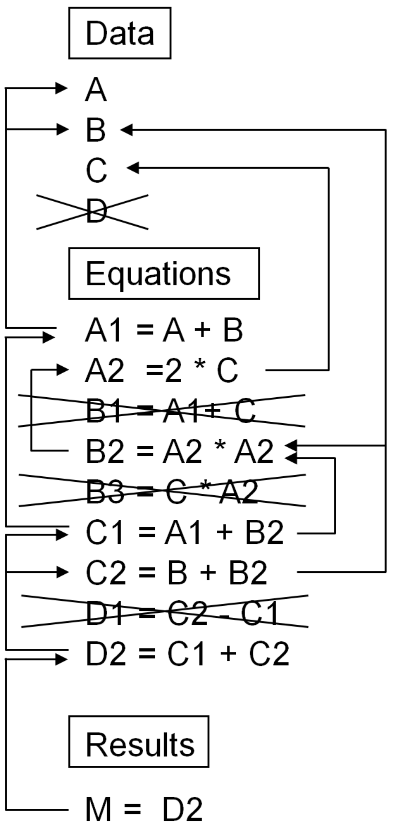


Figure 5. Recursive scheme condensation.

Symbolic manipulation requires dynamic memory allocation to create and store new expressions (C-structures in our case). The symbolic process briefly described above, which recursively re-organizes any new expressions in accordance with the ordering rules, can generate in the memory thousands of auxiliary expressions which are not necessary in the final tree representation of a given (complex) expression. Experiments showed that, whatever the multibody formalism used, these short-lived expressions induced by symbolic manipulations represent more than 90 % of the whole set of expressions generated! This explains why symbolic programs can lead to an explosive use of computer memory. To solve this delicate problem, one may try to cut long equations into segments and evaluate and print them separately; then, after dealing with each segment, we would clear out all the generated expressions from the memory. By experience, this solution is mediocre because it strongly degrades the symbolic engine capabilities in terms of simplification.

To solve this problem, the following three elements were introduced into ROBOTRAN.

1. When generating a symbolic equation, in order to keep track of each new symbolic expression (complex or not, short-lived or not) created by the generation process, the address (pointer) of this expression is systematically

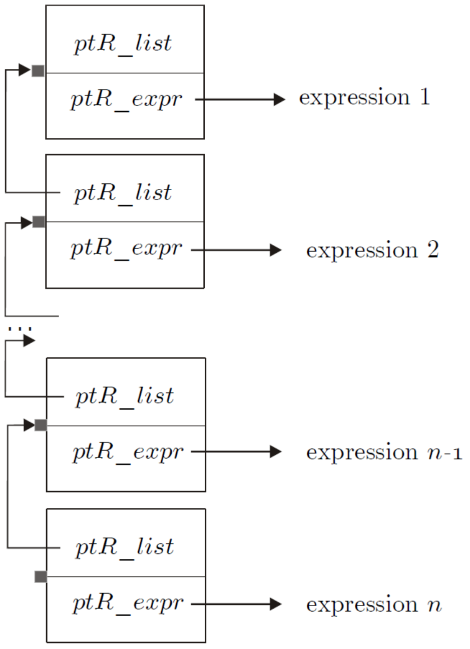


Figure 6. Dynamic linked list of expressions.

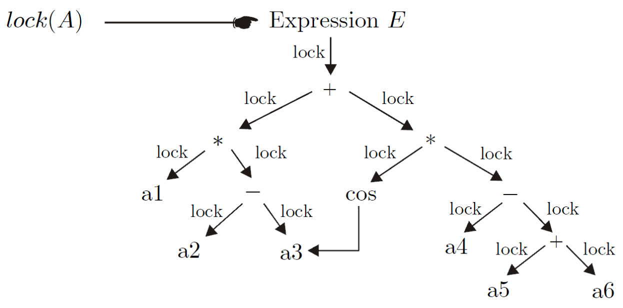


Figure 7. Locking an expression in memory.

stored in a *dynamic linked-list* which grows as and when it is needed, as shown in Fig. 6.

2. In order to protect any expression (e.g. A, $a2$, cos(), '+', etc.), we add to the corresponding C-structure, a boolean field "lock". The value lock = TRUE tells ROBOTRAN that the corresponding expression cannot be removed from memory, whereas lock = FALSE states that it can. FALSE is the default value.

3. We introduce the procedure expr_lock(E) to lock a given expression E, i.e. by setting to TRUE the "lock" field of each node belonging to the *final* tree of expression E. The procedure is thus intrinsically recursive as illustrated in Fig. 7.

By thus locking a given equation E in its *final, simplified* form, we do not lock the intermediate "short-lived" expressions (more than 90 %!) created during the evaluation of E,

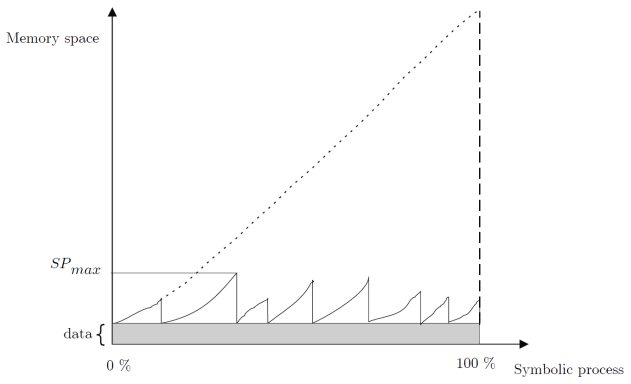


Figure 8. Memory allocation for a ROBOTRAN generation process.

those which do not contribute to its final tree: for these expressions, "lock" keeps its default value FALSE. Thus, to free the memory in an optimal way, once the evaluation of E is finished, and its tree has been purged of superfluous operations, ROBOTRAN:

1. recursively protects the final form of expression E (lock(E)),

2. frees the memory by removing every expression x which has not been locked by covering the list of Fig. 6 from tail to head. Each element of the list is also removed to end up with an empty list … ready for the evaluation of a subsequent equation.

Thanks to this methodology, the memory space required by and during a ROBOTRAN process is illustrated in Fig. 8 and has thus a "toothed" shape, rather than a monotonic growing one as in classical symbolic packages. The maximum memory space (SP_{max}) is reached by the greediest equation.

This freeing process is of the utmost importance in multibody dynamics since it allows us to eliminate the most critical bottleneck (i.e. the memory space) of the generation of large multibody models, up to 300 d.o.f. in our case.

3.5 Advanced features

3.5.1 Fully symbolic generation of constrained MBS

Up to now, the symbolic capabilities of the ROBOTRAN program were only exploited for generating the dynamics and kinematics of tree-like MBS: this means that for closed loop systems (like vehicle suspensions, parallel robots, etc.), only the main ingredients of Eqs. (6), (7), (8) and (9) were generated symbolically but *separately*. The subsequent coordinate partitioning reduction (16) being performed numerically. While being far more efficient than a pure numerical model (Fisette, 1994), we recently noticed that there was still a lot of superfluous operations in the numerical processes underlying the reduction (empty mass and Jacobian

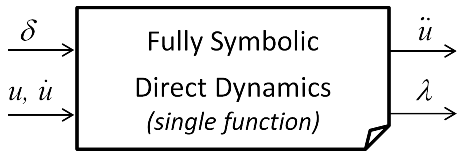


Figure 9. Closed-loop MBS: fully symbolic generation of the direct dynamics.

sub-matrices etc.) The latter could be avoided if the symbolic generation mentioned above would be applied *globally* to the model, i.e. from the input $(u,\dot{u})$ to the output $(\ddot{u})$ in case of direct dynamics of constrained systems (Fisette et al., 2002), (Poncelet et al., 2010).

Let us rewrite the semi-explicit form (16) in an *explicit* way (exactly as the form 2) for tree-like MBS) in terms of the independent accelerations $\ddot{u}$:

$$\ddot{u} = f(u,\dot{u},fr,tr,\phi,\delta) \qquad (37)$$

Thanks to the ROBOTRAN symbolic engine capabilities, in particular in terms of memory requirement (see Sect. 3.4), it is now possible to generate the independent accelerations $\ddot{u}$ according to (Eq. 37) in a fully symbolic manner and in the form of a *unique* function (C, Matlab, etc.) which successfully computes, as a unique global recursive scheme:

1. the constraints and their solving at position, velocity and acceleration levels,
2. the external forces and torques (interfaced with possible external user constitutive equations),
3. the dynamics of the restored tree-like MBS,
4. the reduction and the resolution of system (16) with respect to the generalized accelerations $\ddot{u}$.

Regarding the constraints solution at position level, despite robust and appealing formulations (e.g. the kinematic transformer technique, Kecskeméthy et al., 1997), it is not always possible to solve them analytically via a closed-form kinematic solution, because of their inherent nonlinearities. In case of complex 3-D closed-loop structures, as depicted in Fig. 3, we must often resort to a numerical iterative process to converge towards an accurate solution. As far as we are concerned, the Newton-Raphson algorithm (with possible relaxation) has been chosen to solve the constraints $h(q) = 0$ (Eq. 7) via the necessary iterations on the dependent coordinates v (see Eq. 13):

$$\Delta v^k = (J_v)^{-1}\, h|_{v=v^k} \qquad (38)$$

Being a numerical iterative process, it would be illusory to implement it symbolically. However, it we examine the RHS of the previous iterative formula, it mainly contains kinematic ingredients, namely the constraint Jacobian sub-matrix

```
[udd, λ] = direct_dynamics(u, ud, δ)
/* Recursive Forward Kinematics */
$\omega_j = \omega_i + \dots$
...
/* Loop closure */
while (norm(h) > ε)
    /* Recursive Contraints Kinematics */
    $J_{jk} = J_{ik} + \dots$
    ...
    $\Delta v = \dots$
end
/* Recursive Backward Dynamics*/
$F_j = F_i + \dots$
$L_j = L_i + \dots$
/* DAE=> ODE Reduction */
...
udd = ...
λ = ...
return
```

Figure 10. Recursive symbolic computation of the constraints: local iterative process.

J_v and the constraints themselves $h(u,v)$, which are computed within the recursive generation (see Sect. 3.3). Thus, since the Newton-Raphson algorithm simply amounts to repeat the evaluation of the RHS of Eq. (38) until convergence, it is rather straightforward to insert – in the symbolic output file – suitable statements upstream (e.g.: "while $\|h(q)\| > \epsilon$") and downstream ("$\Delta v^k = ...$, end") the RHS computation of (38)). Figure 10 illustrates this "trick" which has been systematized in ROBOTRAN when generating the explicit direct model (37) whose efficiency in terms of CPU time is rather impressive.

The only drawback of this fully symbolic approach, in comparison with the semi-symbolic one of Sect. 2.2.1, lies in the fact that the $\{u,v\}$ partitioning must be symbolically hard-coded in the block of Fig. 9. This requires a pre-process to fix the partitioning via a numerical technique (e.g. a LU factorization of the Jacobian matrix $J(q)$ with full pivoting) which is able to find a robust $\{u,v\}$ partition and variable permutation.

3.5.2 Recursive differentiation of multibody models

Various numerical analyses require the derivatives of multibody (kinematic or dynamic) equations with respect to a given parameter or variable or a set of those. This is for instance the case for model linearization, control design, deterministic optimization and sensitivity analysis. In the present section, we will focus on the latter topic to illustrate the symbolic differentiation process recently implemented in ROBOTRAN (Poncelet et al., 2010).

Sensitivity analysis of MBS

In the multibody dynamic context, a typical objective function $\psi(p)$ for sensitivity analysis purpose (see Eberhard, 1996; Ding et al., 2007) can be written as:

$$\psi(p) = G^1(t^1, u^1, \dot{u}^1, p) + \int_{t^0}^{t^1} F(t, u, \dot{u}, \ddot{u}, p)\,\mathrm{d}t \tag{39}$$

in which:

- p denotes the parameter;
- t^0 and t^1 are the initial and final simulation time;
- G^1 refers to the final state (e.g. the configuration at time t^1 of a given body of the system);
- F depends on the dynamic behavior of the system in the time interval $[t^0, t^1]$, such as the root mean square (rms) acceleration of the car driver, the mean power dissipated at a wheel/ground contact, etc.

For the general objective function (39), sensitivity analysis consists in computing $\frac{\mathrm{d}\psi}{\mathrm{d}p}$, that is:

$$\frac{\mathrm{d}\psi}{\mathrm{d}p} = \frac{\partial G^1}{\partial u^1}\frac{\mathrm{d}u}{\mathrm{d}p}\bigg|_{t^1} + \frac{\partial G^1}{\partial \dot{u}^1}\frac{\mathrm{d}\dot{u}}{\mathrm{d}p}\bigg|_{t^1} + \frac{\partial G^1}{\partial p} + \int_{t^0}^{t^1}\left(\frac{\partial F}{\partial u}\frac{\mathrm{d}u}{\mathrm{d}p} + \frac{\partial F}{\partial \dot{u}}\frac{\mathrm{d}\dot{u}}{\mathrm{d}p} + \frac{\partial F}{\partial \ddot{u}}\frac{\mathrm{d}\ddot{u}}{\mathrm{d}p} + \frac{\partial F}{\partial p}\right)\mathrm{d}t \tag{40}$$

Let us first point out that in our case the only variables are the *independent* coordinates u (and $\dot{u}$). Indeed, the remaining variables v (and $\dot{v}$) have already been eliminated from the model during the reduction process (from the DAE (6–9) to the ODE (16) or (37)), v and $\dot{v}$ being expressed in terms of u and $\dot{u}$ according to the constraints solution.

Within expression (41), the unknown sensitivity matrices $\frac{\mathrm{d}u}{\mathrm{d}p}(t)$, $\frac{\mathrm{d}\dot{u}}{\mathrm{d}p}(t)$ and $\frac{\mathrm{d}\ddot{u}}{\mathrm{d}p}(t)$ can be computed via the so-called *direct method* (Eberhard, 1996) which consists in solving the differential equations for sensitivity matrices *simultaneously* with the equations of motion as explained here below.

Semi-explicit approach

Considering the semi-explicit form of the dynamic Eq. (16), the above sensitivity matrices can be calculated via the following equations in which, for sake of simplicity, we have defined $\Gamma(u, \dot{u}, \ddot{u}, \delta) \triangleq M_{\text{red}}(u, \delta)\ddot{u} + F_{\text{red}}(u, \dot{u}, f, \delta, p)$:

$$M_{\text{red}}\frac{\mathrm{d}\ddot{u}}{\mathrm{d}p} + \frac{\partial \Gamma}{\partial \dot{u}}\frac{\mathrm{d}\dot{u}}{\mathrm{d}p} + \frac{\partial \Gamma}{\partial u}\frac{\mathrm{d}u}{\mathrm{d}p} + \frac{\partial \Gamma}{\partial p} = 0 \tag{41}$$

This equation can be time integrated simultaneously with the equations of motion (16). The main reason of the "symbolic versus numerical" benefits (factor 8 to 10) comes more from the *recursive* nature of the dynamic equations which is at the root of the symbolic elimination of useless equations in ROBOTRAN (see Sect. 3.3), than from the symbolic simplification of the expressions themselves. In particular, this recursivity is fully exploited when computing the explicit form (37) as stated in Sect. 3.5.1.

In view of Eq. (41), the computation of $\frac{\mathrm{d}\ddot{u}}{\mathrm{d}p}$, $\frac{\mathrm{d}\dot{u}}{\mathrm{d}p}$ and $\frac{\mathrm{d}u}{\mathrm{d}p}$ is required, the first one being the direct result of the model differentiation, the next two one being obtained via time integration.

Explicit approach

In view of the complexity of the semi-explicit form (41), which requires the partial derivative of the Γ term, it is clear that it would be far more advantageous to compute the derivatives $\frac{\mathrm{d}\ddot{u}}{\mathrm{d}p}$ by *directly* differentiating the explicit form (37):

$$\ddot{u} = f(u, \dot{u}, p) \;\Rightarrow\; \frac{\mathrm{d}\ddot{u}}{\mathrm{d}p} = \frac{\mathrm{d}f}{\mathrm{d}p}\left(u, \dot{u}, \frac{\mathrm{d}\dot{u}}{\mathrm{d}p}, \frac{\mathrm{d}u}{\mathrm{d}p}, p\right) \tag{42}$$

Thanks to the – recent – availability of the explicit model (37) in ROBOTRAN under the form of a fully symbolic recursive scheme (see Sect. 3.5.1), the computation of $\frac{\mathrm{d}\ddot{u}}{\mathrm{d}p}$ will be greatly facilitated, for tree-like as for constrained MBS: in ROBOTRAN, a direct symbolic differentiation can be achieved straightforwardly according to (42).

However, as for the model generation, the main complexity comes once again from the constraints at position level $h(q) = 0$ (assumed to be previously solved with respect to v). Without entering into details, the derivative of the dependent coordinates $\frac{\mathrm{d}v}{\mathrm{d}p}$ (and later on, the derivative of the velocities $\frac{\mathrm{d}\dot{v}}{\mathrm{d}p}$) are needed *explicitly* to compute the sensitivity model (42).

Starting from the differentiation of the implicit form of the constraints (which must be satisfied for any value of p),

$$\frac{\mathrm{d}h}{\mathrm{d}p} = \frac{\partial h}{\partial p} + \frac{\partial h}{\partial u}\frac{\mathrm{d}u}{\mathrm{d}p} + \frac{\partial h}{\partial v}\frac{\mathrm{d}v}{\mathrm{d}p} = 0, \tag{43}$$

we can isolate $\frac{\mathrm{d}v}{\mathrm{d}p}$,

$$\frac{\mathrm{d}v}{\mathrm{d}p} = -\left(\frac{\partial h}{\partial v}\right)^{-1}\left[\frac{\partial h}{\partial p} + \frac{\partial h}{\partial u}\frac{\mathrm{d}u}{\mathrm{d}p}\right] = -J_v^{-1}\frac{\partial h}{\partial p} + B_{vu}\frac{\mathrm{d}u}{\mathrm{d}p} \tag{44}$$

Symbolic implementation

Since the symbolic engine of ROBOTRAN (see next section) blindly differentiates any expression on the basis of recursive chain rules, any expression of the model derivative (e.g. $\frac{\mathrm{d}f}{\mathrm{d}p}$ must be the derivative of an "existing" expression in the model itself (e.g. f). For the particular case of (44),we can observe that it exactly corresponds to the derivative of the following relation (which is, incidentally, rather similar to the Newton-Raphson iteration 13):

$$v = -J_v^{-1} h(q) \tag{45}$$

Indeed, remembering that the constraints are satisfied (i.e.: $h(q) = 0$), by differentiating (45), we obtain:

$$\frac{\mathrm{d}v}{\mathrm{d}p} = -J_v^{-1}\frac{\mathrm{d}h}{\mathrm{d}p} = -J_v^{-1}\left[\frac{\partial h}{\partial p} + \frac{\partial h}{\partial u}\frac{\mathrm{d}u}{\mathrm{d}p}\right] \tag{46}$$

which is the desired result (44) associated with the implicit constraints derivation. Sensitivity analysis of large constrained MBS (with more than 100 d.o.f.) led us to develop a specific procedure in ROBOTRAN to symbolically differentiate a given recursive scheme with respect to a given (set of) parameter(s) p. In the context of differentiation, the equations produced by a recursive multibody formalism (see Fig. 5) can be advantageously considered as interwoven functions $(f(g(h(...(p)))))$. However, if the corresponding differentiation rules are blindly applied to such a recursive scheme, we have observed that they produce a very large non-optimized symbolic output (e.g. $\frac{\mathrm{d}f}{\partial p}$ in Eq. 42) even for medium-sized multibody models: the interest of the recursive computation is thus completely lost.

Therefore, we take advantage of the condensation procedure described in Sect. 3.3 to solve this problem. When evaluating a given recursive scheme, we assume – a priori – that each equation depends, explicitly or not, on the set of system parameters or variables (e.g. $p_1, p_2, \dots p_k$) with respect to which the differentiation must be performed. For instance, in the following equation:

$$\mathrm{AUXJ} = \mathrm{AUXI} + 2 \cdot P$$

AUXJ explicitly depends on the variable P via the second term. A priori, it may also depend implicitly on P via the first term AUXI.

We thus systematically create and evaluate a new *recursive* variable, for instance AUXJ_P, for the *total* derivative of the current equation with respect to P

$$\mathrm{AUXJ\_P} = \ \mathrm{AUXI\_P} + 2$$

even if, in the end, it appears that this new auxiliary variable is 0 or simply useless. If it is useless, the elimination process (of Fig. 5) will detect it and remove the corresponding equation from the list, before printing. Such a technique gives rise to a *compact recursive computation* of the derivatives.

To illustrate this, let us consider in (47) and (48), the symbolic evaluation of one element $J_{(3,1)}$ of the Jacobian matrix $\frac{\partial x}{\partial q^t}$ of a position vector $x(q)$ associated with a kinematic chain composed of nine joints.

In (47), the classical differentiation rule applied to $x(q)$ is far more consuming in terms of single operations (it contains 220 multiplications, additions and subtractions) than when obtained via the proposed recursive total differentiation (involving only 62 single operations).

$$\begin{aligned} J(3,1) \quad = \quad & q8\cdot(C1\cdot C7\cdot(-C3\cdot S4\cdot C5+S3\cdot S5)-S1\cdot(S2\cdot(S3\cdot S4\cdot \\ & (-C5\cdot C7+S5\cdot C6\cdot S7)+S7\cdot(-C3\cdot C5\cdot C6+S3\cdot C4\cdot S6)) \\ & +C7\cdot(C2\cdot C4\cdot C5-S2\cdot C3\cdot S5))+S7\cdot(C1\cdot(C3\cdot C4\cdot S6 \\ & +C6\cdot(C3\cdot S4\cdot S5+S3\cdot C5))-S1\cdot C2\cdot(-C4\cdot S5\cdot C6+S4\cdot S6))) \\ & +D13\cdot S1\cdot S2+D14\cdot(C1\cdot S3+S1\cdot S2\cdot C3)+D15\cdot(C1\cdot S3+S1 \\ & \cdot S2\cdot C3)+D16\cdot(C1\cdot(C3\cdot S4\cdot S5+S3\cdot C5)+S1\cdot(C2\cdot C4\cdot S5 \\ & -S2\cdot(-C3\cdot C5+S3\cdot S4\cdot S5)))+D17\cdot(C1\cdot C6\cdot(C3\cdot S4\cdot S5 \\ & +S3\cdot C5)+S1\cdot(-S2\cdot S3\cdot(C4\cdot S6+S4\cdot S5\cdot C6)+C6\cdot(C2\cdot C4 \\ & \cdot S5+S2\cdot C3\cdot C5))+S6\cdot(C1\cdot C3\cdot C4-S1\cdot C2\cdot S4))+D18\cdot \\ & (C1\cdot S7\cdot(C3\cdot S4\cdot C5-S3\cdot S5)-S1\cdot(S2\cdot(S3\cdot S4\cdot(C5\cdot S7 \\ & +S5\cdot C6\cdot C7)+C7\cdot(-C3\cdot C5\cdot C6+S3\cdot C4\cdot S6))+S7\cdot(-C2 \\ & \cdot C4\cdot C5+S2\cdot C3\cdot S5))+C7\cdot(C1\cdot(C3\cdot C4\cdot S6+C6\cdot(C3\cdot S4 \\ & \cdot S5+S3\cdot C5))-S1\cdot C2\cdot(-C4\cdot S5\cdot C6+S4\cdot S6)))+D19\cdot(C1 \\ & \cdot S7\cdot(C3\cdot S4\cdot C5-S3\cdot S5)-S1\cdot(S2\cdot(S3\cdot S4\cdot(C5\cdot S7+S5 \\ & \cdot C6\cdot C7)+C7\cdot(-C3\cdot C5\cdot C6+S3\cdot C4\cdot S6))+S7\cdot(-C2\cdot C4 \\ & \cdot C5+S2\cdot C3\cdot S5))+C7\cdot(C1\cdot(C3\cdot C4\cdot S6+C6\cdot(C3\cdot S4\cdot S5 \\ & +S3\cdot C5))-S1\cdot C2\cdot(-C4\cdot S5\cdot C6+S4\cdot S6))); \end{aligned} \tag{47}$$

$$\begin{aligned} & \mathrm{RO22} = S1\cdot S2; \mathrm{RO32} = -C1\cdot S2; \mathrm{RO82} = -S1\cdot C2; \mathrm{RO92} = C1\cdot C2; \\ & \mathrm{RO23} = \mathrm{RO22}\cdot C3 + C1\cdot S3; \mathrm{RO33} = \mathrm{RO32}\cdot C3 + S1\cdot S3; \\ & \mathrm{RO53} = -\mathrm{RO22}\cdot S3 + C1\cdot C3; \mathrm{RO63} = -\mathrm{RO32}\cdot S3 + S1\cdot C3; \\ & \mathrm{RO54} = \mathrm{RO53}\cdot C4 + \mathrm{RO82}\cdot S4; \mathrm{RO64} = \mathrm{RO63}\cdot C4 + \mathrm{RO92}\cdot S4; \\ & \mathrm{RO84} = -\mathrm{RO53}\cdot S4 + \mathrm{RO82}\cdot C4; \mathrm{RO94} = -\mathrm{RO63}\cdot S4 + \mathrm{RO92}\cdot C4; \\ & \mathrm{RO25} = \mathrm{RO23}\cdot C5 - \mathrm{RO84}\cdot S5; \mathrm{RO35} = \mathrm{RO33}\cdot C5 - \mathrm{RO94}\cdot S5; \\ & \mathrm{RO85} = \mathrm{RO23}\cdot S5 + \mathrm{RO84}\cdot C5; \mathrm{RO26} = \mathrm{RO25}\cdot C6 + \mathrm{RO54}\cdot S6; \\ & \mathrm{RO36} = \mathrm{RO35}\cdot C6 + \mathrm{RO64}\cdot S6;\ \mathrm{RO27} = \mathrm{RO26}\cdot C7 - \mathrm{RO85}\cdot S7; \\ & \mathrm{RO87} = \mathrm{RO26}\cdot S7 + \mathrm{RO85}\cdot C7; \mathrm{RL23} = \mathrm{RO22}\cdot D13;\ \mathrm{RL24} = \mathrm{RO23}\cdot D14; \\ & \mathrm{JT34\_1} = \mathrm{RL23} + \mathrm{RL24}; \mathrm{RL25} = \mathrm{RO23}\cdot D15; \mathrm{JT35\_1} = \mathrm{JT34\_1} + \mathrm{RL25}; \\ & \mathrm{RL26} = \mathrm{RO25}\cdot D16; \mathrm{JT36\_1} = \mathrm{JT35\_1} + \mathrm{RL26}; \mathrm{RL27} = \mathrm{RO26}\cdot D17; \\ & \mathrm{RL28} = \mathrm{RO27}\cdot D18 + \mathrm{RO87}\cdot q(8); \mathrm{JT37\_1} = \mathrm{JT36\_1} + \mathrm{RL27}; \\ & \mathrm{JT38\_1} = \mathrm{JT37\_1} + \mathrm{RL28}; \mathrm{RL29} = \mathrm{RO27}\cdot D19; J(3,1) = \mathrm{JT38\_1} + \mathrm{RL29}; \end{aligned} \tag{48}$$

For larger models and in particular for the explicit direct dynamics (37) of constrained multibody systems, the advantage of the recursive differentiation is amazing. In fact, the *explosive* increase in size of the classical differentiation technique (based on partial differentiation of interwoven functions) is quite understandable since it amounts to destroying the recursivity of the original scheme, leading to an in extenso formulation. Although the proposed recursive differentiation process is very consuming in terms of both memory storage and symbolic CPU time – because thousands of "potential" total derivatives are computed, these drawbacks are negligible in ROBOTRAN since the storage requirement is drastically controlled during the symbolic process as explained in Sect. 3.4.

4 ROBOTRAN computer framework

From the practical point of view, modeling a physical system using the multibody approach in ROBOTRAN involves several steps that can be summarized as follows:

– *drawing the multibody system*, which consists in defining the system topology (body structure, connecting

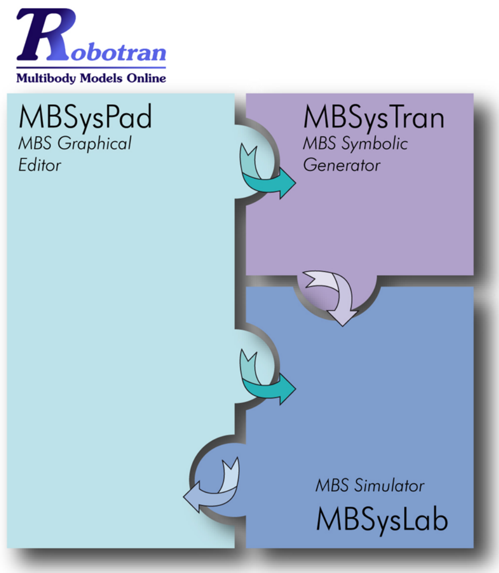


Figure 11. Illustration of the programs composing the ROBOTRAN framework.

joints, external/internal force, loop constraints, etc.) and the numerical data,

– *writing the multibody equations*, which relies on the symbolic engine for the equation of motion (using the Newton/Euler recursive formulation or the Virtual Power Principle) and requires a user intervention for specific constitutive law,

– *simulating the multibody model*, which requires the use of numerical methods so as to exploit the symbolic equations (for instance, numerical integration algorithm for a direct dynamic problem),

– *analyzing the multibody simulation results*, which needs efficient tools for presenting numerical results clearly or producing 3-D animation of the complete system.

In order to go through those various stages, the ROBOTRAN software is composed of several computer programs that are strongly related one to each other as illustrated in Fig. 11 and explained in the following subsections.

4.1 Drawing the multibody system

The first step of any multibody modeling process consists in identifying the involved bodies and the joints which connect them. This often requires a "pre-process" engineering work performed independently of any software. The main originality of ROBOTRAN is to introduce the system data and topology as it would be drawn on a sheet of paper using simple "potato" shapes like in Fig. 2. This way of thinking has guided the design of the graphical editor *MBsysPad*. It can be described as a "topology oriented" graphical editor (instead of a "3-D CAD oriented" editor which comes with most of commercial multibody programs): it relies on a 2-D graphical representation which highlights the MBS topology of the system rather than its 3-D representation. For instance, Fig. 12 illustrates the MBsysPad 2-D diagram of a 5-point suspension quarter car model. This 2-D sketch is composed of several components.

– *Bodies* are represented by various 2-D shapes that can be chosen so as to ensure the readability of the diagram. Specific points of a body are introduced using *anchor points* (arrows in the 2-D diagram).

– ROBOTRAN defines six simple *Joints* with 1 d.o.f.: 3 rotational joints about x-, y- or z-axis and 3 translational joints along x-, y- or z-axis. Combining several single joints permits to model any kind of complex joints. For instance, in Fig. 12, the 4 arms are connected by a *R1–R3* joint sequence which represents a universal joint. In such a way, the relative d.o.f. between two bodies appear directly and explicitly on the 2-D diagram.

– *Links* define point-to-point forces between two bodies (represented by a spring on the 2-D sketch).

– *Cuts* imposes a constraint between two bodies in order to deal with system with kinematic loops. For instance, in Fig. 12, the ball joints between the wheel carrier and the arms are modeled with a *ball cut* that ensures that the two connected points always coincide by imposing 3 algebraic constraints.

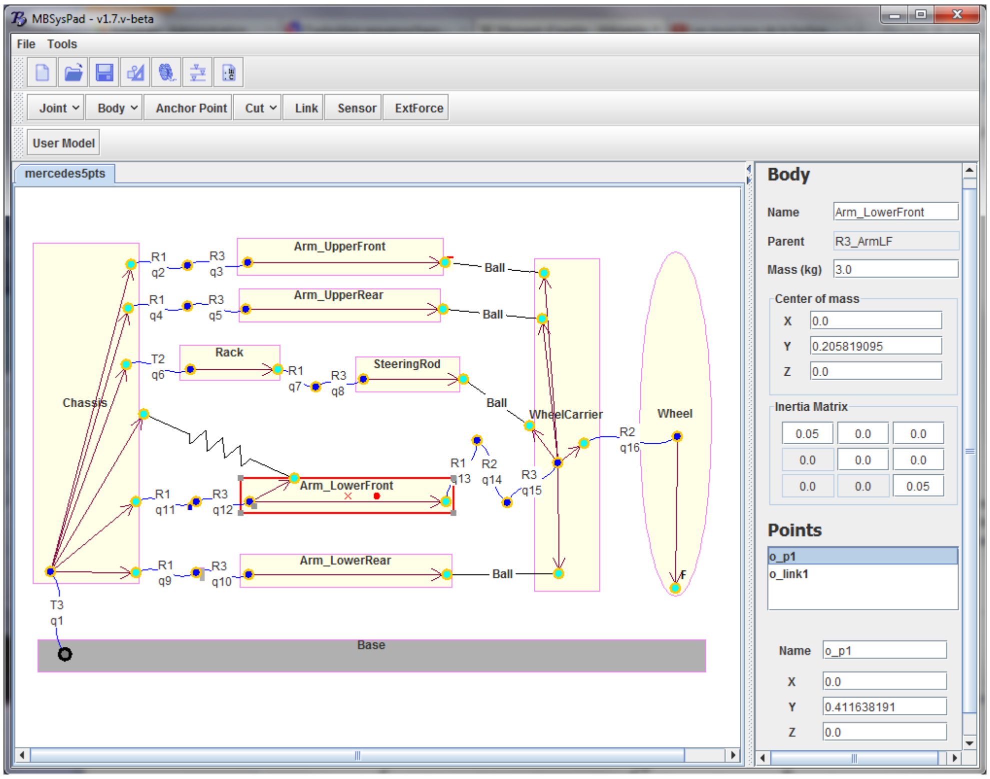
– *External force* or torque can be imposed on the system ("F" symbol in the diagram).

This 2-D representation gives a straightforward access to the element properties that can be modified via an edition panel such as the one appearing in Fig. 12 (on the right) for editing the body properties.

This approach, specific to ROBOTRAN, strengthens the software ergonomics, focusing on the work on the model itself, rather than on "cosmetic" features, by making the tree-like structure of the MBS, loop closure constraints and internal or external forces appearing clearly and explicitly on a single view of the system. Nevertheless, a 3-D representation can also be built in parallel so as to obtain a global view of the model which may be useful for instance for producing 3-D animation of the simulation results (see Fig. 13 which illustrates the 3-D representation of the 5-point suspension).

4.2 Writing the multibody equations

The process of writing the equations can be divided into two steps. The first one refers to the symbolic generation of the equations of motion of the MBS. The second one consists



Body

Name: Arm_LowerFront
Parent: R3_ArmLF
Mass (kg): 3.0

Center of mass
X: 0.0
Y: 0.205819095
Z: 0.0

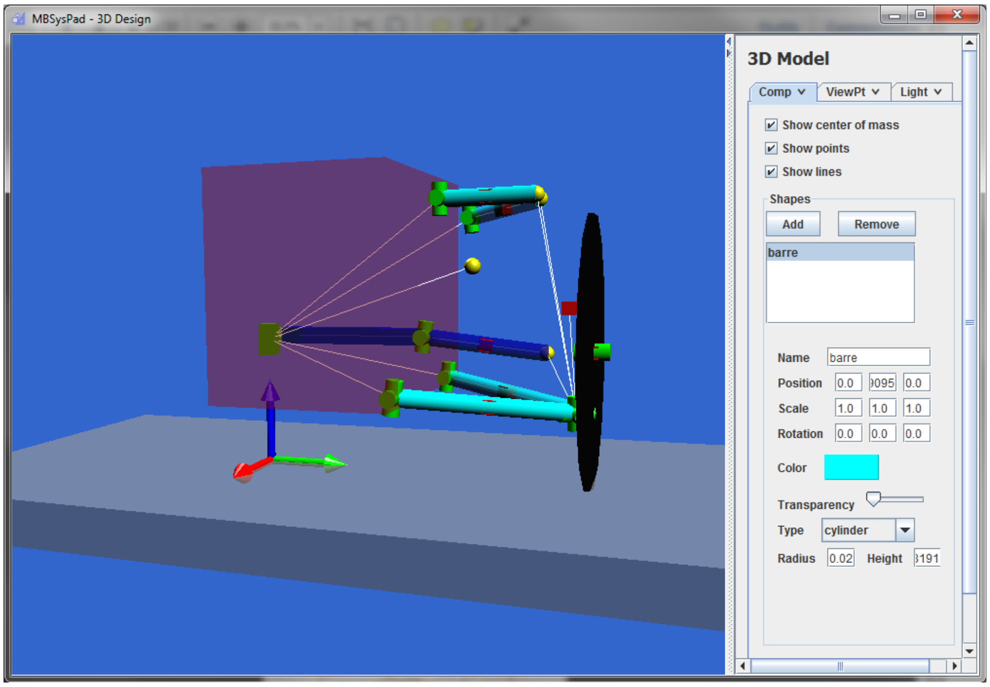
Inertia Matrix

0.05	0.0	0.0
0.0	0.0	0.0
0.0	0.0	0.05

Points
o_p1
o_link1

Name: o_p1
X: 0.0
Y: 0.411638191
Z: 0.0

Figure 12. Representation of a 5 points suspension model in MBsysPad 2-D diagram.



3D Model

☑ Show center of mass
☑ Show points
☑ Show lines

Shapes
barre

Name: barre
Position: 0.0 | 095 | 0.0
Scale: 1.0 | 1.0 | 1.0
Rotation: 0.0 | 0.0 | 0.0
Color:
Transparency
Type: cylinder
Radius: 0.02 Height: 191

Figure 13. Representation of a 5 points suspension model in MBsysPad 3-D view.

in writing constitutive laws which are specific to the application, using symbolic ingredients produced by the previous step.

4.2.1 Symbolic equations

The symbolic equations are generated automatically on the basis of the symbolic formalism described in Sect. 3. Practically, this symbolic generation process (*MBsysTran* in Fig. 11) is performed online via the ROBOTRAN web server and does not require any additional software on the client computer. The symbolic equations consist in a set of C-code or M-code functions for calculating the various matrices and vectors of the equations of motion (i.e., mass matrix, dynamic vector, constraint vector, Jacobian matrix, etc. (see Eqs. 1, 5, 18 or Eqs. 6–9)).

The symbolic engine also generates helpful functions for specific features such as link forces or external forces. For example, for the link forces (i.e. point-to-point internal forces), a symbolic function calculates all the kinematics of the link (distance and distance time variation between the connected points) and performs all the operations required for projecting the force into the joint coordinate space. The user can thus skip this tedious work and concentrate on the tasks specific to his project.

4.2.2 User equations

For introducing force constitutive laws, ROBOTRAN relies on an "open approach" in which the user has the freedom and the responsibility of writing its own *user equations*. For instance, for the 5-point suspension in Fig. 12, the spring-damper element is model by a link force (i.e., point-to-point internal force) for which all the kinematics is calculated by the symbolic process. It remains to implement the suspension constitutive law which can be a simple linear spring-damper equation or a more complex law for which the user can benefit of all the functionalities provided by the language chosen for generating the symbolic files.

This flexible and powerful method applies for writing force constitutive laws (either internal link forces or external forces) but also for imposing the trajectory of a joint, adding specific user constraints or considering additional state equations for mechatronic MBS.

4.3 Simulating and analyzing the multibody system

In order to analyze the multibody model, all equation files (symbolic equations and user equations) must be assembled in a unique program. For this purpose, ROBOTRAN is distributed with a set of Matlab and C functions which constitute the *MBsysLab* environment.

4.3.1 MBsysLab modules for Matlab

The symbolic and user functions written in Matlab language can be used with various modules:

– the *coordinate partitioning module* checks or determines the choice of dependent and independent variables (see Sect. 2.2.1),

– the *equilibrium module* finds the equilibrium position of a given system,

– the *direct dynamics module* performs a time integration of the equations of motion,

– the *inverse dynamics module* calculates the joint forces for a given trajectory,

– the *modal analysis module* determines the eigen modes of a linearized multibody model.

This approach is very efficient for the model prototyping since it allows to benefit of the Matlab language flexibility and to call functions provided by other Matlab toolboxes. A module can for instance be called by an optimization algorithm or a user function can call any specific Matlab function.

4.3.2 MBsysLab for Simulink

MBsysLab also contains modules for building C-code S-Function in Simulink. In this case, symbolic and user files are written in C language and compiled into a binary file that is embedded in a unique Simulink block. It is thus very straightforward to incorporate the multibody model in a classical Simulink block diagram. This is a very powerful way of dealing with control or robotics applications for example. Furthermore, since all the code is compiled, this approach leads to very high performances in terms of calculation time making possible real time simulations or optimization of complex systems in a user-friendly environment.

Finally, it must be noticed that those modules are not specific to Simulink and can be combined with a own written integration algorithm so as to obtain a simulation tool completely independent of Matlab/Simulink. Additionally, the generated code can be transfered to an onboard card for hardware-in-the-loop control.

5 Illustrative applications

Three applications are shortly described in this section namely, (i) the performance of a modern car equipped with the Kinetic$^{\text{TM}}$-type hydraulic suspension, (ii) the analysis of a truck-mounted attenuator and (iii) the modeling of a grand piano action, to respectively highlight the capabilities (i) to simulate multiphysics systems with real-time performances, (ii) to deal with large multibody systems and (iii) to build complex systems in an open-type environment.

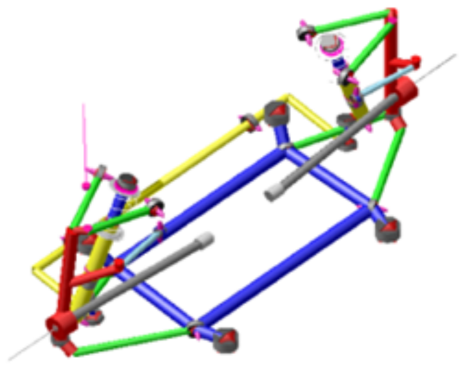


Figure 14. CAD representation of the front wheel-axle unit of the modeled Audi A6.

5.1 A modern car equipped with a Kinetic™ H2 suspension

This first application example deals with the modeling of a full modern car (Audi A6) equipped with multi-link suspensions at front and at rear. Figure 14 shows the front wheel-axle unit consisting of a part of the chassis, the suspension rods, the dampers, the anti-roll bar, the tie rod and the rubber bushings at the various connection. From the multibody modeling point of view, this vehicle is a rather complex model since it involves:

- more than 80 generalized coordinates,
- multi-link suspensions (rear and front), which induce 16 three-dimensional kinematic loops,
- a longitudinallateral wheelground model with saturation effect.

The symbolic equations of this model have been generated in C-language and compiled into a Simulink S-Function so as to perform time-efficient simulation and to easily couple the mechanical model to a hydraulic model in a second step. A line change manoeuvre has been simulated by imposing the motion of the direction rack while the vehicle is running at $10\,\mathrm{m\,s^{-1}}$. The result have been compared to the ones obtained with the multibody software SAMCEF/MECANO. The later relies on a finite-element numerical approach which is a complementary solution to the one proposed by ROBOTRAN, opening the way to the coupling of MBS with structural analysis for instance. As illustrated in Fig. 15 which shows the crushing of the four suspensions during the line change manoeuvre, the two approaches (symbolic generation and relative coordinates for ROBOTRAN, numerical generation and nodal coordinates for SAMCEF/MECANO) give similar results which reinforce the confidence in our model.

The next step has consisted in modeling the *Kinetic H2* suspension system developed by Tenneco Automotive. This innovative device consists in replacing the classical hydraulic

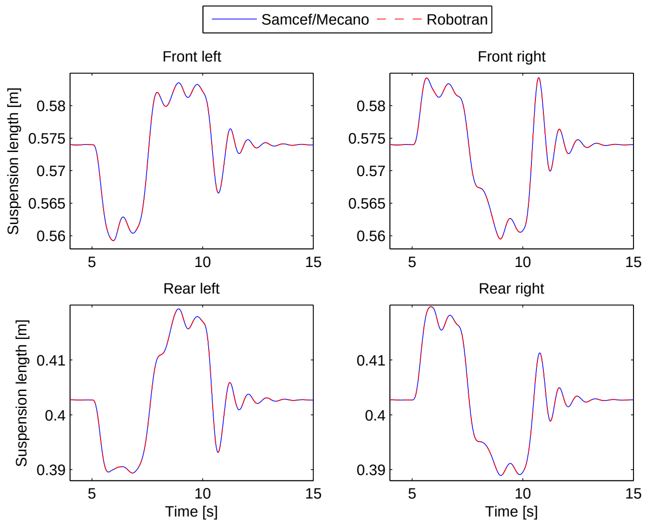


Figure 15. Suspension crushing during a line change manoeuvre.

dampers by double acting cylinders connected together by two distinct hydraulic circuits. This system breaks the tradeoff between a high roll stiffness and a low warp stiffness, as demonstrated by Docquier et al. (2010).

From a practical point of view, the hydraulic model, which involves 22 state equations, has been implemented in C-language. As illustrated in Fig. 16, two options have been retained for coupling it to the MBS. Firstly, the *H2* model has been implemented as a separated Simulink S-Function and coupled a posteriori to the multibody model of the car. Secondly, the hydraulic state equations have been combined to the mechanical state equations and compiled in a unique S-Function, giving one monolithic set of equation to the Simulink time integrator. As shown in Fig. 17, the roll angle of the car during the line change manoeuvre is smaller when the car is equipped with the Kinetic H2 system. Furthermore, the two approaches for coupling the hydraulic and mechanical models result in exactly the same results. However, as illustrated in Table 1, when using the ode45 time integrator of Simulink based on the Dormand-Prince algorithm, the strongly coupled permits larger time steps, resulting in a smaller simulation time. When using the ode15s integrator, the number of time steps is smaller in the case of the strong coupling but the simulation time is equivalent while the integrator does not converge in the case of the weak coupling (2 block diagram).

5.2 A truck-mounted attenuator

This project has consisted in testing and modelling Truck Mounted Attenuators (TMA) made of recycled materials, proposed by the ArGEnCo Laboratory of the Université de Liège (ULg, Belgium). Such a device is used on motorway in order to protect people on working site from inattentive drivers. The principle consists in assembling several cubic

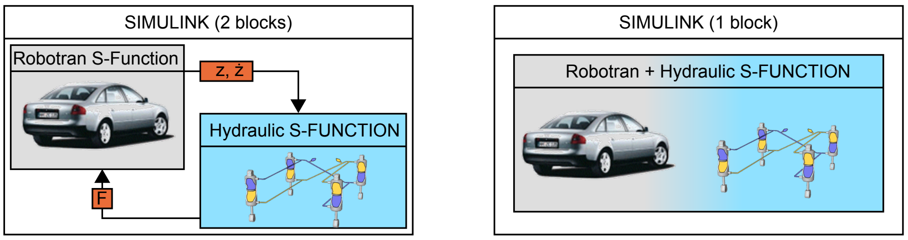


Figure 16. Illustration of the two approaches for coupling the hydraulic and mechanical models. Left: weak coupling using two separated blocks. Right: strong coupling using one monolithic block.

Table 1. Calculation performance for the line change manoeuvre simulation of the model of the Audi A6 equipped with the Kinetic H2 suspension. The simulation was performed on a computer running with Windows 7 and equipped with an Intel Core 2 Duo CPU (2.53GHz), 4 GB RAM.

	ode45 integrator		ode15s integrator	
	weak coupling (2 blocks)	strong coupling (1 block)	weak coupling (2 blocks)	strong coupling (1 block)
Simulation time	66 s	38 s	failed	40 s
Number of time steps	13 754	7946		5767

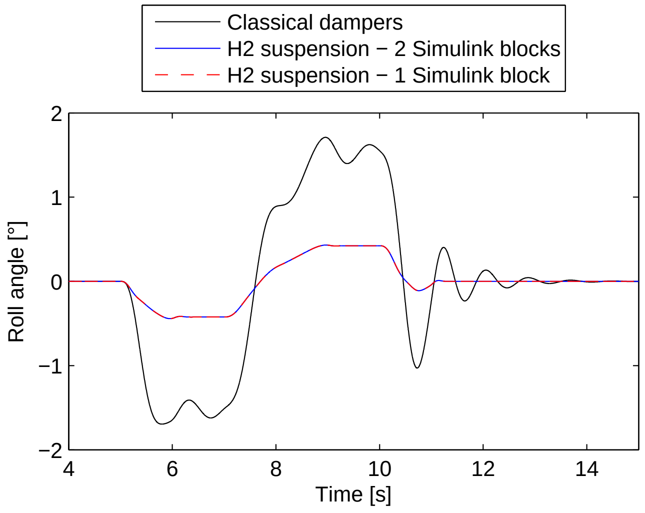


Figure 17. Impact of the Kinetic H2 suspension on the roll angle during the line change manoeuvre.

boxes made of deployed steel containing the recycled materials compacted to an initial preload. Thanks to experimental tests performed at the ULg laboratory, an analytic formulation was established in order to describe the various phases of the material constitutive law: the elastic loading phase, the plastic loading phase and the elastic unloading phase. Then, a multibody model of the TMA has been implemented in ROBOTRAN using a lumped mass approach: each block is splitted into several bodies linked together by prismatic joints. The constitutive law is applied via external forces acting on each bodies, following the action/reaction principle.

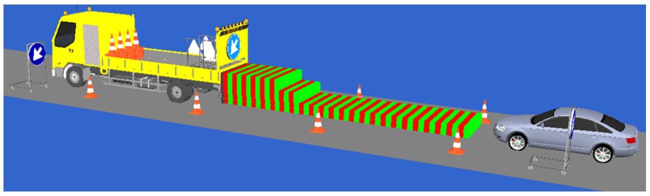


Figure 18. MBS model of a truck-mounted attenuator (TMA).

By comparing models involving 1, 3 and 10 bodies for one block, we observe discrepancies during the collision (oscillations) but a good match in terms of maximum deceleration of the impacting mass (600 kg), as illustrated in Fig. 19 (more details can be found in Abedrabbo et al., 2011). Also, comparing those results in terms of the Acceleration Severity Index (ASI), which is the reference for crash calculations, shows differences that are lower than 0.4 % between the different models, and 7.4 % with experiments which is really acceptable.

Afterward, a realistic TMA was designed for absorbing the shock between a truck and a car running at 90 km h^{-1} or a bus running at 70 km h^{-1}. In order to analyse the TMA performance during the collision, the MBS model of the TMA described above has been used to simulate a longitudinal impact between a car (or a bus) and this TMA carried by a truck, leading to a model containing more than 300 degrees of freedom. The truck, the car and the bus, being more rigid than the attenuator modules, were modeled as rigid MBS, with articulated suspension and wheels. Two deformation laws of the front part of each impacting vehicle (i.e., the car or the bus)

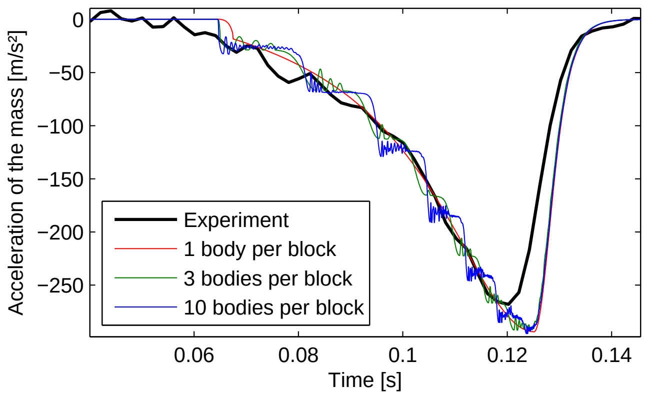


Figure 19. Influence of the number of bodies used to discretize a TMA block and comparison with experimental results.

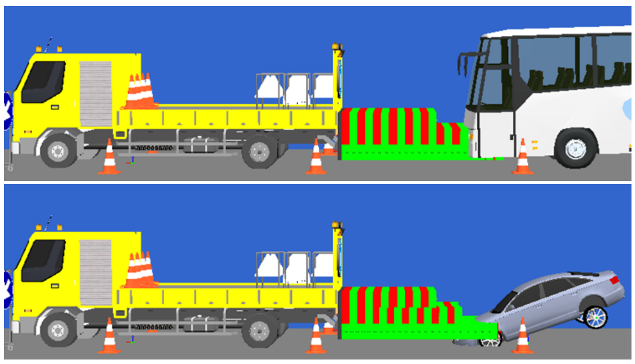


Figure 20. Simulation of the collision of a bus or a car into the TMA.

have been considered, the second one being stiffer than the first one.

Several parameters have been investigated such as:

– the influence of the own mass of the TMA blocks on the ASI measured at the driver head level,

– the influence of the deformation law of the front of the impacting vehicle,

– the influence of the road adherence conditions which impact the safety of workers located in front of the truck.

As an illustrative result, Figs. 20 and 21 illustrate a typical ROBOTRAN simulation and the ASI measured during the crash for various values of the coefficient of friction f between the ground and the wheels of the truck carrying the TMA. The impacting vehicle is supposed to slide perfectly on the ground. The extreme case where the truck is completely locked on the ground is also considered. It clearly appears that the ASI is higher for the car than for the bus, due to its smaller mass. The impact of the friction coefficient is smaller but the diagram reveals that the truck should have the possibility to move with respect to the ground.

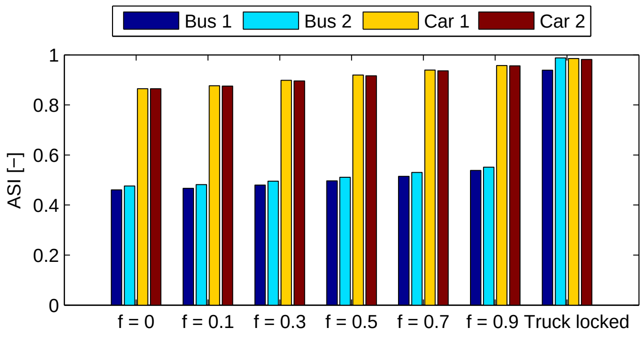


Figure 21. Impact of the friction coefficient f between the ground and the wheels of the truck. The Bus 2 and Car 2 cases correspond to the simulation of a vehicle with a front deformation law stiffer than the Bus 1 and Car 1 cases.

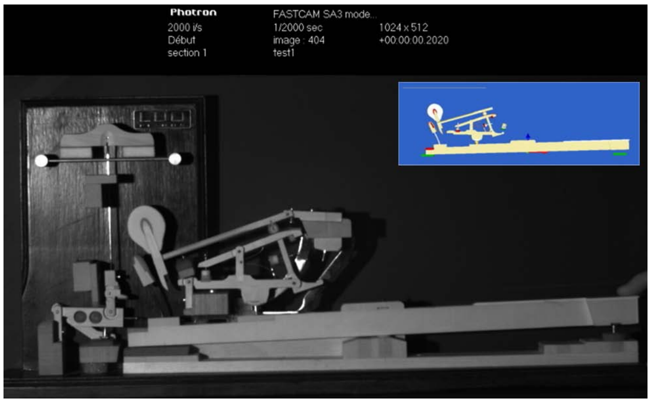


Figure 23. Model versus experiment: full system in action.

5.3 A grand piano action

The goal of the underlying project – presently in progress in close collaboration with the MIM, the Musical Instruments Museum of Brussels – has a double nature: organological and didactic. Indeed, using a multibody approach, we have carried out a virtual demonstrator of a grand piano action mechanism (see Fig. 22) in order to understand, demystify and parameterize the "from key-to-hammer" transmission, and especially the double escapement principle patented by the French Sébastien Erard in 1821. Double escapement actions allow notes to be repeated more easily than in single piano actions.

As one might imagine, a particular attention has been paid to the modeling of the intermittent contacts (they are twelve in number in the mechanism!) for both the geometrical (shapes) and dynamic (constitutive laws) points of view.

To ensure a reasonable match between the multibody model and the real mechanism, the force law parameters have been tuned via experimental validations using a high-speed camera (2000 black and white frame/sec): they have been carried out for the whole system in action (Fig. 23) or for some specific sub-systems, like a single oscillating hammer

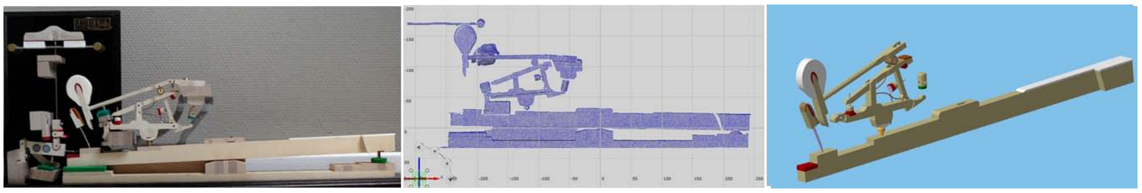
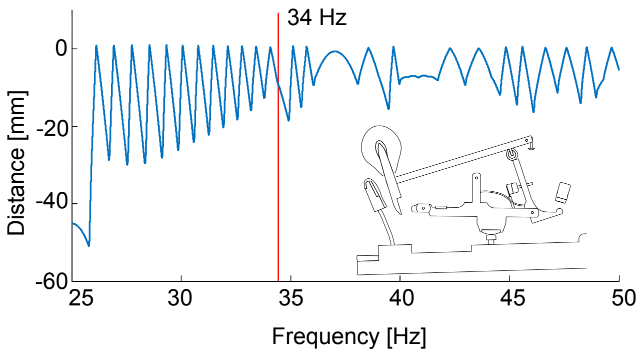


Figure 22. Real action mechanism (Left) – Scan of the action (Center) – Multibody model (Right).



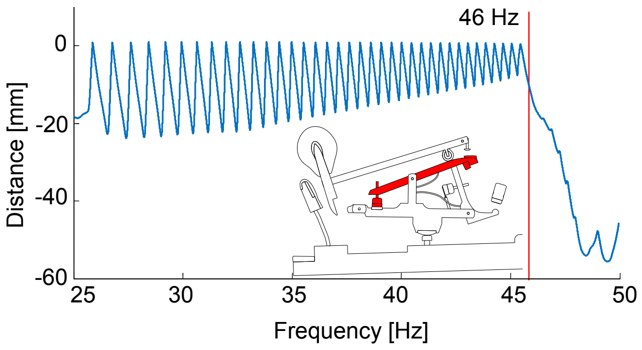


Figure 24. Single (top) versus double escapement (bottom) comparison.

(not shown). After identification, a satisfactory correspondence was obtained, considering the objectives of the project. Let us briefly show and comment on a significant result related to the comparison between single and double escapement configurations. The analyzed piano action mechanism is a modern one (Fig. 22) with the double escapement system, which allows the pianist to repeat a given note with a shorter stroke (for the key and the finger) and thus at a higher speed. By virtually "removing" some parts in the model, the double escapement can be virtually deactivated. To compare the dynamic capabilities of the two mechanisms, a staccato-type sinusoidal input with increasing frequency is applied on the key front. In Fig. 24, we plot the vertical distance [m] between the top of the hammer and the string, versus the blow frequency [Hz]. This result clearly shows that single escapement is unable to correctly repeat the motion above 34 Hz, while the original system, thanks to the double escapement principle, can reach 46 Hz (more details in Bokiau et al., 2012). This kind of result is greatly appreciated by our collaborators at the MIM, because such a comparison appears to be quite instructive and also difficult to observe experimentally.

6 Conclusions

In this paper, the main assets and recent developments of ROBOTRAN software are reviewed. The symbolic generation capabilities of the software have been greatly enhanced to deal with large models (in terms of d.o.f.) whose symbolic generation requires less than one second via a web server. The fully symbolic generation of constrained MBS is a new feature of the program that notably improves the CPU time performances but, above all, allows us to provide a completely free-standing symbolic function (in C, Matlab, etc.) ready for use for various scientific computer environments. More recently, the recursive symbolic differentiation of MBS direct dynamics with possible nonlinear constraints has been implemented and has revealed its superiority with respect to standard symbolic engines in terms of symbolic simplification and equation conciseness. A novel user interface based on an intuitive 2-D representation of the MBS – the 3-D representation being automatically constructed in the background – and the open-type architecture of the program are presented. Finally, three illustrative applications are briefly discussed to highlight the capabilities of ROBOTRAN in building complex and large models and dealing with multiphysics applications.

For the next future, the main developments will mainly concern the periphery of the ROBOTRAN symbolic generator because we wish to keep the latter as open as possible for the user, and to interface the symbolic models with specific external software. These couplings refer to multiphysics modeling (in the continuity of our present research work), the geometrical and dynamic contact between 3-D profiled bodies with a real-time approach, the coupling with FEM models (flexible bodies), CFD software (multibody-fluid interaction) and DEM software (discrete element modelling for granular materials). As regards flexible bodies, our past investigations for beams (Fisette et al., 1997) and plates (El Ouatouati et al., 1999) have shown that in case of small deformations, the floating-frame approach in relative coordinates was a suitable option for symbolic generation, leading to very efficient models: the revival of those models is part of our future objectives for the symbolic core of ROBOTRAN.

Acknowledgements. This research work has been supported by the Belgian National Fund for Scientific Research (F.R.S.-FNRS) which is acknowledged.

Edited by: O. Brüls

References

Abedrabbo, G., Poncelet, A., Sepulveda, P., Cescotto, S., and Fisette, P.: Multibody Simulation of a Crash Test Attenuator Made of Recycled Materials, in: Proceedings of the 17th International Symposium on Plasticity & Its Current Applications, Mexico, 2011.

Bokiau, B., Poncelet, A., Fisette, P., and Docquier, N.: Multibody Model of a Grand Piano Action Mechanism Aimed at Understanding and Demystifying the Escapement Principle, in: Proceedings of the 2nd Joint International Conference on Multibody System Dynamics, Stuttgrart, Germany, 2012.

Chenut, X., Fisette, P., and Samin, J.-C.: Recursive Formalism with a Minimal Dynamic Parameterization for the Identification and Simulation of Multibody Systems. Application to the Human Body, Multibody Syst. Dyn., 8, 117–140, doi:10.1023/A:1019555013391, 2002.

Ding, J.-Y., Pan, Z.-K., and Chen, L.-Q.: Second order adjoint sensitivity analysis of multibody systems described by differential–algebraic equations, Multibody Syst. Dyn., 18, 599–617, doi:10.1007/s11044-007-9080-4, 2007.

Docquier, N., Poncelet, A., Delannoy, M., and Fisette, P.: Multiphysics Modeling of Multibody Systems: Application to Car Semi-Active Suspensions, Vehicle Syst. Dyn., 48, 1439–1460, 2010.

Eberhard, P.: Analysis and optimization of complex multibody systems using advanced sensitivity analysis methods, ZAMM Zeitschrift fur Angewandte Mathematik und Mechanik, 76, 40–43, 1996.

El Ouatouati, A., Fisette, P., and Johnson, D.: A Fully Symbolic Model of Multibody Systems Containing Flexible Plates, Nonlinear Dynamics, 18, 357–382, 1999.

Fisette, P.: Génération symbolique des équations du mouvement de systèmes multicorps et application dans le domaine ferroviaire, Ph.D. thesis, Université catholique de Louvain, 1994.

Fisette, P., Johnson, D., and Samin, J.: A Fully Symbolic Generation of the Equations of Motion of Multibody Systems containing Flexible Beams, Comput. Method. Appl. M., 142, 132–152, 1997.

Fisette, P., Postiau, T., Sass, L., and Samin, J.: Fully symbolic generation of complex multibody models, Mech. Struct. Mach., 30, 31–82, doi:10.1081/SME-120001477, 2002.

Ganovski, L., Fisette, P., and Samin, J.-C.: Piecewise Overactuation of Parallel Mechanisms Following Singular Trajectories: Modeling, Simulation and Control, Multibody Syst. Dyn., 12, 317–343, doi:10.1007/s11044-004-2532-1, 2004.

Kecskeméthy, A.: Mobile – An object-oriented tool-set for the efficient modeling of mechatronic systems, in: Proceedings of the Second Conference on Mechatronics and Robotics, 27–29 September, 27–29, 1993.

Kecskeméthy, A., Krupp, T., and Hiller, M.: Symbolic Processing of Multiloop Mechanism Dynamics Using Closed-Form Kinematics Solutions, Multibody Syst. Dyn., 1, 23–45, doi:10.1023/A:1009743909765, 1997.

Kurz, T., Eberhard, P., Henninger, C., and Schiehlen, W.: From Neweul to Neweul-M2: symbolical equations of motion for multibody system analysis and synthesis, Multibody Syst. Dyn., 24, 25–41, doi:10.1007/s11044-010-9187-x, 2010.

Maes, P., Samin, J.-C., and Pierre, W.: Multibody Systems Handbook, chap. Robotran, 246–264, Springer-Verlag, 1990.

Poncelet, A., Collard, J.-F., and Fisette, P.: Parameter Sensitivity Analysis: Symbolic Computation for Large Multibody Systems, in: Proceedings of the First Joint International Conference on Multibody System Dynamics, Lappeenranta, Finland, 2010.

Postiau, T., Sass, L., Fisette, P., and Samin, J.-C.: High-Performance Multibody Models of Road Vehicles: Fully Symbolic Implementation and Parallel Computation, Vehicle Syst. Dyn., 35, 57–83, 2001.

Raison, M., Aubin, C., Detrembleur, C., Fisette, P., Mahaudens, P., and Samin, J.: Quantification of global intervertebral torques during gait: comparison between two subjects with different scoliosis severities, Studies in Health Technology and Informatics, 158, 107–111, 2010.

Samin, J. and Fisette, P.: Symbolic modeling of multibody systems, Springer, 2003.

Samin, J.-C., Brüls, O., Collard, J.-F., Sass, L., and Fisette, P.: Multiphysics modeling and optimization of mechatronic multibody systems, Multibody Syst. Dyn., 18, 345–373, doi:10.1007/s11044-007-9076-0, 2007.

Schiehlen, W.: Multibody Systems Handbook, Springer-Verlag, Berlin, Germany, 1990.

Schwertassek, R. and Rulka, W.: Real-Time Integration Methods for Mechanical System Simulation NATO ASI Series, Series F, vol. 69, chap. Aspects of Efficient and Reliable Multibody Systems Simulation, Springer-Verlag, Berlin, Germany, 1989.

Shi, P. and McPhee, J.: Dynamics of Flexible Multibody Systems using Virtual Work and Linear graph Theory, Multibody Syst. Dyn., 4, 355–381, 2000.

Wehage, R. A. and Haug, E. J.: Generalized Coordinate Partitioning for Dimension Reduction in Analysis of Constrained Dynamic Systems, J. Mech. Design, 104, 247–255, doi:10.1115/1.3256318, 1982.