Open source symbolic and numerical tools for the simulation of multibody systems

O. Verlinden, G. Kouroussis, S. Datoussaïd, C. Conti
Faculté Polytechnique de Mons
Boulevard Dolez 31 - 7000 Mons (BELGIUM)
Olivier.Verlinden@fpms.ac.be

Abstract—**This paper presents a framework, called `EasyDyn`, developed for the simulation of dynamics problems and in particular, multibody systems. The system is not classically defined from a data file but from a C++ file where the user must program the kinematics and the expression of applied forces acting on each body. By kinematics, we mean the expression of position, velocity and acceleration of each body in terms of generalized coordinates. From the kinematics and the efforts, `EasyDyn` automatically builds and integrates the equations of motion, according to methods described in the paper.**

To help the user in his task, the library provides on one hand object-oriented classes for vector algebra (vectors, rotation and inertia tensors, homogeneous transformation matrices) so that vector expressions can be written as is in the C++ code without having to manage the coordinates. On the other hand, a `MuPAD` script is distributed with the library to automatically generate a core C++ program from a minimal information. In particular, the script uses the symbolic features of `MuPAD` to derive the expressions of velocities and accelerations from only the position information. The user just needs to add the contribution of the efforts in the C++ initial code, which is made easier by the vector library and the availability of some basic routines.

The library and all the tools it relies on are available for free on the internet and can be used either under Windows or Unix. It was not designed as a replacement of commercial softwares. The latter are developed for the purpose of efficiency and productivity for high-end applications. On the contrary, `EasyDyn` has been written so as to get the maximum compacity, readability, and scalability of the code, often to the detriment of efficiency. It is particularly well-suited for teaching and, as it is open source, offers a foundation for collaborative or research work.

Keywords— **Dynamics, simulation, second-order differential equations, multibody systems, object-oriented programming, open source, symbolic**

I. Introduction

Commercial tools like `ADAMS`, `LMS/DADS` or `SIMPACK` are now commonly used in industry to simulate the kinematic and dynamic behaviour of multibody systems like manipulators, aircrafts, vehicles or complex articulated mechanisms. Those tools are powerful, user-friendly and efficient but represent a human and financial investment that the industry cannot afford if the application is not inside its core business. Moreover, as the source code is not available, the user is highly dependent on the editor for any adaptation of the program.

As an alternative, some simulation codes like `DynaFlex`, `Dynamechs`, or `MBDyn`, unfortunately not sufficiently known, can be downloaded from the net and used for free as a replacement of their commercial counterpart. These tools have the advantage to be open source [1], and can consequently be adapted by the community of users according to their needs. In this paper, we describe another open source solution, consisting of a C++ library called `EasyDyn`, where the system is not defined in a data file but as a C++ program from the description of the kinematics and the applied forces. The library then provides the routines to build and integrate the equations of motion. Besides the C++ library, a `MuPAD` script has been written to generate symbolically the complete kinematics (velocities and accelerations) from only the position information.

The major purpose of the approach is to give to the user the maximum control. The library, initially developed for teaching [6] is easily readable, relies only on software available for free on the net and can be used under Windows or Unix. Moreover, as the source is available, `EasyDyn` can be adapted to any particularity of the application.

In this paper, we first describe the theoretical background of the library and then present the principal implementation details. The potentialities of the library are finally illustrated through some examples.

II. Description of the system

A multibody system consists of rigid or flexible bodies connected to each other by kinematic joints and/or element forces like springs or dampers. We will consider that the system comprises n_B bodies whose spatial situation is described by n_{cp} configuration parameters gathered in a vector denoted $\boldsymbol{q}$.

The configuration parameters are chosen arbitrarily, in a number corresponding to the number of degrees of freedom. The approach is said to be in *minimal* or *generalized* coordinates [3].

A local reference frame i composed of unit vectors $\boldsymbol{x}_i$, $\boldsymbol{y}_i$, $\boldsymbol{z}_i$ is attached to body i and will be used to express the spatial situation of the body. Although it is not necessary, we will assume that it is located on the center of gravity.

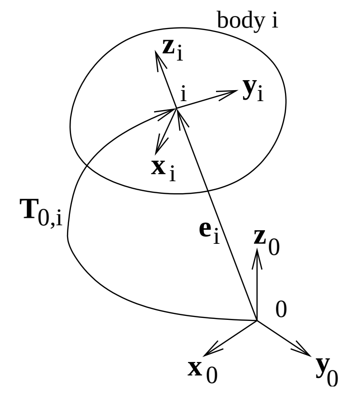

Fig. 1. Reference frame of a body

Body 0 is assumed to be fixed. The related reference frame $\boldsymbol{x}_0$, $\boldsymbol{y}_0$, $\boldsymbol{z}_0$ will be used as the global reference frame.

The situation of each body will be expressed by means of the homogeneous transformation matrix $\boldsymbol{T}_{0,i}$, giving the situation of the local frame i with respect to the global frame in terms of the configuration parameters $\boldsymbol{q}$. Let us recall that this matrix has the following form

$$\boldsymbol{T}_{0,i}(\boldsymbol{q}) = \begin{pmatrix} \boldsymbol{R}_{0,i}(\boldsymbol{q}) & \{\boldsymbol{e}_i(\boldsymbol{q})\}_0 \\ 0\ 0\ 0 & 1 \end{pmatrix} \tag{1}$$

where
• $\boldsymbol{e}_i$ is the coordinate vector of frame i with respect to the global reference frame 0 ($\{\boldsymbol{e}_i\}_0$ being the 3x1 matrix gathering the components of the coordinate vector $\boldsymbol{e}_i$ expressed in the coordinate system of frame 0);
• $\boldsymbol{R}_{0,i}$ is the rotation tensor describing the orientation of frame i with respect to frame 0.

Physically the columns of $\boldsymbol{R}_{0,i}$ correspond to the unit vectors $\boldsymbol{x}_i$, $\boldsymbol{y}_i$ and $\boldsymbol{z}_i$, expressed in the axes of frame 0

$$\boldsymbol{R}_{0,i} = (\{\boldsymbol{x}_i\}_0\ \{\boldsymbol{y}_i\}_0\ \{\boldsymbol{z}_i\}_0) \tag{2}$$

As an example, let us consider the double pendulum illustrated in figure 2. It is composed of 2 bodies, each one with its own frame. Two revolute joints constraint the motion of the bodies, on O between the ground and body 1 and on A between bodies 1 and 2. It is easy to figure out that the system has 2 degrees of freedom so that the configuration of the system can be univoquely defined from angles q_1 and q_2 indicated on the figure.

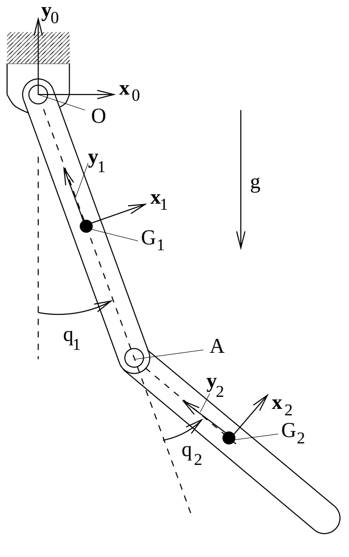

Fig. 2. Double pendulum

The homogeneous transformation matrices giving the situation of the two bodies can be easily established as

$$\boldsymbol{T}_{0,1} = \begin{pmatrix} c_1 & -s_1 & 0 & \frac{l_1}{2}s_1 \\ s_1 & c_1 & 0 & -\frac{l_1}{2}\,c_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3}$$

$$\boldsymbol{T}_{0,2} = \begin{pmatrix} c_{12} & -s_{12} & 0 & l_1 s_1 + \frac{l_2}{2}s_{12} \\ s_{12} & c_{12} & 0 & -l_1 c_1 - \frac{l_2}{2}\,c_{12} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4}$$

with l_1 and l_2 the lengths of the arms, $c_1 = \cos(q_1)$, $s_1 = \sin(q_1)$, $c_{12} = \cos(q_1 + q_2)$ and $s_{12} = \sin(q_1 + q_2)$.

We will denote $\boldsymbol{v}_i$ and $\boldsymbol{\omega}_i$ the translation and rotation velocities of the reference frame of a body i. They are expressed in terms of the configuration parameters $\boldsymbol{q}$ and their time derivatives $\dot{\boldsymbol{q}}$ upon which they depend linearly as

$$\boldsymbol{v}_i = \sum_{j=1}^{n_{cp}} \boldsymbol{d}^{i,j} \cdot \dot{q}_j \qquad \boldsymbol{\omega}_i = \sum_{j=1}^{n_{cp}} \boldsymbol{\theta}^{i,j} \cdot \dot{q}_j \tag{5}$$

The vectors $\boldsymbol{d}^{i,j}$ and $\boldsymbol{\theta}^{i,j}$ represent the partial contributions of parameter q_j in the translation and rotation velocities of frame i.

In the same way, the translation and rotation accelerations of the reference frame of body i will be denoted $\boldsymbol{a}_i$ and $\dot{\boldsymbol{\omega}}_i$ and will be expressed in terms of the configuration parameters $\boldsymbol{q}$ and their first and second time derivatives $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$.

III. Equations of motion

When a multibody system is described in terms of generalized coordinates (without constraints), the application of the d'Alembert's principle [3], leads to the following generalized equations of motion

$$\begin{aligned} &\textstyle\sum_{i=1}^{n_B} \boldsymbol{d}^{i,j} \cdot (\boldsymbol{R}_i - m_i \boldsymbol{a}_i) \\ &+\boldsymbol{\theta}^{i,j} \cdot (\boldsymbol{\mathcal{M}}_i - \boldsymbol{\Phi}_{G_i}\dot{\boldsymbol{\omega}}_i - \boldsymbol{\omega}_i \times \boldsymbol{\Phi}_{G_i}\boldsymbol{\omega}_i) = 0 \quad j = 1, n_{cp} \end{aligned} \tag{6}$$

with
• m_i and $\boldsymbol{\Phi}_{G_i}$ the mass and the central inertia tensor of body i;
• $\boldsymbol{R}_i$ and $\boldsymbol{M}_{Gi}$ the resultant force and moment, at the center of gravity G_i, of all applied efforts exerted on body i.

The n_{cp} resulting equations of motion are of the form

$$\boldsymbol{M}(\boldsymbol{q}) \cdot \ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) = 0 \tag{7}$$

with
• $\boldsymbol{M}$ the mass matrix of dimension n_{cp}xn_{cp}, defined by

$$\boldsymbol{M}_{jk} = \textstyle\sum_{i=1}^{n_B} m_i \boldsymbol{d}^{i,j} \cdot \boldsymbol{d}^{i,k} + \boldsymbol{\theta}^{i,j} \cdot (\boldsymbol{\Phi}_{G_i} \cdot \boldsymbol{\theta}^{i,k}) \tag{8}$$

• $\boldsymbol{h}$ a general term gathering the centrifugal and Coriolis terms and the contribution of the applied efforts.

IV. Integration of second-order differential equations

Although it is usual to transform second order differential equations to their equivalent first-order form, we prefer here to work with the natural second-order form of the equations of motion

$$\boldsymbol{f}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, t) = \boldsymbol{0} \tag{9}$$

To perform an integration step from time t to time $t+h$, so-called integration formulas are introduced, of the form

$$\dot{\boldsymbol{q}}_i^{t+h} = \check{\Lambda}(\dot{\boldsymbol{q}}_i^{\leq t}, \ddot{\boldsymbol{q}}_i^{\leq t}, \ddot{\boldsymbol{q}}_i^{t+h}) \tag{10}$$

$$\boldsymbol{q}_i^{t+h} = \Lambda(\boldsymbol{q}_i^{\leq t}\dot{\boldsymbol{q}}_i^{\leq t}, \ddot{\boldsymbol{q}}_i^{\leq t}, \ddot{\boldsymbol{q}}_i^{t+h}) \tag{11}$$

with $\boldsymbol{q}^{\leq t}$, $\dot{\boldsymbol{q}}^{\leq t}$, $\ddot{\boldsymbol{q}}^{\leq t}$ a given number of configurations at and before time t and $\boldsymbol{q}^{t+h}$, $\dot{\boldsymbol{q}}^{t+h}$, $\ddot{\boldsymbol{q}}^{t+h}$ the configuration at time $t+h$.

For example, the Newmark integration formulas are written

$$q^{t+h} = q^t + h\dot{q}^t + (0.5-\beta)h^2\ddot{q}^t + \beta h^2 \ddot{q}^{t+h} \tag{12}$$
$$\dot{q}^{t+h} = \dot{q}^t + (1-\gamma)h\ddot{q}^t + \gamma h \ddot{q}^{t+h} \tag{13}$$

where β and γ are the Newmark parameters ($0.25 \leq \beta \leq 0.5$ and $0.5 \leq \gamma \leq 1$ to assure unconditional stability).

The most often, the integration formulas are implicit which means that they involve the accelerations at time $t+h$. Once positions and velocities have been replaced by the integration formulas, the equations of motion are expressed only in terms of the accelerations at time $t+h$, as

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{q}^{t+h}, \dot{\boldsymbol{q}}^{t+h}, \ddot{\boldsymbol{q}}^{t+h}, t+h) &= \boldsymbol{f}(\boldsymbol{\Lambda}, \check{\boldsymbol{\Lambda}}, \ddot{\boldsymbol{q}}^{t+h}, t+h) \\ &= \boldsymbol{F}(\ddot{\boldsymbol{q}}^{t+h}) = \boldsymbol{0} \end{aligned} \tag{14}$$

Performing a time step then comes down to solving the nonlinear equations $\boldsymbol{F}$ in the accelerations $\ddot{\boldsymbol{q}}^{t+h}$. This operation is generally achieved by means of the iterative procedure of Newton-Raphson, where the nth estimation is calculated from the preceding one as

$$\ddot{\boldsymbol{q}}^{t+h,n} = \ddot{\boldsymbol{q}}^{t+h,n-1} - \boldsymbol{J}^{-1} \cdot \boldsymbol{F}(\ddot{\boldsymbol{q}}^{t+h,n-1}) \tag{15}$$

where $\boldsymbol{J}$ is the jacobian matrix of the equations $\boldsymbol{F}$ with respect to the unknowns $\ddot{\boldsymbol{q}}^{t+h}$. Each term of this matrix, also called *iteration matrix*, is defined by

$$\boldsymbol{J}_{ij} = \boldsymbol{M}_{ij} + \boldsymbol{CT}_{ij} \cdot \frac{\partial \check{\Lambda}}{\partial \ddot{q}^{t+h}} + \boldsymbol{KT}_{ij} \cdot \frac{\partial \Lambda}{\partial \ddot{q}^{t+h}} \tag{16}$$

with $\boldsymbol{M}$ the mass matrix and $\boldsymbol{KT}$ and $\boldsymbol{CT}$ the tangent stiffness and damping matrices, defined by

$$\boldsymbol{KT}_{ij} = \frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{q}_j} \qquad \boldsymbol{CT}_{ij} = \frac{\partial \boldsymbol{f}_i}{\partial \dot{\boldsymbol{q}}_j} \tag{17}$$

V. The EasyDyn library

A. Introduction

Initially developed for teaching [6], the purpose of the `EasyDyn` library is to help users to set up and integrate the equations of motion of a multibody system with a minimal effort. The library, available for free on the internet (http://www.mecara.fpms.ac.be), is written in C++ and provides 4 modules

- the `sim` module to integrate second-order differential equations;
- the `mbs` module, a frontend to `sim` which automatically builds the differential equations of motion of a multibody system from the kinematics and the applied forces;
- the `vec` module, introducing classes related to vector calculus: vectors, rotation tensors, inertia tensors, and homogeneous transformation matrices;
- the `visu` module, allowing to build object-oriented scenes composed of moving objects.

The major objective during development was not the numerical efficiency but the compacity, readability and scalability of the code. For instance, the module `mbs` comprises only 150 lines of code.

B. The `sim` module

The `sim` module consists of routines to integrate second-order differential equations as far as the user provides the routine `ComputeResidual` which yields the residuals of the differential equations in terms of time, the state variables and their first and second time derivatives, that's to say $\boldsymbol{f}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, t)$

The only integration method available so far is based on the Newmark formulas. For the sake of simplicity, the iteration matrix is built from a numerical derivation [5] where the jth column is computed from the variation of the residuals for a variation of the jth configuration parameter

$$\boldsymbol{J}_j = \frac{\boldsymbol{f}(\boldsymbol{q}+\Delta\boldsymbol{q}^j, \dot{\boldsymbol{q}}+\Delta\dot{\boldsymbol{q}}^j, \ddot{\boldsymbol{q}}+\Delta\ddot{\boldsymbol{q}}^j, t) - \boldsymbol{f}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, t)}{\epsilon} \tag{18}$$

where $\Delta\boldsymbol{q}^j$, $\Delta\dot{\boldsymbol{q}}^j$ and $\Delta\ddot{\boldsymbol{q}}^j$ are vectors with all components null but the jth one, given by

$$\Delta\boldsymbol{q}^j_j = \epsilon\beta h^2 \qquad \Delta\dot{\boldsymbol{q}}^j_j = \epsilon\gamma h \qquad \Delta\ddot{\boldsymbol{q}}^j_j = \epsilon \tag{19}$$

The increment ϵ is chosen from the increment δq on positions according to the heuristics used in the well-known integration code DASSL [4]

$$\delta q = \beta h^2 \epsilon = \max(|h\dot{q}|, |q|, \epsilon_a + \epsilon_r|\dot{q}|)\mathrm{sign}(\dot{\mathrm{q}})\sqrt{\mathrm{u}} \tag{20}$$

with u the roundoff error of the computer and ϵ_a and ϵ_r the absolute and relative error tolerances used in the integration process.

The Newmark scheme has an accuracy order equal to 2. The error on positions depends on the third time derivative and is estimated from the variation of acceleration over the time step as

$$\epsilon_q = \frac{h^3}{12}\frac{\partial^3 q}{dt^3} \simeq \frac{h^3}{12}\frac{\Delta\ddot{q}}{h} = \frac{h^2\Delta\ddot{q}}{12} \tag{21}$$

As several configuration parameters are involved, the global error is estimated from the Euclidean norm of the variation of acceleration as

$$\epsilon_q \simeq \frac{h^2\|\Delta\ddot{\boldsymbol{q}}\|}{12\sqrt{n_{cp}}} \tag{22}$$

Practically, the time step is chosen to keep the rate of error ϵ_q/h below a tolerance equal by default to 10^{-6}.

For the sake of readability and efficiency, matrix and vector operations are programmed systematically on the base of the GSL (`GNU` *Scientific Library*), available for free from the web site of the Free Sotware Foundation(`www.gnu.org`).

As an example, let us consider the system represented in figure 3, consisting of an hydraulic jack pushing a mass m, attached to the ground by a spring of stiffness k. The jack comprises two chambers numbered 1 and 2. It is a good test for `EasyDyn` as the hydraulic equations are well-known to be particularly stiff.

The volume flow Q_i entering in each chamber i (i=1,2) verifies on one hand the discharge law through the throttling section S_{ei}

$$Q_i = C_d S_{ei}\sqrt{\frac{2(p_{Ei}-p_i)}{\rho}} * \mathrm{sign}(\mathrm{p_{Ei}} - \mathrm{p_i}) \tag{23}$$

with C_d the orifice discharge coefficient, ρ the fluid density, p_i the fluid pressure inside the chamber, and p_{Ei} the circuit pressure at the entrance of the chamber.

On the other hand, the flow is driven by the variation of volume of the chamber and by the variation of the fluid pressure

$$Q_i = \dot{V}_i + \frac{V_i}{K}\dot{p}_i \tag{24}$$

with V_i the volume of the chamber and K the compressibility coefficient of the fluid.

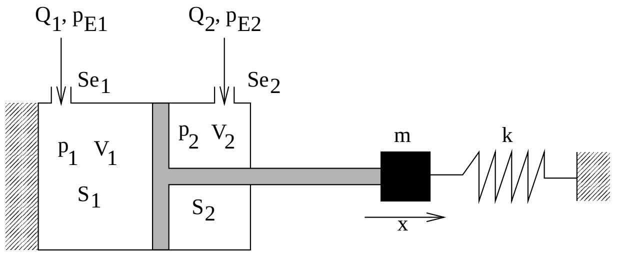

Fig. 3. Hydraulic system

If we consider that the position x=0 corresponds to the rest length of the spring, the dynamic equilibrium of mass m is given by

$$m\ddot{x} + kx - p_1 S_1 + p_2 S_2 = 0 \tag{25}$$

If V_{0i} is the volume of the chamber for the position x=0, the volume can be expressed in terms of x

$$V_1 = V_{01} + S_1 x \qquad V_2 = V_{02} - S_2 x \tag{26}$$

with S_1 and S_2 the cross sections of chambers 1 and 2.

The other equations of motion are obtained by matching the two expressions of the flow

$$\begin{aligned} S_1\dot{x} + \tfrac{V_{01}+S_1 x}{K}\dot{p}_1 \\ -C_d S_{e1}\sqrt{\tfrac{2(p_{E1}-p_1)}{\rho}} * \mathrm{sign}(\mathrm{p_{E1}} - \mathrm{p_1}) = 0 \end{aligned} \tag{27}$$

$$\begin{aligned} -S_2\dot{x} + \tfrac{V_{02}-S_2 x}{K}\dot{p}_2 \\ -C_d S_{e2}\sqrt{\tfrac{2(p_{E2}-p_2)}{\rho}} * \mathrm{sign}(\mathrm{p_{E2}} - \mathrm{p_2}) = 0 \end{aligned} \tag{28}$$

Although two equations are of first-order form, they can be naturally treated by specifying the 3 state variables as

$$q_1 = x \quad \dot{q}_2 = p_1 \quad \dot{q}_3 = p_2 \tag{29}$$

The system has been simulated with the data in table I with the following initial conditions

$$x_0 = \dot{x}_0 = 0 \qquad p_{1_0} = p_{2_0} = 10 \text{ bar}$$

for the pressure p_{E1} increasing linearly from 10 to 100 bar, in the interval [0:0.01] seconds. The initial accelerations ($\ddot{x}$, $\dot{p}_1$ and $\dot{p}_2$) are determined from the equations of motion.

Some results are presented in figures 4 to 6. The mass position (figure 4) is driven initially by the flow through the orifices and stops when the equilibrium is reached between the spring reaction and the force exerted by the jack. At the end of the simulation, the pressure in the chambers is the one of the corresponding circuit (figure 5). The stiff nature of the equations is illustrated on the evolution of the mass velocity (figure 6) where the oscillations due to the fluid compressibility can be clearly identified. Figure 7 shows how the integration procedure adapts the time step during the simulation. This evolution must be correlated with figure 6. In the beginning of the simulation, a small time step is necessary due to heavy oscillations of the system. Later, the behaviour is quasi-stationary and a larger time step is allowed. It can be seen that a maximum time step of 0.0005 seconds has been specified by the user.

TABLE I
Simulation data for the hydraulic jack

m=10 kg	$k = 10^5$ N/m	$K = 2 \cdot 10^8$ Pa
$\rho = 860$ kg/m^3	$p_{E2} = 10$ bar	S_1=0.001 m^2
S_2=0.0005 m^2	$V_{01} = 10^{-4}$ m^3	$V_{01} = 3 \cdot 10^{-4}$ m^3
$C_d = 0.611$	S_{e1}=10^{-5} m^2	S_{e2}=10^{-5} m^2

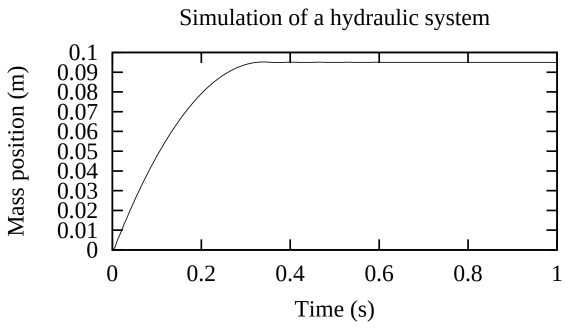

Fig. 4. Mass position

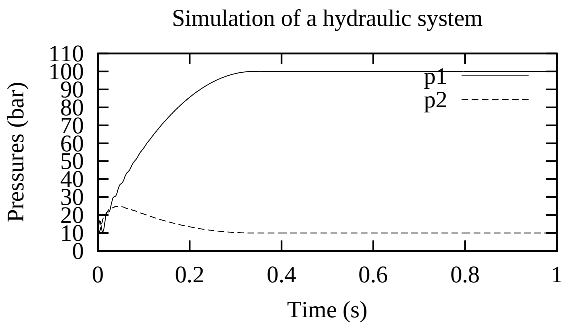

Fig. 5. Pressure in the chambers

C. The `mbs` *module*

The `mbs` module is a frontend to `sim` which automatically builds the residuals of the motion equations of a multibody system according to the formulation presented in section III. The user just has to provide two principal routines

• `ComputeMotion` which yields for each body the position matrix $\boldsymbol{T}_{0,i}$, the translation and rotation velocities $\boldsymbol{v}_i$ and $\boldsymbol{\omega}_i$ and the corresponding accelerations $\boldsymbol{a}_i$ and $\dot{\omega}_i$, expressed in terms of the chosen configuration parameters $\boldsymbol{q}$ and their first and time derivatives $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$

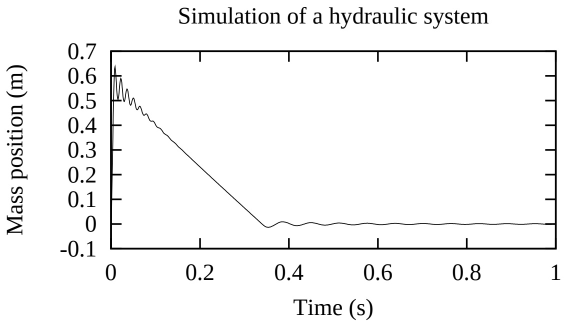

Fig. 6. Mass velocity

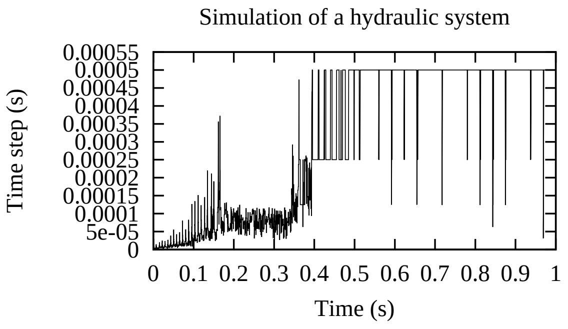

Fig. 7. Evolution of time step

• `AddAppliedEfforts` which builds the resultant force $\boldsymbol{R}_i$ and the resultant moment with respect to center of gravity $\boldsymbol{M}_{Gi}$, of the applied efforts exerted on each body.

Let us remark that the user doesn't have to provide the expression of the partial contributions. It is indeed clear, from the following relationship

$$\boldsymbol{v}_i = \sum_{j=1}^{n_{cp}} \boldsymbol{d}^{i,j} \cdot \dot{\boldsymbol{q}}_j \qquad \boldsymbol{\omega}_i = \sum_{j=1}^{n_{cp}} \boldsymbol{\theta}^{i,j} \cdot \dot{\boldsymbol{q}}_j \tag{30}$$

that the partial contribution $\boldsymbol{d}_{i,j}$ ($\boldsymbol{\theta}_{i,j}$) is equal to the translation or (rotation) velocity of body i when all generalized velocities are null but the jth one. The partial contributions will then be determined numerically from

$$\boldsymbol{d}^{i,j} = \boldsymbol{v}_i|_{\dot{q}_k=\delta_{jk}} \qquad \boldsymbol{\theta}^{i,j} = \boldsymbol{\omega}_i|_{\dot{q}_k=\delta_{jk}} \tag{31}$$

where δ_{ij} is the Kronecher index ($\delta_{ij} = 1$ if $i = j$, $\delta_{ij} = 0$ if $i \neq j$).

D. The `vec` *module*

Although the user's work is largely simplified, writing the kinematics remains difficult, especially for the accelerations. The main reason is that the laws of mechanics are compact when expressed in vector form, but become very complex once projected in a given coordinate system. Let's take the example of the double pendulum presented before. In vector form, the acceleration of the center of gravity of body 2 is written

$$\begin{aligned} \boldsymbol{a}_{G_2} &= \dot{\boldsymbol{\omega}}_1 \times \boldsymbol{OA} + \boldsymbol{\omega}_1 \times (\boldsymbol{\omega}_1 \times \boldsymbol{OA}) \\ &\quad + \dot{\boldsymbol{\omega}}_2 \times \boldsymbol{AG_2} + \boldsymbol{\omega}_2 \times (\boldsymbol{\omega}_2 \times \boldsymbol{AG_2}) \end{aligned} \tag{32}$$

with

$$\boldsymbol{\omega}_1 = \dot{q}_1 \vec{z}_0 \qquad \dot{\boldsymbol{\omega}}_1 = \ddot{q}_1 \vec{z}_0 \tag{33}$$

$$\boldsymbol{\omega}_2 = (\dot{q}_1 + \dot{q}_2)\vec{z}_0 \qquad \dot{\boldsymbol{\omega}}_2 = (\ddot{q}_1 + \ddot{q}_2)\vec{z}_0 \tag{34}$$

The expression, already sizeable, becomes much more complex when expressed in the axes of a coordinate sytem, as shown by the x component of the acceleration $\boldsymbol{a}_{G2}$, with respect to the global reference frame

$$\begin{aligned} \boldsymbol{a_{G_2}}_x &= \ddot{q}_1 \cos(q_1) - \dot{q}_1^2 \sin(q_1) + \ddot{q}_1 \cos(q_1+q_2) \\ &\quad + \ddot{q}_2 \cos(q_1+q_2) - \dot{q}_1^2 \sin(q_1+q_2) \\ &\quad - \dot{q}_2^2 \sin(q_1+q_2) - 2\,\dot{q}_1\,\dot{q}_1 \sin(q_1+q_2) \end{aligned} \tag{35}$$

It would become even more complex if the two axes of rotation were not parallel.

To solve the problem, the `vec` module defines classes related to the major components of vector algebra: vectors (type `vec`), rotation tensors (type `trot`), inertia tensors (type `tiner`) and homogeneous transformation matrices (type `mth`). The classical operators have been overloaded so that vector expressions can be written as in the C++ code. For example, the acceleration of body 2 can be programmed in the following way (note that all indices are shifted as they begin from 0 in C)

```
body[1].TOG=Trotz(q[0])*Tdisp(0,-l1,0)
          *Trotz(q[1])*Tdisp(0,-0.5*l2,0);
body[1].omega=body[0].omega+vcoord(0,0,qd[1]);
body[1].omegad=body[0].omegad+vcoord(0,0,qdd[1]);
vec AG2=body[1].TOG.R*vcoord(0,-0.5*l2,0);
body[1].vG=2*body[0].vG+(body[1].omega^AG2);
body[1].aG=2*body[0].aG+(body[1].omegad^AG2)
         +(body[1].omega^(body[1].omega^AG2));
```

This small piece of code illustrates some potentialities of the library

• the position matrix is built from the multiplication of predefined forms corresponding to rotations or displacements;
• the routine `vcoord` allows to build a vector from its 3 components;
• the homogeneous transformation matrix (`TOG`) comprises a rotation tensor (`R`) which can be used to build a vector expressed in the global frame from its local coordinates.

E. The `visu` *module*

The `visu` module allows to define a graphical scene composed of simple objects like boxes, frustums, lines, triangles, ... Each shape is attached to an homogeneous transformation matrix, generally the one giving the situation of a body, allowing to build successive configurations of the scene that can be saved to a file and animated by an independent viewer distributed with `EasyDyn`.

Figures 11 and 14 of next section are examples of scenes built with the help of the `visu` module.

VI. The help of symbolic tools

Even with the help of the vector classes, the kinematics remains problematic for an unexperienced user. It can be dramatically simplified if we figure out that all the kinematics can be derived from only the homogeneous transformation matrices.

The translation velocity $\boldsymbol{v}_i$ of body i can indeed be derived directly from the homogeneous transformation matrix

$$\{\boldsymbol{v}_i\}_0 = \frac{d}{dt}\{\boldsymbol{e}_i\}_0 = \sum_{j=1}^{n_{cp}} \frac{\partial\{\boldsymbol{e}_i\}_0}{\partial q_j} \cdot \dot{q}_j = \sum_{j=1}^{n_{cp}} \{\boldsymbol{d}_{i,j}\}_0 \cdot \dot{q}_j \quad (36)$$

In the same way the rotation vector is related to the time derivative of the rotation tensor by

$$\{\tilde{\boldsymbol{\omega}}_i\}_0 = \left(\begin{array}{ccc} 0 & -\omega_{z_i} & \omega_{y_i} \\ \omega_{z_i} & 0 & -\omega_{x_i} \\ -\omega_{y_i} & \omega_{x_i} & 0 \end{array}\right)_0 = \dot{\boldsymbol{R}}_{0,i} \cdot \boldsymbol{R}_{0,i}^T \quad (37)$$

One further derivation then naturally leads to the accelerations

$$\{\boldsymbol{a}_i\}_0 = \frac{d}{dt}\{\boldsymbol{v}_i\}_0 \qquad \{\dot{\boldsymbol{\omega}}_i\}_0 = \frac{d}{dt}\{\boldsymbol{\omega}_i\}_0 \quad (38)$$

A supplementary tool, called `CAGeM` (Computer Aided Generation of Motion) has been developed to help the users of `EasyDyn`. Practically, `CAGeM` is a MuPAD script which builds the core of a C++ application using `mbs` to simulate the behaviour of a multibody system. The `MuPAD` software is the result of several years of research in the university of Paderborn. Although it is also available for free on the net (http://www.mupad.de), it offers features comparable to its commercial counterparts like `Mathematica` or `MathCad`.

To use `CAGeM`, the user provides a MuPAD code with the following information

- the number of bodies and the number of configuration parameters;
- the inertia data of each body;
- the expression of the homogeneous transformation matrices of each body, expressed in terms of the chosen configuration parameters;
- the initial conditions;
- the gravity vector.

The script `CAGeM` uses the symbolic derivation features of `MuPAD` to build the expressions of velocities and accelerations from the position matrices.

The following code illustrates the user file related to the example of the double pendulum

```
titre:="Simulation of a double pendulum":
nbrdof:= 2:  // Number of degrees of freedom
nbrbody:= 2: // Number of bodies
// Gravity vector
gravity[1]:=0: gravity[2]:=-9.81: gravity[3]:=0:
// Some Constants
l0:=1.2: l1:=1.1:
// Inertia data
mass[0]:=1.1: mass[1]:=0.9:
Ixx[0]:=1: Ixx[1]:=1:
Iyy[0]:=1: Iyy[1]:=1:
Izz[0]:=l0^2/12*mass[0]:
Izz[1]:=l1^2/12*mass[1]:
// Position matrices
TOG[0] := Trotz(q[0]) * Tdisp(0,-l0/2,0):
TOG[1] := Trotz(q[0]) * Tdisp(0,-l0,0)
        * Trotz(q[1]) * Tdisp(0,-l1/2,0):
// Non null initial conditions
qi[1]:=1:
// Simulation conditions
TempsFinal:=5:
StepSave:=0.01:
StepMax:=0.005:
```

In this case, where the gravity is the only applied effort, no other task but compiling the resulting code is necessary. In some cases, the user will have to complete the procedure `AddAppliedEfforts`. Let us note that routines are provided for classical force elements like springs or dampers. Other routines for other types of elements can be added in the future. The ability to write the forces in vector form makes anyway the work much easier.

VII. Examples

A. Double pendulum

The double pendulum presented before has been simulated when subjected to gravity from the following initial conditions

$$q_1{=}0,\ \dot{q}_1{=}0,\ q_2{=}1 \text{ rad},\ \dot{q}_2{=}0$$

The generation of the C++ code from `MuPAD` took less than 1 second on a PC equipped with a processor running at 2GHZ. The evolution of the angles is illustrated in figure 8.

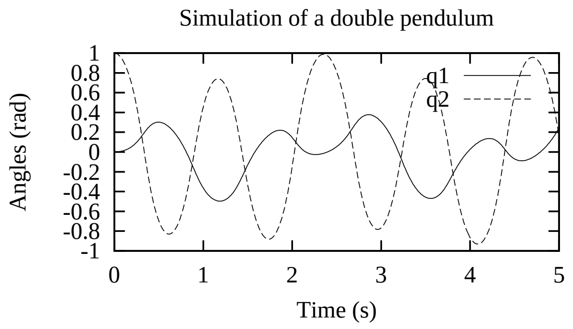

Fig. 8. Evolution of angles

B. Slider-crank mechanism

The fact of working with generalized coordinates does not prevent to simulate closed-loop mechanisms, as shown by the slider-crank mechanism illustrated in figure 9. The parameters α and x can indeed be expressed univoquely in terms of the angle q_0 by

$$\alpha = \arcsin\left(\frac{l_1 \sin(q_0)}{l_2}\right) \quad x = l_1 \cos(q_0) + l_2 cos(\alpha) \quad (39)$$

The relationships can be introduced symbolically in the `MuPAD` code describing the system as

```
l1 := 1: l2 := 2:
alpha := arcsin(l1*sin(q[0])/l2):
```

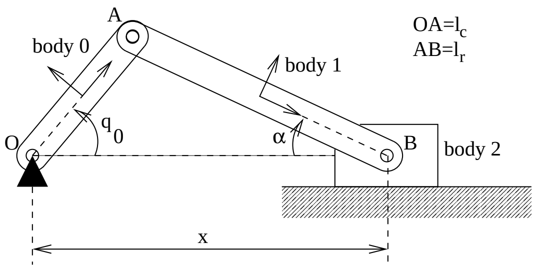

Fig. 9. Slider-crank mechanism

```
x := l1*cos(q[0]) + l2*cos(alpha):
TOG[0]:= Trotz(q[0]) * Tdisp(l1/2,0,0):
TOG[1] := Tdisp(x,0,0) * Trotz(-alpha)
        * Tdisp(-l2/2,0,0):
TOG[2] := Tdisp(x,0,0):
```

and the intermediary variables α and x will be automatically replaced by their expression and derived in a consistent way.

For more complex systems, the intermediary variables could eventually be determined in terms of the chosen configuration parameters from the symbolic or numeric resolution of constraint equations. This approach has been largely applied by Hiller et al. [3] on complex mechanical systems.

For this example, the generation of the C++ code from MuPAD takes approximately 3 seconds on a PC equipped with a 2GHz processor. The response of the system subjected to gravity has been simulated from initial conditions q_0=1 rad and $\dot{q}_0$=0, for the data detailed in table II. The evolution of the angle q_0, represented in figure 10, shows that the mechanism comes back to the same position after one oscillation, as there is no energy dissipation.

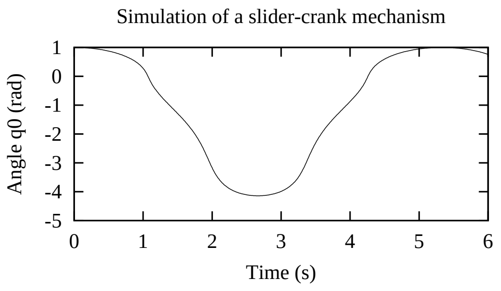

Fig. 10. Simulation of the slider-crank mechanism

TABLE II
CHARACTERISTICS OF THE SLIDER-CRANK MECHANISM

m_1=1 kg	I_{G1zz}=0.0833 kgm^2
m_2=2 kg	I_{G2zz}=0.6667 kgm^2
m_3=5 kg	I_{G3zz}=2 kgm^2
l_1=1 m	l_2=2 m

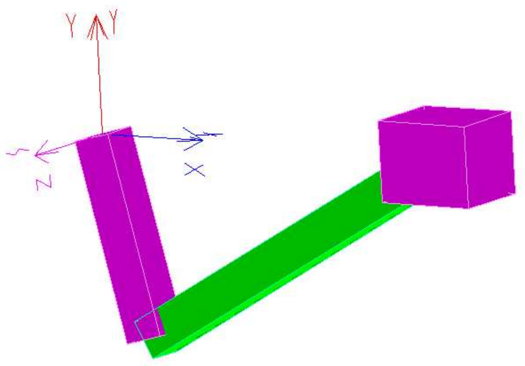

Fig. 11. Image of the slider-crank mechanism

C. Spatial robot

To show the generality of the library, the robot illustrated in figure 12 has been studied. The system is well documented as it was used as a benchmark for multibody systems softwares [2].

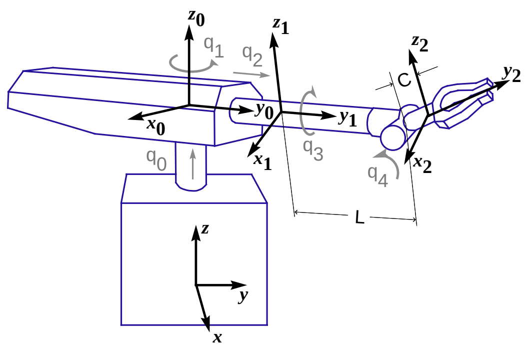

Fig. 12. Robot

The robot owns 5 degrees of freedom, depicted in figure 12 (C = 0.05 m and L = 0.5 m) and 3 bodies. With the help of the symbolic tool, the only information necessary to build the kinematics is the position matrix, which gives for the end arm

```
TOG[2] := Trotz(q[1]) * Tdisp(0,0,q[0])
        * Troty(q[3]) * Tdisp(0,q[2]+L,0)
        * Trotx(q[4]) * Tdisp(0,C,0):
```

TABLE III
INERTIA PARAMETERS OF THE ROBOT

	Body		
	0	1	2
mass (kg)	250	150	100
I_{xx} ($kg.m^2$)	(90)	13	4
I_{yy} ($kg.m^2$)	(10)	0.75	1
I_{zz} ($kg.m^2$)	90	13	4,3

The generation of the C++ program from MuPAD takes about 13 seconds on a PC equipped with a processor running at 2GHz.

The force law at each actuator (table IV) is defined in the benchmark for a point-to-point motion of the end point.

TABLE IV
Efforts developed by actuators

Time interval τ $[s]$	Efforts $[N]$ or $[N.m]$
$[0 : 0.5]$	$F0_Z = 6348$ $F1_Y = 36t + 986$ $T0_Z = 637t - 508$ $T1_Y = 0$ $T2_X = 63.5$
$[0.5 : 1.5]$	$F0_Z = 4905$ $F1_Y = -2$ $T0_Z = 148\ \exp(-5.5(t - 0,5)) - 8$ $T1_Y = 0$ $T2_X = 49,05$
$[1.5 : 2]$	$F0_Z = 3462$ $F1_Y = -1019$ $T0_Z = 240$ $T1_Y = 0$ $T2_X = 34,6$

The definition of these efforts in the C++ file is largely simplified by the elements of the `vec` library. Local axes of the frames can indeed be used directly to specify the direction of each effort. We get for example for the time interval [0:0.5]

```
if (t<0.5)
{
 body[0].R += 6348 * body[0].TOG.R.uz() ;
 body[0].MG += (673*t-508) * body[0].TOG.R.uz() ;
 body[1].R += (36*t+986) * body[1].TOG.R.uy() ;
 body[0].R -= (36*t+986) * body[1].TOG.R.uy() ;
 body[2].MG += 63.5 * body[1].TOG.R.ux() ;
 body[1].MG -= 63.5 * body[1].TOG.R.ux() ;
 }
```

Let us note that the user must pay attention to apply the action and the reaction to assure the coherence of the simulation.

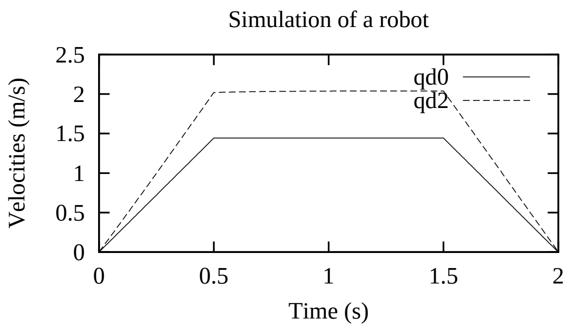

Fig. 13. Results of robot simulation

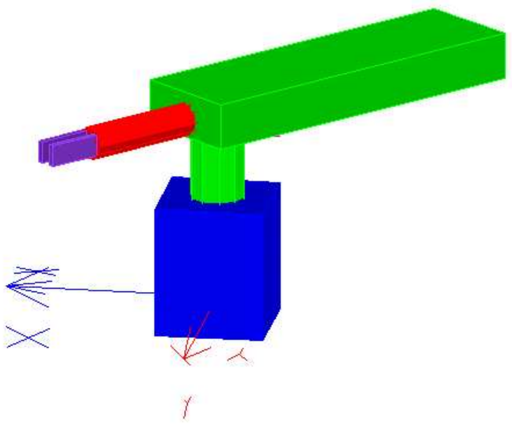

Fig. 14. Robot visualization

VIII. Conclusion

We have presented a framework, called **EasyDyn**, that can be used to simulate dynamics problems and in particular, multibody systems. The system is not described in a data file but from its kinematics and the expression of applied forces, programmed in a C++ application. Some utilities, like vector algebra classes and external symbolic tools, are provided to make the task of the user as easy as possible.

Although the framework has been tested on quite complex systems (3D manipulators, controlled systems, railway vehicles), it is not a replacement of available commercial simulation tools. In particular, the code was written for a maximum compactity, readability and scalability, often to the detriment of efficiency. It constitutes a flexible alternative for simple or moderately complex systems. Its main advantage is to be completely free and open source, allowing the user to tune the routines to its peculiarities, each one enlarging the application field. It is particularly well-suited for teaching and offers a foundation for collaborative or research work.

The future development will focus on the ability to build and integrate equations of motion comprising kinematic constraints.

References

[1] E.S. Raymond, *The Cathedral and the Bazaar*, O'Reilly and Associates, 2001

[2] W. Schiehlen, *Multibody Systems Handbook*, Springer-Verlag, 1991.

[3] M. Anantharaman and M. Hiller, *Numerical simulation of mechnanical systems using methods for differential-algebraic equations*, Int. J. Num. Meth Eng, Vol.21, pp 1531-1542, 1991

[4] K.E. Brenan, S.L. Campbell and L.R. Petzold, *Numerical Solution of Initial-Value Problems in Differentiel-Algebraic Equations*, North-Holland, 1989.

[5] O. Verlinden, P. Dehombreux, C. Conti and S. Boucher, *A New Formulation for the Direct Dynamic Simulation of Flexible Mechanisms Based on the Newton-Euler Inverse Method*, Int. J. Num. Meth Eng, Vol.37, pp 3363-3387, 1994

[6] C. Conti, P. Dehombreux, O. Verlinden and S. Datoussaid, *ACIDYM, a Modular Software for Computer-Aided Learning of Kinematic and Dynamic Analyis of Multibody Systems* in *Advanced Multibody System Dynamics: Simulation and Software Tools*, W. Schiehlen, Kluwer Academic Publishers, 1993