A Mixed symbolic-numeric software environmenrnt and its application to control systems engineering

François Delebecque, Ramine Nikoukhah

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.: (1) 39 63 55 11

Rapports de Recherche

1992

25ème anniversaire

N° 1695

Programme 5
Traitement du Signal,
Automatique et Productique

A MIXED SYMBOLIC-NUMERIC SOFTWARE ENVIRONMENT AND ITS APPLICATION TO CONTROL SYSTEMS ENGINEERING

François DELEBECQUE
Ramine NIKOUKHAH

Mai 1992

RR-1695

A mixed symbolic-numeric software environment and its application to control systems engineering

François Delebecque and Ramine Nikoukhah

INRIA, Rocquencourt BP 105, 78153 Le Chesnay Cedex, France

Abstract A high-level interactive symbolic-numeric software environment based on the symbolic computer algebra system Maple, its extension Macrofort and the high performance scientific software package Ψlab is presented. This general purpose software environment can be used both for solving specific problems and for developing specialized sofware packages. We shall illustrate this latter application by developing a simple sofware package for the simulation and control of multibody mechanical systems. As an example, simulation and control of a bicycle are presented.

Un environnement de programmation mixte, formel et numérique et son application à l'automatique

Résumé On présente un environnement de programmation mixte, formel et numérique, basé sur le système de calcul formel Maple, son extension Macrofort et le logiciel de CAO Ψlab. Cet environnement peut être utilisé aussi bien pour résoudre des problèmes spécifiques que pour le développement de logiciels spécialisés. Pour illustrer cette dernière application, on présente le développement d'un logiciel de simulation et de contrôle optimal de systèmes mécaniques multicorps, ainsi que son utilisation pour étudier le comportement d'un vélo.

1 INTRODUCTION

In most control system design problems some symbolic computations have to be carried out before the standard numerical techniques available on many CACSD packages such as Matlab, X-Math, Program CC or Basile can be used for the controller design. The application of symbolic computations in control vary widely: standard linearization of non linear models, sophisticated differential geometric manipulations such as feedback linearization, derivative array techniques for simulation and control of differential/algebraic systems [2], construction of the adjoint system for open-loop control of nonlinear systems, and many others. For this reason it is important for a control system designer to have access to symbolic computational facilities as well as numerical computational facilities. In particular, he should be able to pass on the results of his symbolic calculations to a CACSD package for further processing. The software environment presented here has been developed to provide such facilities. The basic philosophy in the development of this package has been to use existing programs with as little modifications as possible and developing powerful interfaces for data transfer. The advantage of this approach is that the user can fully exploit the capabilities of each program and customize the environment according to his specific needs.

The major components of this environment are Maple which has become a standard computer algebra system (see [3] for more information), Macrofort [4] a Fortran code generator in Maple and Ψlab a Basile based [5] interactive CACSD package (Macrofort and Ψlab are developed at INRIA). In this environment the user have access to the results of his Maple calculations in Ψlab. Macrofort and all the Fortran programs that it generates are completely transparent to the user. He sees only Maple with a few extra procedures for data transfer to Ψlab and Ψlab. So he is always in an interactive environment and he can fully take advantage of all the facilities available both in Maple and in Ψlab.

It would be rather tedious to present one-by-one the capabilities of this environment. That is why we have decided to highlight some of its most important functions by showing how a specialized package (in this case, for the simulation and control of multibody mechanical systems) can be developed in this environment and used (all the codes are given in the Appendix A).

In Section 2 we present briefly Macrofort and Ψlab (information about Maple is widely available). A software for modeling, simulation and open-loop control of constrained multibody mechanical systems is developed in Section 3. In Section 4 we use this software to study the dynamics of a bicycle. We conclude with a brief conclusion in Section 5.

2 BASIC SOFTWARE ELEMENTS

The Computer Algebra Maple has been designed to perform symbolic computations. Even though it is possible to perform numerical computations in Maple as well, the running time would be prohibitive due to internal representation of floating point numbers. This of course is common to all computer algebra systems for which symbolic computation is the main concern. Macrofort [4] is a package added to Maple which does complete Fortran 77 code generation. Macrofort uses the *fortran* procedure of Maple which translates Maple algebraic expressions into Fortran syntax. Macrofort is a collection of Maple procedures in the share library; it allows to generate complete Fortran subroutines (including labels, declarations, do loops, etc...) by transforming a list of instructions into a Fortran subroutine and of course each instruction may be the result of a sophisticated symbolic calculation performed by Maple. The generated code can be optimized using the optimize procedure in Maple (this is done by identifying common subexpressions).

Ψlab is an interactive CACSD package. In addition to standard matrix manipulation routines and graphics capabilities (2D, 3D and animation), Ψlab includes special functions for control, simulation and optimization of linear and non linear systems. Thanks to its open architecture, C and Fortran routines can be dynamically linked to Ψlab and in that case they become indistinguishable from standard Ψlab macros. This is a key feature used in our software environment. Other interesting features of Ψlab include: manipulation of polynomial and rational matrices, existence of lists in the basic data structure and recursive macro definition.

In our environment, the user need not worry at all about the generation of the Fortran code by Macrofort and its linking with Ψlab; all is done automatically by a Maple procedure called *maple2scilab*. This procedure takes a symbolic matrix (vector or scalar), generates the Fortran subroutine which has for inputs the variables of the matrix and for output the entries of the matrix, using Macrofort. It then "links" this Fortran subroutine to a Ψlab macro. As far as the Ψlab user is concerned, this macro is indistinguishable from any other Ψlab macro; it is as if the equations defining the matrix were written in Ψlab.

There is also a Maple procedure to convert Maple matrices into Ψlab language [6]. It is however clear that for complex problems there is an enormous advantage in terms of computation time in passing through Fortran (Ψlab is an interpreted language).

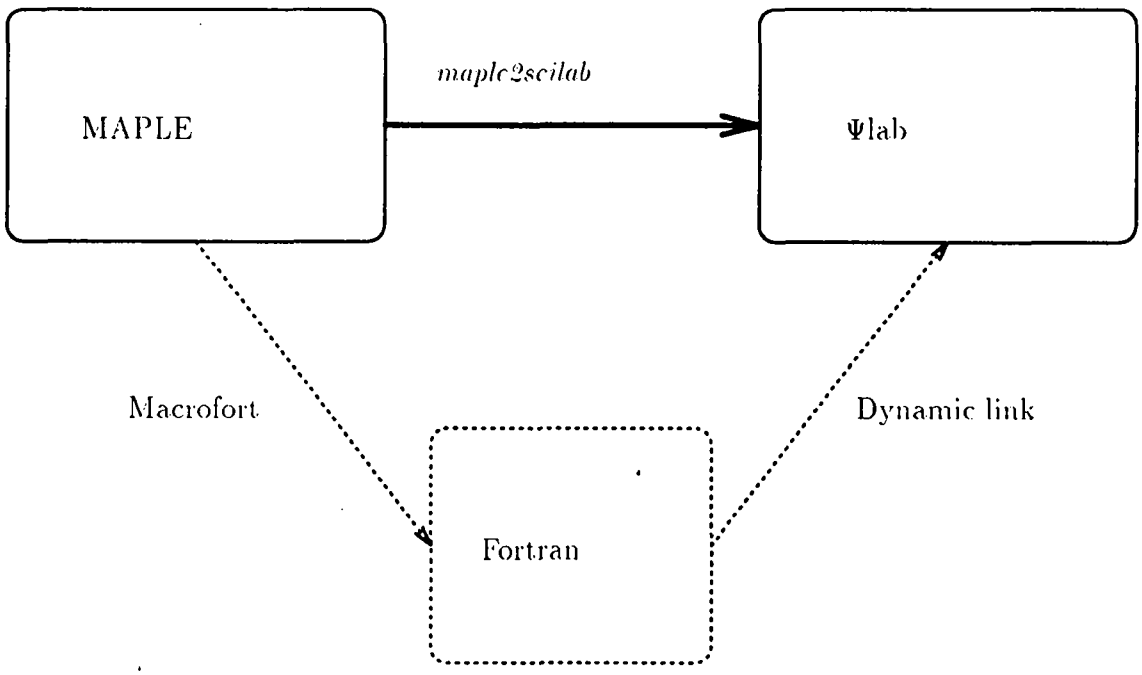

Figure 1. Software organization

The syntax of the *maple2scilab* procedure is the following:

maple2scilab(*macroname*, *maplematrix*, *arguments*)

maplematrix is the name of the Maple matrix having *arguments* as arguments (list of vectors or matrices) which is to be transferred. *macroname* is the name of the Ψlab macro which is to

be generated; it has for inputs numerical values of the arguments and for output the numerical value of the matrix. We have also developed a similar Maple procedure *sparse2scilab* for the transfer of sparse matrices (both Maple and Ψlab have facilities to handle sparse matrices) and a procedure which generates the Ψlab macro for calling the sparse linear system solver of Ψlab. This function is particularly useful for the application that we present next. *maple2scilab* and *sparse2scilab* generate each two files: *macroname.f* and *macroname.bas* which contain respectively the generated Fortran and Ψlab codes. Other facilities of this environment include automatic translation of Maple procedures into Ψlab macros and data transfer from Ψlab to Maple.

In the next section, we present a specialized sample software package developed in this environment to highlight some of its important features.

3 CONTROL OF MULTIBODY SYSTEMS

There exist a number of specialized software packages for modeling and simulation of multibody mechanical systems such as Adams, Sdfast, Gemmes, and many others. Our goal here is not to rewrite these large software packages but rather to show that in our software environment, with little effort, thanks to a compact code which extensively uses the matrix notations, we can emulate some of the features that exist in these specialized packages, needed for our application, and go further in the study of a particular problem which in this case consist of open-loop control. In particular, we consider the problem of modeling, simulation and control of multibody mechanical systems. We have not sought to have many of the facilities available on the specialized packages for modeling multibody systems; the inputs to our program are the system variables and parameters, the energy functions and the constraints (holonomic or non-holonomic). We use the variational approach for deriving the equations of motion as well as for obtaining the optimality conditions (equations for the adjoint system, etc...).

The outline of this section is as follows. In section 3.1 we start by reviewing quickly the variational approach for the construction of the equations of motion and its implementation in Maple. In section 3.2 we discuss the difficulties in simulating these equations which, almost always, are differential/algebraic. The integrator uses simulation functions of Ψlab. In Section 3.3, we study the open-loop control problem for these differential/algebraic systems which leads to solving a two-point boundary value system; a solution based on the gradient method is presented. The two-point boundary-value system is constructed by Maple and solved in Ψlab using its nonlinear optimization functions. A number of interesting problems related to the closed-loop (feedback) control of non-holonomic systems are presented in Section 3.4. In Section 3.5, the organization of the package and its use are discussed.

3.1 Modeling

Among various techniques for deriving the equations of motion, the Lagrangian method is perhaps, conceptually, the simplest approach. The idea is to minimize the integral action of the system. Specifically, we have to write the Lagrangian of the multibody system as if each body were free, and then minimize this Lagrangian under the constraints expressing how each body is connected to the others. Let q be the generalized coordinates of our system, $\dot{q}$ the generalized velocities, $\mathcal{L}(q,\dot{q})$ the Lagrangian (the kinetic energy minus the potential energy) and k the vector of external forces. Then, for a system with no constraints, the equations of motion are

obtained from the Euler-Lagrange's equation:

$$\frac{d}{dt}\frac{\partial}{\partial \dot{q}}\mathcal{L} - \frac{\partial}{\partial q}\mathcal{L} = k \tag{1}$$

which yields:

$$M(q)\ddot{q} = \phi(q, \dot{q}) + k \tag{2}$$

where M is called the generalized inertia matrix. When the system is subject to holonomic constraints, $f(q) = 0$, and/or non-holonomic constraints, $g(q)\dot{q} = 0$, the equations are modified as follows:

$$M(q)\ddot{q} = \phi(q, \dot{q}) + k - f_q^T(q)\lambda_1 - g^T(q)\lambda_2 \tag{3}$$

$$f(q) = 0 \tag{4}$$

$$g(q)\dot{q} = 0. \tag{5}$$

(3)-(5) is a differential/algebraic system in $q, \dot{q}, \lambda_1, \lambda_2$ where λ_1 and λ_2 are the Lagrange multipliers functions associated with the holonomic and non-holonomic constraints respectively; they can be interpreted as internal generalized forces which keep various components of the system together.

In the software, the construction of the generalized inertia matrix $M(q)$ and the function $\phi(q, \dot{q})$ from the Lagrangian $\mathcal{L}$ are done in "MAIN MAPLE PROGRAM FOR SIMULATION" using an auxiliary Maple procedure *euler_equations*, (see Appendix A.1). The external forces vector k can be obtained using the method of virtual work [9].

In the following section, we shall discuss specific problems that come up in the simulation of differential/algebraic system (3)-(5).

3.2 Simulation

The differential/algebraic system (3)-(5) is of index three [7]; this system cannot be numerically integrated properly. For the simulation, we need to reduce the index of the system by differentiating the constraints. In particular, we should differentiate holonomic constraints (4) twice and non-holonomic constraints (5) once. The result can be expressed as follows:

$$\begin{bmatrix} M & f_q^T & g^T \\ f_q & 0 & 0 \\ g & 0 & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} \phi + k \\ -\dot{f}_q \\ -\dot{g}\dot{q} \end{bmatrix}. \tag{6}$$

It can be shown that the matrix appearing on the left hand side of (6) is invertible provided the constraints are independent. In that case, system (6) can be integrated in time because we can compute explicitly $\ddot{q}$ as a function of q and $\dot{q}$ (and not λ_1 and λ_2). There are, however, two major difficulties: finding consistent initial conditions and stabilizing the integrator.

3.2.1 Initial conditions

To start the integration of second order systems of equations, we need to have initial values of the variables and their derivatives. In the case of differential/algebraic systems, however, these values are not independent. For system (6), consistent initial conditions, in addition to (4) and (5), must satisfy:

$$\dot{f} = f_q\dot{q} = 0. \tag{7}$$

Equations (4), (5) and (7) are nonlinear equations in q and $\dot{q}$. In order to find consistent initial conditions, we fix a number of the elements of q and $\dot{q}$ and we find the rest by minimizing

$$j = 1/2 \times \text{constr}^T \text{constr}, \tag{8}$$

where

$$\text{constr} = \begin{bmatrix} f \\ \dot{f} \\ g\dot{q} \end{bmatrix} \tag{9}$$

If the minimum value of zero (within machine precision) is found for j, we have a consistent initial condition. If not, the elements that we had fixed in advance are not consistent.

The minimization is carried out in Ψlab using the optimization primitive *optim*. A simplified syntax of this primitive for minimizing $j(x)$ is as follows:

$$[j^*, x^*] = \textbf{optim}(macroname, ['b', x_{inf}, x_{sup}], x_0, [algorithm])$$

where j^* is the minimum value of the function j to be minimized, achieved at x^*. *macroname* is the name of the macro which provides the value of the function j and its gradient j_x with respect to an input x (x may be a scalar, a vector or a matrix). $['b', x_{inf}, x_{sup}]$ indicates that x is bounded below and above by x_{inf} and x_{sup} respectively; this is an optional input. x_0 is the initial guess and *algorithm* specifies which algorithm is to be used (if no algorithm is specified *optim* will use a quasi-Newton algorithm.

In the developed software (see "MAIN SIMULATION SCILAB PROGRAM"), consistent initial conditions are found by *optim* with as first argument the macro *norm_ cstr*: this macro calculates j and its gradient j_x by calling the two macros *constr* and *gjx* which are automatically generated by Maple (see "MAIN SIMULATION MAPLE PROGRAM").

3.2.2 Integration stabilization by feedback

Theoretically, integrating (6) with consistent initial conditions should give the solution to the differential/algebraic system (3)-(5). But the integration will not work well because by differentiating the constraints, we have introduced marginally stable mode into the system. A simplistic explanation of this phenomenon is that every time that we differentiate a constraint we introduce a pole at zero.

In order to overcome this difficulty, we can use feedback to stabilize the integrator [8]. So that, instead of simply differentiating the holonomic constraint, $f = 0$, to obtain:

$$\ddot{f} = 0 \tag{10}$$

we use:

$$\ddot{f} + 2\alpha\dot{f} + \beta f = 0. \tag{11}$$

It is clear that if the initial condition is consistent, which means that f and $\dot{f}$ are zero, both equations (10) and (11) express the same thing: f must be zero everywhere. There is however one major difference. Assuming that α and β are chosen properly, if $f(q)$ or its derivative is not exactly zero, (11) will try to bring it back to zero during the integration. In contrast, the error will stay or even accumulate if (10) is used for the integration. Same thing holds for the derivative of the non-holonomic constraint (5) which can be stabilized as follows[1]:

$$g\ddot{q} + \dot{g}\dot{q} + \gamma g\dot{q} = 0. \tag{12}$$

[1]Stabilization of non-holonomic constraints is not crucial because, unlike in the case of holonomic constraints, there is no error accumulation.

Clearly α, β and γ must be chosen so that the resulting system is stable (i.e., zeros of $s^2 + \alpha s + \beta$ and $s + \gamma$ have negative real parts) and has good damping properties.

The complete system which is to be integrated can be expressed as follows:

$$\begin{bmatrix} M & f_q^T & g^T \\ f_q & 0 & 0 \\ g & 0 & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} \phi + k \\ -(\dot{f}_q + 2\alpha f_q)\dot{q} - \beta f \\ -(\dot{g} + \gamma g)\dot{q} \end{bmatrix}. \tag{13}$$

This equation is symbolically constructed by Maple in "MAIN MAPLE PROGRAM FOR SIMULATION" and transferred to Ψlab (see Appendix A.1) and numerically integrated in Ψlab. In each step of the integration, the linear system in $\ddot{q}, \lambda_1, \lambda_2$ is solved numerically by taking advantage of the possible sparsity of the matrix on the left hand side of (13).

The integration is performed using the integration primitive *ode* in Ψlab (see "MAIN SIMULATION SCILAB PROGRAM" in Appendix A.2). A simplified syntax of the *ode* primitive for solving the system of differential equations $\dot{x} = f(t, x)$, $x(0) = x_0$, where x is scalar, vector or matrix is the following:

$$x = \textbf{ode}([algorithm], x_0, t_0, t, macroname)$$

where *algorithm* specifies the choice of the integration routine, x_0, the initial condition at t_0, t, the time vector where x is to be evaluated, and *macroname*, the name of the Ψlab macro which evaluates f.

In Appendix A.2, (13) is numerically integrated using the *ode* primitive with a non stiff integrator (*algorithm* = *'adams'*). The macro passed to *ode* is *simul*, which computes $\text{xdot}^T = [\dot{q}^T\ ,\ \ddot{q}^T]$ as a function of $x^T = [q^T\ ,\ \dot{q}^T]$. *simul* calls ths *iihs* macro which solves the sparse linear system (13). Matrix on the left hand side of (13) is denoted by H and vector on the right hand side of (13) is denoted by h. The *iihs* macro and its associated Fortran code are generated automatically by our environment (in particular by the *sp_ scilab* procedure).

3.3 Open-loop control

In [10] a solution to the optimal control problem for constrained multibody systems has been presented. The first step is to express all the constraints (holonomic or not) as:

$$A(q)\dot{q} = b. \tag{14}$$

Of course non-holonomic constraints are already in this form, and holonomic constraints can be put into this form by differentiation:

$$A = \begin{bmatrix} f_q \\ g \end{bmatrix}. \tag{15}$$

With this notation, and by assuming that k is a function of q and a control vector u (and possibly $\dot{q}$), (3) can be expressed as follows:

$$M(q)\ddot{q} = F(q, \dot{q}, u) + A^T(q)\lambda \tag{16}$$

where $F = \phi(q, \dot{q}) + k(q, u)$ is called the generalized force vector and $\lambda^T = (\lambda_1^T, \lambda_2^T)$.

The control problem considered in [10] consists in minimizing

$$J = \int_0^T c(q, \dot{q}, u)dt + \Psi(q(T), \dot{q}(T)) \tag{17}$$

over the control trajectories $u(.)$ and possibly the time T. c and Ψ are respectively the integral and final cost functions.

Using a variational approach, the necessary conditions of optimality are obtained (for simplicity we assume that T is constant): optimal open-loop control function $u(.)$ satisfies:

$$\max_{u \in U}(c + p_2^T F) \tag{18}$$

where U denotes the set of admissible controls and p_2 satisfies the adjoint system:

$$\begin{bmatrix} M & 0 & A^T \\ (M\dot{q})_q^T & I & (A\dot{q})_q^T \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{p}_2 \\ \dot{p}_1 \\ \dot{\mu} \end{bmatrix} = \begin{bmatrix} -[F_{\dot{q}} + (\dot{M}\dot{q})_{\dot{q}}]^T p_2 - p_1 - c_{\dot{q}} \\ -[F_q + (\dot{M}\dot{q})_q - (A^T\lambda)_q]^T p_2 - c_q \\ -\dot{A}p_2 \end{bmatrix}, \tag{19}$$

$$\begin{bmatrix} M & 0 & A^T \\ (M\dot{q})_q^T & I & (A\dot{q})_q^T \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} p_2(T) \\ p_1(T) \\ \mu(T) \end{bmatrix} = \begin{bmatrix} \Psi_{\dot{q}}^T \\ \Psi_q^T \\ 0 \end{bmatrix}. \tag{20}$$

Here μ is the Lagrange multiplier associated with the constraint (14). The solution to the optimal control problem can be obtained by solving the two point boundary value system (14), (16), (18), and (19). Even though it is possible to use direct methods such as shooting to solve this two point boundary value system, here we shall use an optimization approach which allows us to take advantage of the sophisticated optimization primitives of Ψlab. In particular, we consider u to be piecewise constant; we compute the gradient of J with respect to u_i, constant value of u between t_i and t_{i+1}, and pass it on to the optimization primitive.

The gradient of J with respect to u is computed as follows: first integrate the primal system using u (and of course using the feedback stabilization technique introduced in the previous section); then integrate backwards the adjoint system and simultaneously compute

$$\mathcal{I} = \int (c_u + p_2^T F_u) dt; \tag{21}$$

finally the gradient of J with respect to u_i is given by:

$$J_{u_i} = \mathcal{I}(t_{i+1}) - \mathcal{I}(t_i) \tag{22}$$

Matrix appearing on the left hand side of (19) is represented by the variable *Emat* in the software environment. Vectors appearing on the right hand side of (19) and (20) are represented, respectively, by Fvec and vecfin. Finally, $c_u + p_2^T F_u$ is denoted hamu. These variables are constructed symbolically in "MAIN MAPLE PROGRAM FOR CONTROL" (Appendix A.1) and all the numerical integrations are performed in "MAIN CONTROL SCILAB PROGRAM" (Appendix A.2). Ψlab takes advantage of the sparsity of *Emat* to reduce the computation time.

3.4 Linear analysis

In most control problems, it is necessary to construct the linearized model of the nonlinear system. In the case that we are treating here (implicit systems), linearization is slightly more complicated; it is however straightforward to verify that linearization of (13) yields:

$$\begin{bmatrix} \dot{\delta q} \\ \dot{\delta \dot{q}} \end{bmatrix} = A \begin{bmatrix} \delta q \\ \delta \dot{q} \end{bmatrix} + B\delta u \tag{23}$$

where

$$A = \begin{bmatrix} 0 & I \\ X.(h_q - J_q) & X.h_{\dot{q}} \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0 \\ X.h_u \end{bmatrix} \tag{24}$$

with

$$X = \begin{bmatrix} I & 0 \end{bmatrix} \Pi^{-1} \quad \text{and} \quad J = \Pi \begin{bmatrix} \ddot{q} \\ \lambda_1 \\ \lambda_2 \end{bmatrix} \tag{25}$$

where Π is the matrix appearing on the left hand side of (13).

The necessary symbolic computations (matrix J_q, jacobian of J with respect to q and vectors $h_q, h_{\dot{q}}, h_u$, respectively, the gradients of h with respect to $q, \dot{q}$ and u) are performed by "MAPLE PROGRAM FOR LINEARIZATION" and the A and B matrices are numerically evaluated at a given $[q^T\ ;\ \dot{q}^T]$ by the Ψlab macro *linear* (see Appendix A.2).

Having access to A and B matrices in Ψlab is extremely useful because they can be used to design linear controllers, using the control toolbox. Linear control of implicit systems is an interesting area of research. For example, by linearization, we note that non holonomic constraints imply uncontrollables poles at zero. This means that standard time invariant linear controllers cannot be used to stabilize such systems.

3.5 Software organization

The simulation and control of multibody systems program presented here is composed of three main Maple programs: *simul.maple* which generates the equations of motion, *control.maple* which generates the equations of the adjoint system for optimal open loop control problem and *linear.maple* which generates the linearized model of the system. These Maple programs have their counterpart Ψlab macros: *simul.bas* for simulation, *control.bas* for trajectory optimization and *linear.bas* for the construction of the linear model (all of the programs are given in Appendix A).

To use this software the user must provide two Maple files: *data.maple*, containing the Lagrangian and the constraints as a function of the generalized coordinates, velocities and system parameters and *datopt.maple* which should define cost functions and their parameters. The user can of course uses all the functions available in Maple for constructing the Lagrangian and the constraints. The user must also provide two Ψlab files: *data.bas* providing numerical values for the system parameters and the initial condition and *datopt.bas* providing numerical values for the optimization parameters. Again, the user can use all Ψlab facilities. Finally, he should provide a Ψlab macro called *control* for the simulation providing the input forces and torques to the system: these inputs may be a function of the coordinates and velocities of the system as well as time, and thus it allows simulation of mechanical systems under feedback control. Of course, the Ψlab macro *control* can be written directly in Ψlab or can be written in Maple and transferred. In applications such as feedback linearization, *control* may be very complex and is in general obtained via complicated symbolic manipulations and thus it makes sense in that case to use Maple to construct *control*.

Once Maple programs executed, user can start working in Ψlab. All the Fortran and the associated Ψlab macros are generated automatically; as far as the user is concerned the whole transfer is transparent. He can execute the provided Ψlab macros to simulate and find the optimal control or use the macros generated by Maple in his own programs. He can thus take advantage of all the tools available in Ψlab to analyze his mechanical system or its linearized version.

In order to complete this package, we need a preprocessor for deriving the Lagrangian and the constraints from geometrical descriptions. Such user friendly preprocessors exist on most specialized programs.

4 EXAMPLE: RIDERLESS BICYCLE

In this section we illustrate the use of the software by an example. The system considered is a riderless bicycle shown in the following figure:

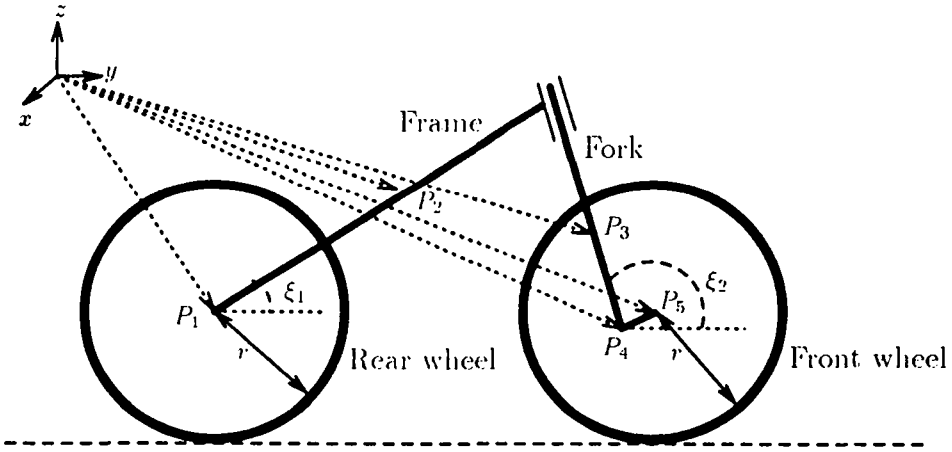

Figure 2. Bicycle model

Even though the geometries of the frame and the fork that we consider are not those of a real bicycle, they provide a good approximation.

Two controls are considered: a torque on the handlebars (steering) and a torque on the rear wheel (pedaling). These choices have been made because it is rather straightforward to construct a riderless bicycle with simple actuators positioned between the frame and the fork and between the frame and the rear wheel. It would have been also interesting to consider another input corresponding to a rider leaning to the left or to the right. However, emulating this input mechanically on a riderless bicycle is complicated and anyway not necessary for controlling the bicycle.

To make a bicycle turn at a low speed a rider usually leans. However, at higher speeds and particularly on a motorcycle the rider uses more and more countersteering, i.e., turning the handlebars in the opposite direction of the desired turn. Countersteering is of course well known to people who race motorcycles and can be used at low speed as well as at high speed to manœuver the bicycle. We shall see that the optimization algorithm that we use is capable of finding the countersteering strategy. This result has also been obtained in [10] for a model of unicycle.

Our simplified model of the bicycle is made of four parts: the rear wheel, the frame, the fork and the front wheel. The choice of the generalized coordinates set that has been made contains twenty three variables and twenty parameters (see Appendix B for their definitions). The model includes sixteen holonomic constraints specifying the geometry of the bicycle and four non-holonomic constraints specifying that the two wheels of the bicycle roll on the ground

without slipping.

The following figure shows the one second simulation result for a bicycle starting from the right up position with a speed of eight meters per second which after 0.1 seconds is subjected to a torque pushing its handlebars to the left for a period of 0.2 seconds.

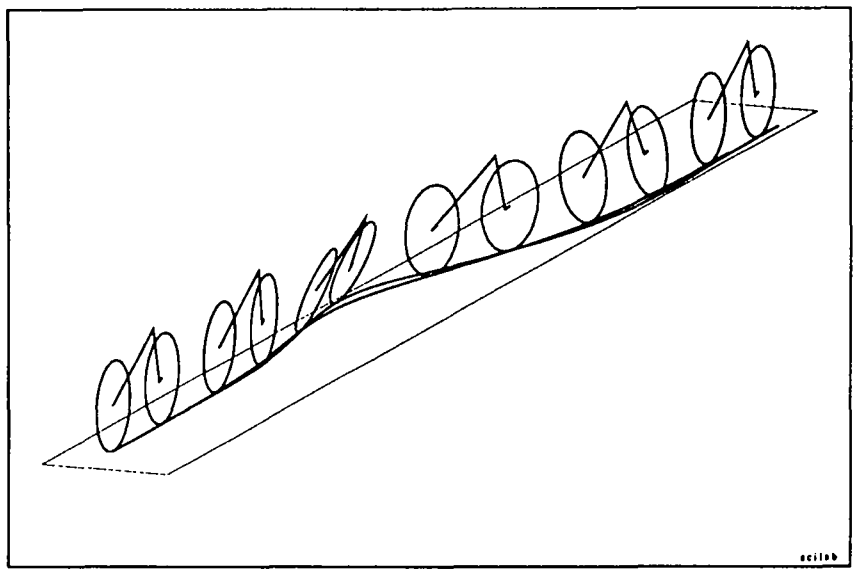

Figure 3. Bicycle simulation.

Since the speed of the bicycle is above a certain threshold, the bicycle gets back to the right up position and follows a linear trajectory. At lower speeds the bicycle would have either entered a zig-zag trajectory, a circular path or fallen down. These various behaviors of the bicycle are related to its stability properties as a function of its speed. This can be studied by examining the eigenvalues of the A matrix associated with the linearized model.

As an example of a trajectory optimization problem, we have considered the problem of making the bicycle turn by $\pi/4$. in the shortest possible time, making sure that at the end of the turn, it is in the right up position and would continue a linear trajectory in the absence of external forces. The two controls were the steering torque and the pedaling torque (both controls were supposed to be bounded in magnitude). The constraint on the turn angle has been modeled as a final cost; no integral cost (except time) has been considered.

The result obtained by our program is illustrated in the following figure:

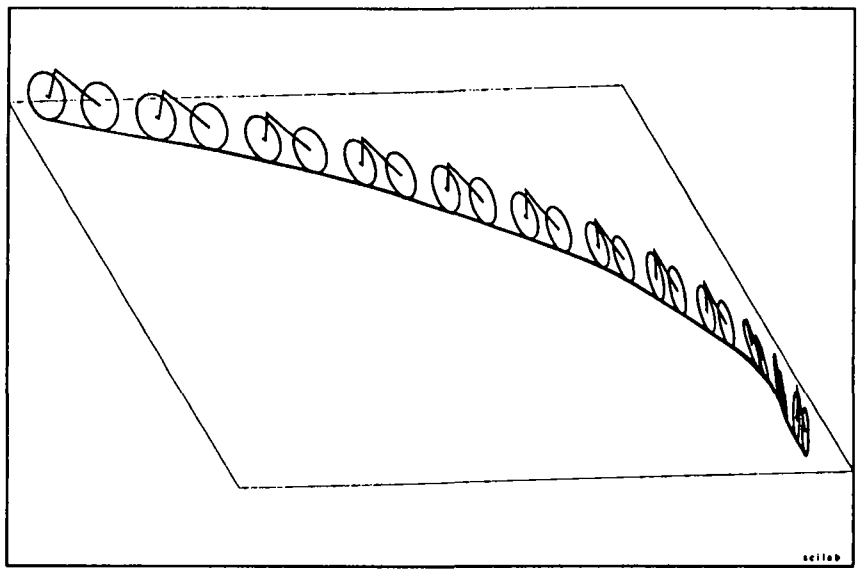

Figure 4. Bicycle control.

The optimal controls, as expected, are bang-bang: for the steering torque, it consists of first pushing the handlebars in the opposite direction of the turn (to lean the bicycle) and near the end of the turn, pushing the handlebars in the direction of the turn to bring back the bicycle to the right up position, as for the pedaling torque, the problem being a minimum time problem, the optimal control consists of pedaling with maximum torque (as fast as possible).

5 CONCLUSION

Because of numerous applications of symbolic computations in control system design, we have developed a software environment allowing the designer to transform efficiently the results of his Maple calculations into Ψlab macros. Thanks to its open architecture, this environment is particularly suitable for the development of specialized software packages, an example of which has been presented.

We consider going further in the integration of symbolic and numeric packages by developing a unique shell in which commands in each system can be invoked.

APPENDICES

A Codes

This appendix presents all the Maple and Ψlab programs in our specialized package for simulation and control for multibody mechanical systems.

A.1 Maple programs

```
#              MAIN MAPLE PROGRAM FOR SIMULATION
#                    (File simul.maple)
#
# User input file data.maple
read('data.maple'):
# The file data.maple should provide:
#     -n: dimension of vector q (generalized coordinates)
#     -nf: number of holonomic constraints
#     -ng: number of non-holonomic constraints
#     -ncontr: number of controls (size of u)
#     -L: Lagrangian (function of q,qd,u,param)
#     -f: vector of holonomic constraints (function of q,param)
#     -gqd: vector of non-holonomic constraints (function of q,qd,param)
#           (linear in qd)
#     -k: control vector (function of q,u,param)
#
# Preliminary calculations
# ------------------------
# Internal variables x=[q;qd]:
x:=convert(stack(convert(q,matrix),convert(qd,matrix)),vector):
qdd:=array(1..n):

# Defining Jacobians of constraints and their time derivatives:
fq:=jacobian(f,q):
g:=jacobian(gqd,qd):
fqd:=map(time_diff,fq,n):
gd:=map(time_diff,g,n):
A:=stack(fq,g):

# Defining the vector constr = [f;dot(f);gqd]
# and gjx = its jacobian w.r.t  x = [q,qd] (functions of q,qd,param)
constr:=map(simplify,vstack(f,evalm(fq &* qd),gqd)):
gjx:=map(simplify,jacobian(constr,x)):

#    Derivation of equations of motion
#    ---------------------------------
#  M qdd = phi + k - fq.lambda1 -g.lambda2
```

```
V:=euler_equations(L,n):
#V is the Euler operator (n vector function of q,qd,qdd)

M:=jacobian(V,qdd):
phi:=map(simplify,evalm((M &* qdd) - V)):

#       [M  fq' g']
#  II = [fq  0   0]      (n+nf+ng)x(n+nf+ng) matrix (Pi matrix).
#       [g   0   0]
#
col1:=stack(M,A):
col2:=stack(transpose(A),matrix(nf+ng,nf+ng,0)):
II:=map(simplify,concat(col1,col2)):

#       [              phi + k              ]
#   h = [- ( dot(fq)+2 alfa fq ) qd - beta f ]    (n+nf+ng) vector
#       [    - ( dot(g) + gama g ) qd        ]
#
h:=vstack(                    evalm( phi + k )  ,
          evalm( -1 * ( (fqd + 2*alfa * fq) &* qd )  - (beta * f)  ),
                 evalm( -1 * (gd + gama * g) &* qd  )      ):
h:=map(simplify,h):

# Transforming the Maple matrices into scilab macros via Fortran:
#---------------------------------------------------------------
# constr and gjx ----> scilab macros 'constr' and 'gjx'

maple2scilab('constr',constr,[q,qd,param]):
maple2scilab('gjx',gjx,[q,qd,param]):

# II sparse matrix and h vector ----> 'ii' and 'h' scilab macros

sparse2scilab('ii',II,[q,qd,u,param,alfa,beta,gama]):
maple2scilab('h',h,[q,qd,u,param,alfa,beta,gama]):

#    Making iihs = scilab macro solving II . x = h

sp_scilab(iihs,'II','h',[q,qd,u,param,alfa,beta,gama],rowdim(II)):
```

Auxiliary procedures

```
euler_equations:=proc(L,n)
# Calculation of Euler's equation given the Lagragian L:
# L(q[1],q[2],..,q[n],qd[1],qd[2],...qd[n]) of the state variables q
```

```
# and their time derivative qd
# this function returns a vector V[1],...,V[n], with
#          d  del (L)      del (L)
#  V[k] =  -- ----     -   ----
#          dt del q[k]     del qd[k]
# V[k] is an expression V(q[1],...,q[n],qd[1],...,qd[n],
#                         qdd[1],...,qdd[n])
# with qdd[k] 2nd time derivative of q[k]
#
local k:
v:=array(1..n):
for k to n do
          v[k]:=time_diff( diff(L,qd[k]),n)-diff(L,q[k] ):
          od:
op(v):
end:

time_diff:=proc(phi,n)
# Given an expression
#            phi(q[1],q[2],..,q[n],qd[1],qd[2],...qd[n])
# this procedure computes time derivative of phi when the arguments
# q[1],..,q[n],qd[1],..,qd[n] are time dependent functions and each
# function qd[k](t) is interpreted as the time derivative of q[k](t).
# The output is a function of
#             (q[1],...,q[n],qd[1],...,qd[n],qdd[1],...,qdd[n])
# where qdd is the time derivative of qd.
local phi_copy,k:
phi_copy:=phi:
for k to n do
    phi_copy:=subs(q[k]=q[k](t),phi_copy):
    phi_copy:=subs(qd[k]=qd[k](t),phi_copy):
          od:
diff_phi:=diff(phi_copy,t):
for k to n do
    diff_phi:=subs(diff(q[k](t),t)=qd[k],diff_phi):
    diff_phi:=subs(diff(qd[k](t),t)=qdd[k],diff_phi):
    diff_phi:=subs(q[k](t)=q[k],diff_phi):
    diff_phi:=subs(qd[k](t)=qd[k],diff_phi):
          od:
end:

vstack:=proc(v1,v2,v3)
# Returns the vector
#            [v1]
#            [v2]
#            [v3]
#
convert(stack(convert(v1,matrix),
```

```
              convert(v2,matrix),
              convert(v3,matrix)),vector):
end:

#              MAIN MAPLE PROGRAM FOR CONTROL
#
# Initialization: this program requires the results of 'simul.maple'

# User input file cost.dat
read('cost.dat'):
# The file cost.dat should provide:
#            -psi: final cost  (function of q,qd,paramopt)
#            -c: integral cost (function of q,qd,u,paramopt)
#            -mparo: size of paramopt
#
# Transferring c and psi to scilab
maple2scilab('c',c,[q,qd,u,paramopt]):
maple2scilab('psi',psi,[q,qd,paramopt]):
#
# Preliminary calculations
# ------------------------
# Internal variables
lambda:=array(1..nf+ng):
p1:=array(1..n):
p2:=array(1..n):
Idn:=array(identity,1..n,1..n):
w4:=evalm(map(time_diff,M,n) &* qd):

# Emat matrix
col1:=stack(             M,
            transpose(jacobian(evalm(M &* qd),q)),
                         A      ):
col2:=stack(        matrix(n,n,0),
            array(identity,1..n,1..n),
                    matrix(nf+ng,n,0)   ):
col3:=stack(       -transpose(A),
            -transpose(jacobian(evalm(A &* qd),q)) ,
                   matrix(nf+ng,nf+ng,0)    ):

Emat:=map(simplify,concat(col1,col2,col3));

# RHS vector
F:=evalm( phi+k ):
w1:=evalm(-1 * transpose( jacobian(F,qd)  + jacobian(w4,qd) ) &* p2
          - p1 - grad(c,qd)):
w2:=evalm(-1 * transpose(  jacobian(F,q) + jacobian(w4,q)
                         - jacobian(transpose(A) &* lambda,q) ) &* p2
          - grad(c,q)  ):
```

```
w3:=evalm(-1 * map(time_diff,A,n) &* p2):

Fvec:=vstack(w1,w2,w3):
Fvec:=map(simplify,Fvec):
#
vecfin:=vstack(grad(psi,qd),grad(psi,q),vector(nf+ng,0)):
vecfin:=map(simplify,vecfin):

# Gradient of cost w.r.t u
hamu:=evalm(  transpose( convert(grad(c,u),matrix) )
            + transpose( convert(p2,matrix) ) &* jacobian(F,u)  ):

hamu:=map(simplify,transpose(hamu)):
#
# Transfering Maple matrices into scilab macros via Fortran

maple2scilab('hamu',hamu,[q,qd,u,lambda,p1,p2,param,paramopt]):
sparse2scilab('emat',Emat,[q,qd,u,lambda,p1,p2,param,paramopt]):
maple2scilab('fvec',Fvec,[q,qd,u,lambda,p1,p2,param,paramopt]):
maple2scilab('vecfin',vecfin,[q,qd,u,lambda,p1,p2,param,paramopt]):

# Making scilab macro for solving the sparse system Emat x = rhs

sp_scilab(efs,'emat','rhs',[q,qd,u,lambda,p1,p2,param,paramopt],
          rowdim(Emat)):


#                    MAPLE PROGRAM FOR LINEARIZATION
#This program requires the matrix II and the vector h calculated
#by the simulation program simul.maple.
#
lambda:=array(1..nf+ng):
all:=convert(stack(convert(qdd,matrix),convert(lambda,matrix)),vector):

jq:=map(simplify,jacobian( evalm(II &* all) ,q )):
hq:=map(simplify,jacobian(h,q)):
hqd:=map(simplify,jacobian(h,qd)):
hu:=map(simplify,jacobian(h,u)):

maple2scilab('jq',jq,[q,qd,qdd,lambda,param]):
maple2scilab('hq',hq,[q,qd,u,param,alfa,beta,gama]):
maple2scilab('hqd',hqd,[q,qd,u,param,alfa,beta,gama]):
maple2scilab('hu',hu,[q,qd,u,param,alfa,beta,gama]):
```

A.2 Ψlab programs

```
-----------------MAIN  SIMULATION SCILAB PROGRAM--------------------
```

```
//[x]=simulation(tmin,tmax,nn)
//Simulation macro: the output x contains nn discretized positions and
//velocities from tmin to tmax.

exec('data.bas');
// data.bas should provide the param vector and initialize:
// -n,nf,ng
// -q0 and qd0: guessed initial position and velocity
// -flags: binary vector indicating which components
//        of q0 and qd0 are fixed.
// scilab macro ''control'' should be provided as well evaluating  u
// as a function of time, position and velocity; u = control(t,[q;qd])

// Internal variables
deltat=(tmax-tmin)/nn;
instants=tmin:deltat:tmax;
indq=1:n;indqd=(n+1):2*n; //(indexes of q and qd in x).

//    Finding consistent initial conditions

[jstar,x_init]=optim(norm_cstrs,[q0;qd0]);
if jstar>1.d-5 then write('inconsistent initial conditions');return;
end

//    Simulation
x=ode('adams',x_init,tmin,instants,simul);

//end

 -------------------------AUXILIARY MACROS--------------------------
//[j,j_x]=norm_cstrs(x)
// Computes j: the 2-norm of the inconsistency of initial condition
// and j_x: its projected gradient in the direction indicated by flags
q=x(indq);qd=x(indqd);
ctr=constr(q,qd,param);
j_x=ctr'*gjx(q,qd,param)*diag(flags);
j=0.5*ctr'*ctr;
//end

//[xdot]=simul(t,x)
// Returns xdot = [qd;qdd] given t and x=[q;qd]
q=x(indq);qd=x(indqd);
u=control(t,x);
// Solving the sparse linear system II qdd_lam = h, using scilab macros
// II, h and iihs automatically generated by Maple.
qdd_lam=iihs(II,h,q,qd,u,param,alfa,beta,gamma);
qdd=qdd_lam(1:n);
xdot=[qd;qdd];
```

```
//end
 ------------------------------------------------------------------


 -----------------MAIN  CONTROL SCILAB PROGRAM--------------------
exec('data.bas');

exec('datopt.bas');
// datopt.bas should provide:
// paramopt,
// boundsmin and boundsmax: bounds on  control
// u0: initial guess for control (size ncontr by nn-1)

// Internal variables
indq=1:n;indqd=(n+1):2*n;   //(Indexes of q and qd in x)
xdeltat=(tmax-tmin)/(nnx-1);
deltat=(tmax-tmin)/(nn-1);
instants=tmin:deltat:tmax;
xinstants=tmin:xdeltat:tmax;

// Finding consistent initial conditions
[jstar,x_init]=optim(norm_cstrs,[q0;qd0]);
if jstar > 1.d-5 then
write('inconsistent initial conditions');return;
end

// Finding open loop optimal control
x_initc=[x_init;0];
umin=diag(boundsmin)*ones(ncontr,nn-1);
umax=diag(boundsmax)*ones(ncontr,nn-1);
[costar,uu]=optim(cost,'b',umin,umax,u0);
//end

 --------------------------AUXILIARY MACROS--------------------------
//[totc,totc_u]=cost(uu)
// Computes the total cost totc and its gradient totc_u
// associated with uu.
// Forward integration : x=[q;qd;integral(c)] evaluated at xinstants
x=ode('adams',x_initc,tmin,xinstants,simulfor);
// Computation of the cost: final cost psi + integrated cost intc
intc=x(2*n+1,nnx);qfinal=x(indq,nnx);qdfinal=x(indqd,nnx);
totc=psi(qfinal,qdfinal)+intc;
// Initialization of the backward integration
//        xxf=[p2final;p1final;mufinal;0]
xxf=fcond(qfinal,qdfinal);
// Backward integration : xx=[p2;p1;mu;integral(hamu)]
xx=ode('adams',xxf,tmax,tmax:-deltat:tmin,backsimul);
// Computation of the gradient
ih_u=xx(2*n+nf+ng+1:2*n+nf+ng+ncontr,nn:-1:1);
```

```
totc_u=ih_u(:,2:nn)-ih_u(:,1:nn-1);
//end

//[xdot]=simulfor(t,x)
// Returns xdot = [qd,qdd,c] given t and x=[q,qd]
q=x(indq);qd=x(indqd);
k=ent(t/deltat)+1;u=uu(:,k);
qdd_lam=iihs(II,h,q,qd,u,param,alfa,beta,gamma);
qdd=qdd_lam(1:n);
xdot=[qd;qdd;c(q,qd,u)];
//end

//[xxdot]=backsimul(t,xx)
// Returns [p2dot,p1dot,mudot,hamu]
p2=xx(1:n);p1=xx(n+1:2*n);
k=ent(t/deltat)+1;u=uu(:,k);
// Interpolation
kx=ent(t/xdeltat)+1;
ci=(t-(kx-1)*xdeltat)/xdeltat;
q =x(indq,kx)*ci+x(indq,kx+1)*(1-ci);
qd=x(indqd,kx)*ci+x(indqd,kx+1)*(1-ci);
// Solving the sparse linear system II qdd_lam = h for finding lambda
qdd_lam=iihs(II,h,q,qd,u,param,alfa,beta,gamma);
lambda=qdd_lam(n+1:n+nf+ng);
//Solving the sparse linear system Emat (p2p1mudot) = Fvec
p2p1mudot=efs(Emat,Fvec,q,qd,u,lambda,p1,p2,param,paramopt);
xxdot=[p2p1mudot;hamu(q,qd,u,p1,p2)];
//end

//[p2p1muh_uf]=fcond(qfinal,qdfinal)
// Initialization of the backward integration of the adjoint system
// Solving the sparse linear system Emat p2p1mu = vecfin
p2p1mu=efs(Emat,vecfin,qfinal,qdfinal,zeros(ncontr,1),zeros(nf+ng,1),...
           0*qf,0*qf,param,paramopt));
p2p1muh_uf=[p2p1mu;zeros(ncontr,1)];
//end


          SCILAB MACRO FOR CONSTRUCTING LINEAR MODEL
//[A,B]=linear(q,qd)
//This macro computes the A and B matrices corresponding to the
// system linearized around [q;qd]
qdd_lam=iihs(II,h,q,qd,u,param,alfa,beta,gamma);
qdd=qdd_lam(1:n);lambda=qdd_lam(n+1:nnn);

J_q=jaco(q,qd,qdd,lambda,param);
h_q =hq(q,qd,u,param,alfa,beta,gamma);
h_qd=hqd(q,qd,u,param,alfa,beta,gamma);
h_u =hu(q,qd,u,param,alfa,beta,gamma);
```

```
IIm=ii(q,qd,u,param,alfa,beta,gamma);

X=[eye(n,n),zeros(n,nf+ng)]/IIm;

A=[zeros(n,n) ,       eye(n,n);
    X*(h_q-J_q),        X*h_qd   ];
B=[ zeros(n,ncontr);
       X*h_u ];
//end
```

B Example

In this section we present the Maple data file to be used by the simulation and control package for the case of a bicycle.

```
#                  DATA FOR THE BICYCLE :
#                       L:     Lagrangian
#                       f:     vector of holonomic constraints
#                       gqd:     vector of non holonomic constraints
#                       k:     vector of control actions
#
with(linalg):
# Number of parameters:
mpar:=20:
# Number of generalized coordinates (size of q)
n:=23:
# Number of control variables
ncontr:=2:

# Definition of system parameters and variables:
param:=array(1..mpar):      #parameters
q:=array(1..n):             #generalized coordinates
qd:=array(1..n):            #generalized velocities
u:=array(1..ncontr):        #control variables

# Lagragian L is a function of q[k], qd[k] k=1,..,n and param[.]
#
#                Parameters definition

#param = [wheel radius, rear wheel weight,frame weight, fork weight,
#         front wheel weight, rear wheel radial inertia, frame radial
#         inertia, fork radial inertia, front wheel radial inertia,
#         rear wheel normal inertia, frame normal inertia,
#         fork normal inertia, front wheel normal inertia,
#         acceleration of gravity (9.81m/s/s on earth),|P4-P1|,
#         |P2-P1|,|P3-P4|,2.|P2-P1|,2.|P3-P4|,|P4-P5|]
```

```
#-----------------------------------------------------------------
#                              Rear wheel
#                              ----------
#q[1..6] = position (x,y,z) of P1 and Euler angles (phi,theta,psi)
#qd[1..6]=velocities and angular velocities (time derivatives of q[1..6])
#
# Energy functions for the rear wheel

# Rotational kinetic energy:
Trot:= param[10]/2*(qd[5]^2+qd[4]^2*sin(q[5])^2)+
                    param[6]/2*(qd[4]*cos(q[5])+qd[6])^2:

# Linear kinetic energy:
Ttr:= param[2]/2*(qd[1]^2+qd[2]^2+qd[3]^2):

# Potential energy:
U:=param[2]*param[14]*param[1]*sin(q[5]):

#Rear wheel Lagrangian:

   L1:=Trot+Ttr-U:
#
#-----------------------------------------------------------------
#                               Frame
#                               -----
# q[7..10] = position (x,y,z) of P2 and xi1
# qd[7..10]= time derivatives

# Energy functions

Trot:= param[11]/2*((-qd[5]*sin(q[10])+qd[4]*sin(q[5])*cos(q[10]))^2+
        (qd[4]*cos(q[5])+qd[10])^2)+param[7]/2*(qd[5]*cos(q[10])+
         qd[4]*sin(q[5])*sin(q[10]))^2:

Ttr:= param[3]/2*(qd[7]^2+qd[8]^2+qd[9]^2):

U:=param[3]*param[14]*(param[1]+param[16]*sin(q[10]))*sin(q[5]):

#Lagrangian;

  L2:= Trot+Ttr-U:

#-----------------------------------------------------------------
#                             Front wheel
#                             -----------
#  q[15..20] = position (x,y,z) of P4 and Euler angles
#  qd[15..20]= time derivatives
```

```
#  q[21..23] = position (x,y,z) of P5
#  qd[21..23]= time derivatives

# Energy functions

Trot:= param[13]/2*(qd[19]^2+qd[18]^2*sin(q[19])^2)+
            param[9]/2*(qd[18]*cos(q[19])+qd[20])^2:

Ttr:= param[5]/2*(qd[21]^2+qd[22]^2+qd[23]^2):

U:=param[5]*param[14]*param[1]*sin(q[19]):

#Lagrangian

   L4:=Trot+Ttr-U:

#-----------------------------------------------------------------------
#                                Fork
#                                ----
# q[11..14] = position (x,y,z) of P3 and xi2
# qd[11..14]= time derivatives
#
# Energy functions
#
Trot:=param[12]/2*((-qd[19]*sin(q[14])+qd[18]*sin(q[19])*cos(q[14]))^2+
        (qd[18]*cos(q[19])+qd[14])^2)+param[8]/2*(qd[19]*cos(q[14])
         +qd[18]*sin(q[19])*sin(q[14]))^2:

Ttr:= param[4]/2*(qd[11]^2+qd[12]^2+qd[13]^2):
#
U:=param[4]*param[14]*(param[1]+param[17]*sin(q[14]))*sin(q[19]):
#
#Lagrangian
#
L3:= Trot+Ttr-U:
#
#-----------------------------------------------------------------------
#                              Constraints
#                              -----------
#            (nf = # of holonomic constraints)
#            (ng = # of non holonomic constraints)
#
   nf:=16:
   ng:=4:
#
#                         wheel-earth contact
#                         -------------------
```

```
# Rear wheel
#
f1:=q[3]-param[1]*sin(q[5]):
#
gqd1:=qd[1]+param[1]*(qd[4]*cos(q[5])*cos(q[4])
                     -qd[5]*sin(q[4])*sin(q[5])+qd[6]*cos(q[4])):
gqd2:=qd[2]+param[1]*(qd[4]*cos(q[5])*sin(q[4])
                     +qd[5]*cos(q[4])*sin(q[5])+qd[6]*sin(q[4])):
#-----------
# Front wheel
#
f8:=q[23]-param[1]*sin(q[19]):

gqd4:=qd[21]+param[1]*(qd[18]*cos(q[19])*cos(q[18])-
                   qd[19]*sin(q[18])*sin(q[19])+qd[20]*cos(q[18])):
gqd3:=qd[22]+param[1]*(qd[18]*cos(q[19])*sin(q[18])+
                  qd[19]*cos(q[18])*sin(q[19])+qd[20]*sin(q[18])):
#------------
# Rear wheel-frame and front wheel-fork constraints:
#
f2:= q[1]-q[7]+param[16]*(cos(q[4])*cos(q[10])-
                          sin(q[4])*sin(q[10])*cos(q[5])):
f3:= q[2]-q[8]+param[16]*(sin(q[4])*cos(q[10])+
                          cos(q[4])*sin(q[10])*cos(q[5])):
f4:= q[3]-q[9]+param[16]*(sin(q[10])*sin(q[5])):

f7:= q[15]-q[11]+param[17]*
           (cos(q[18])*cos(q[14])-sin(q[18])*sin(q[14])*cos(q[19])):
f6:= q[16]-q[12]+param[17]*
           (sin(q[18])*cos(q[14])+cos(q[18])*sin(q[14])*cos(q[19])):
f5:= q[17]-q[13]+param[17]*(sin(q[14])*sin(q[19])):


f14:= q[21]-q[15]-param[20]*
          (cos(q[18])*sin(q[14])-cos(q[19])*sin(q[18])*cos(q[14])):

f15:= q[22]-q[16]-param[20]*
          (sin(q[18])*sin(q[14])+cos(q[19])*cos(q[18])*cos(q[14])):

f16:= q[23]-q[17]-param[20]*(-cos(q[14])*sin(q[19])):
#-----------
# Frame-fork constraints:
#
f12:= (q[1]-q[15])^2+(q[2]-q[16])^2+(q[3]-q[17])^2-param[15]^2:

f9:=param[18]*(cos(q[10])*cos(q[4])-sin(q[10])*cos(q[5])*sin(q[4]))
   -param[19]*(cos(q[14])*cos(q[18])-sin(q[14])*cos(q[19])*sin(q[18]))
   -q[15]+q[1]:
```

```
f10:=param[18]*(cos(q[10])*sin(q[4])+sin(q[10])*cos(q[5])*cos(q[4]))
    -param[19]*(cos(q[14])*sin(q[18])+sin(q[14])*cos(q[19])*cos(q[18]))
    -q[16]+q[2]:

f11:=param[18]*sin(q[10])*sin(q[5])-param[19]*
                  sin(q[14])*sin(q[19])-q[17]+q[3]:

f13:=sin(q[5])*sin(q[4])*param[19]*cos(q[14])*cos(q[18])
    - sin(q[5])*sin(q[4])*param[19]*sin(q[14])*cos(q[19])*sin(q[18])
    - sin(q[5])*cos(q[4])*param[19]*cos(q[14])*sin(q[18])
    - sin(q[5])*cos(q[4])*param[19]*sin(q[14])*cos(q[19])*cos(q[18])
    + cos(q[5])*param[19]*sin(q[14])*sin(q[19]):
#-----------------------------------------------------------------------
#
#                          Control forces and torques
#                          --------------------------
#
# Torque applied to the handlebars
# --------------------------------
steerforce:=
vector(n,[0,0,0,
      -(1/(sin(q[19])*sin(q[5])*cos(q[18]-q[4])+
        cos(q[19])*cos(q[5]))) *u[1]*sin(q[5])*cos(q[18]-q[4]),
       (1/(sin(q[19])*sin(q[5])*cos(q[18]-q[4])+cos(q[19])*cos(q[5])))*
        u[1]*cos(q[5])*sin(q[18]-q[4]),
        0,0,0,0,0,0,0,0,
      -(1/(sin(q[19])*sin(q[5])*cos(q[18]-q[4])+
        cos(q[19])*cos(q[5]))) *u[1]*cos(q[14])*(1/sin(q[14]))*sin(q[5])*
        sin(q[18]-q[4]),
        0,0,0,
       (1/(sin(q[19])*sin(q[5])*cos(q[18]-q[4])+cos(q[19])*cos(q[5])))*
        u[1]*sin(q[5])*cos(q[18]-q[4]),0,0,0,0,0]):

# Pedaling torque
# ---------------
pedalforce:=
vector(n,[0,0,0,0,0,u[2],0,0,0,-u[2],0,0,0,0,0,0,0,0,0,0,0,0,0]):
#
# Lagragian for the bicycle
L:= L1+L2+L3+L4:
#
# Holonomic constraint vector
f:=vector(nf,[f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f13,f14,f15,f16]):
#
# Non-holonomic constraint vector
gqd:=vector(ng,[gqd1,gqd2,gqd3,gqd4]):
#
```

```
# Applied forces and torques (controls)
k:=steerforce + pedalforce:
#
```

References

[1] K. E. Branan, S. L. Campbell and L. R. Petzold, Numerical Solutions of initial-Value Problems in Differential-Algebraic Equations, North Holland, 1989.

[2] S. L. Campbell, A Computational Method for General Higher Index Nonlinear Singular Systems of Differential Equations, Proc. IMACS World Congress on Scientific Computation, Paris, 1988.

[3] B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan and S. M. Watt, Maple V Language Reference Manual, Springer Verlag, New York, 1991.

[4] C. Gomez, Macrofort: A Fortran Code Generator in Maple, Inria Technical Report 119, 1990.

[5] F. Delebecque, C. Klimann and S. Steer, Basile: Guide de l'utilisateur, INRIA Press, Paris, 1989.

[6] C. Gomez, G. Le Vey and C. Rougerie, Basile-Maple Interface: a Link Between CACSD Package and Computer Algebra System, INRIA report 130, 1991.

[7] C. W. Gear, Differential-Algebraic Equations,in Computer Aided Analysis and Optimization of Mechanical System Dynamics, Springer-Verlag, 1984.

[8] P. E. Nikravesh, Some Methods of Dynamic Analysis Of Constrained Mechanical Systems: A Survey, in Computer Aided Analysis and Optimization of Mechanical System Dynamics, Springer-Verlag, 1984.

[9] C. Lanczos, The Variational Principles Of Mechanics, University of Toronto Press. Toronto, 1960.

[10] C. Bunks and R. Nikoukhah, Optimal Control of Constrained Mechanical Systems, Proc. 8th IFAC Workshop, Paris, 1989.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249-6399