

## A SYMBOLIC APPROACH FOR AUTOMATIC GENERATION OF THE EQUATIONS OF MOTION OF MULTIBODY SYSTEMS

R. Lot\* and M. Da Lio†

\* Dept. of Mechanical Engineering (DIM)  
University of Padova  
Via Venezia 1, 35131 Padova (Italy)  
e-mail: roberto.lot@unipd.it, web page: <http://www.dim.unipd/lot/>

† Dept. of Mechanical and Structural Engineering (DIME)  
University of Trento  
Via Mesiano 77, 38050 Trento (Italy)  
e-mail: mauro.dalio@ing.unitn.it

**Keywords:** Symbolic Multibody Dynamics, Vehicle Handling and Stability

**Abstract.** *This paper describes a collection of methods and procedures for the automatic generation of the equations of motion of multibody systems using general-purpose Computer Algebra Software. A brief review of existing symbolic multibody systems is given and advantages and disadvantages of symbolic approaches compared with numerical ones are discussed. Then, a set of methods for symbolic modeling of multibody systems is explained. The first step of the modeling procedure consists of the description of the multibody system, by defining objects (such as points, vectors, rigid bodies, forces and torques, special objects) and the relationships between them (kinematic chains, constraints). The second step is the derivation of the equations of motion, which can be performed in a quasi-automatic way. A further step is the linearization of the equations and the calculation of the system's frequency response functions. By way of example, a dynamic model of the motorcycle is developed, obtaining the non linear equations of motion in a dependent coordinates' formulation. Next, the equations of motion are linearized and reduced to an independent formulation, re-obtaining the well known Sharp's<sup>19</sup> model of the straight running of the motorcycle. Root loci and frequency response functions are also calculated. This example demonstrates the power of the given symbolic procedures and shows how a model suitable for stability, handling and control analysis can be developed quickly and easily. The procedure described in this paper has been implemented in a Maple package called 'MBSymba', which is available on the web page [www.dim.unipd.it/lot/MBSymba.html](http://www.dim.unipd.it/lot/MBSymba.html).*

## 1 INTRODUCTION

Nowadays, the panorama of software for the modeling of multibody system is very rich. Many modeling environments are based on a numerical approach, such as ADAMS, DADS, Visual Nastran 4D, FEDEM, to name but a few. These software are geared towards the product's final design and offer the detailed modeling and simulations of mechanical systems, also using a large number of degrees of freedom. They are often integrated with FEM modeling software and with Virtual Prototyping–CAD environments, in addition sub-models for important subsystems (e.g.: tires, hydraulics actuator, and many others) may be available.

The alternative approach is the symbolic development of the equations of motion, which can be derived from a plethora of different techniques<sup>1-16</sup>, but are essentially carried out in a fashion similar to hand derivation. Systems based on the symbolic approach may be either stand-alone environments, or add-ons of general-purpose Computer Algebra Software (CAS) like Mathematica and Maple. Stand alone modeling environments allow a description of the multibody system to be modeled and develop the equations of motion, which in most cases are converted into a C or FORTRAN source code for producing highly specialized simulation programs. Examples of these are Autolev, SD/Fast, NEWEUL, Autosim. Similar to the numerical systems mentioned above, these packages offer the possibility of describing multibody systems in detail, but the equations of motion are simplified before being converted into a source code, thereby obtaining custom-built and extremely fast simulation codes. In addition, some systems can also develop linearized equations, suitable for the synthesis of control systems. Equations can indeed be inspected, and/or exported, but their use and further symbolic manipulation is not the primary objective. Rather, the manipulation of the equations of the mathematical model is the most natural use of the add-on tools of general-purpose CAS, like TsiProPac for Mathematica, Dynaflex for Maple, and a few others. These tools appear most suited to the development and manipulation of “essential” mathematical models that capture the most important features of the system, and especially suited to the early design phases, i.e. preliminary and conceptual design, or to produce simple, essential and effective mathematical models for control design<sup>1-17</sup>. These tools provide the user with a low level of control of the model, such as the choice of the most suitable coordinates set, the handling of linearization and simplifications. They offer the possibility of post processing the equations, for example they may be transformed into a representation most suitable for control systems design algorithms. These tools may also produce C and Fortran source codes like the stand alone symbolic system mentioned above. The use of a general-purpose CAS is nevertheless only an aid in the manipulation of the expression and does not substitute user experience and skills; specialized training of the user is of paramount importance.

## 2 DESCRIPTION OF THE MULTIBODY SYSTEM

The key step in modeling a multibody system is the description of the position of the bodies, which is carried out by attaching a reference frame to each body and by choosing the coordinates set. The choice of the proper coordinates system is essential in the building of mathematical models that need to be both small and effective, and can result in remarkable improvements of efficiency, usability and performance. A minimum set of lagrangian coordinates is not necessarily the best choice: as the kinematic chain gets longer and angular displacements become involved, kinematical expressions reach intolerable complexity. Commercial software tends to use a different approach based on a predefined set of coordinates: each body is described by means of its coordinates' subset, always using the

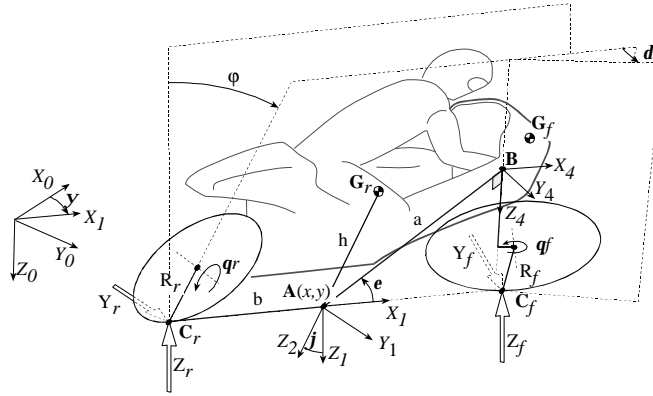


Figure 1 – essential description of the motorcycle model

same criteria (typically the three coordinates of base point plus the four Euler parameter for the attitude are used). This method allows easy assembly of equations, but there are a lot more coordinates than degrees of freedom and consequently there are many algebraic constraint equations. An effective way of reducing the number of dependent coordinates is by the sharing of coordinates at the joints between bodies, or rather the use of only Cartesian coordinates (not rotations) results in simpler constraint equations. The book by Jalon and Bayo<sup>18</sup>, among others, gives a very clear discussion of the options between different types of coordinates. Selecting the most efficient coordinate system is a process that can be very efficient if done by the user, and if they are modeling the system by means of a CAS rich of features, their skills are paramount.

Some methods for easily describing a multibody system are explained below. These method have been implemented in a Maple package ([www.maplesoft.com](http://www.maplesoft.com)), called *MBSymba*. There are some procedures for describing objects (such as points, vectors, rigid bodies, forces and torques) and other procedures which simplify the symbolic manipulation of objects. Due to constraints of space, this paper only describes a subset of the available procedures, however their complete list is given in the Appendix and their complete definition is available on the web page [www.dim.unipd.it/lot/MBSymba.html](http://www.dim.unipd.it/lot/MBSymba.html).

As a case study, a multibody model of a motorcycle is built. This model consists of four rigid bodies (the rear assembly, the front assembly and the wheels, as shown in figure 1) and it is well documented in the literature<sup>19</sup>. The same techniques were also employed in developing a much complex model of the motorcycle<sup>17</sup>.

## 2.1 Reference Frames

A reference frame  $i$  may be defined by means of the coordinates of its origin and the director cosines of its axes in a given absolute reference frame 0. It is useful to pack the above information into a square  $4 \times 4$  matrix  $\mathbf{T}_0^i$ , whose  $3 \times 3$  upper left is the rotation sub-matrix  $\mathbf{R}_0^i$ , which contains the director cosines of the axes of  $i$  as seen from 0, and the upper right  $3 \times 1$  is the origin vector  $\mathbf{b}_0^i$  (see references<sup>11,13,20,21</sup>)

$$\mathbf{T}_0^i = \begin{bmatrix} \mathbf{R}_0^i & \mathbf{b}_0^i \\ 000 & 1 \end{bmatrix} \quad (1)$$

The expression above contains all the information required to define the reference frame  $i$ , and may thus be considered a representation of it. In addition, the equation above may also be read as the rigid body motion which moves frame 0 to  $i$ . By referring to a generic frame  $j$ , the transformation matrix  $\mathbf{T}_j^i$  defines the frame  $i$  with respect to the frame  $j$  and corresponds to the rigid body motion that superimposes the frame  $j$  to the frame  $i$ . Moreover, if a third reference frame  $k$  is considered, the composition of rigid motions is obtained simply by multiplying the matrices associates to each reference frame, as follows:

$$\mathbf{T}_k^j = \mathbf{T}_k^i \mathbf{T}_i^j \quad (2)$$

Equations 1 and 2 give the tools needed for modeling reference frames. A simple way of doing it using a CAS is the definition of a set of commands that define a transformation matrix using simple rigid body motions, like translation and rotations about the three frame axes. The following procedures are valid in Maple:

```
> translate := proc(x,y,z) matrix([[1,0,0,x],[0,1,0,y],[0,0,1,z],[0,0,0,1]]): end:
> rotate := proc(axis,angle)
    if axis='X' then matrix([[1,0,0,0],[0,cos(angle),-sin(angle),0],
        [0,sin(angle),cos(angle),0], [0,0,0,1]]):
    elif axis='Y' then matrix([[cos(angle),0,sin(angle),0],[0,1,0,0],
        [-sin(angle),0,cos(angle),0],[0,0,0,1]]):
    elif axis='Z' then matrix([[cos(angle),-sin(angle),0,0],
        [sin(angle),cos(angle),0,0],[0,0,1,0],[0,0,0,1]]):
    else ERROR("invalid rotation axis"): end if:
end:
```

Using the procedures above, frames  $\mathbf{T}_1$ ,  $\mathbf{T}_2$  (fixed to the rear chassis) and  $\mathbf{T}_4$  (fixed to the fork) of reference<sup>19</sup> can be easily defined as follows:

```
> alias(c=cos,s=sin):
T1 := translate(x(t),y(t),0) * rotate('Z',psi(t)):
T2 := T1 * rotate('X',phi(t));
```

$$T2 := \begin{bmatrix} c(\psi(t)) & -s(\psi(t))c(\phi(t)) & s(\psi(t))s(\phi(t)) & x(t) \\ s(\psi(t)) & c(\psi(t))c(\phi(t)) & -c(\psi(t))s(\phi(t)) & y(t) \\ 0 & s(\phi(t)) & c(\phi(t)) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
> T4 := T2 * rotate('Y',epsilon) * translate(a,0,0) * rotate('Z',delta(t)):
```

where functions ‘c’ and ‘s’ are used for brevity instead of ‘sin’ and ‘cos’. In this case the reference frames are defined as a function of the coordinates  $x, y, \mathbf{y}, \mathbf{j}, \mathbf{d}$ , which correspond to the system’s degrees of freedom. As shown in figure 1, they are respectively the coordinates of point A and the yaw, roll and steering angles. Many other coordinates or sets of displacement parameters can be used for defining the transformation matrix (1): for example, a frame can be defined using the Denavit-Hartenberg parameters<sup>22</sup>, or the four Euler parameters, or the “natural coordinates” of given key points and vectors<sup>18</sup>.

## 2.2 Points and vectors

Points are defined by means of a set of three coordinates plus a reference frame in which the coordinates must be interpreted. Similarly, vectors are defined by means of a set of three director cosines plus a frame. Using the homogeneous representation<sup>20</sup>, they are represented by means of a 4x1 matrix, which includes the three coordinates plus a fourth element equal to

one for points and equal to zero for vectors. In Maple, this information may be embedded into a structured object called “table”, as follows:

```
> make_POINT := proc(TT, x,y,z)
    table ([ obj=POINT, frame=TT, coords=matrix([ [x], [y], [z], [1]]) ]):
end:
```

For example, the center of mass of the rear frame point  $\mathbf{G}_r$  may be defined using its coordinates with respect the rear chassis frame  $\mathbf{T}_2$  as follows:

$$Gr := \text{table}([\text{frame} = T2, \text{obj} = \text{POINT}, \text{coords} = \begin{bmatrix} 0 \\ 0 \\ -h \\ 1 \end{bmatrix}]) \quad Gr = \text{POINT} \left( \text{frame} = T2, \text{coords} = \begin{bmatrix} 0 \\ 0 \\ -h \\ 1 \end{bmatrix} \right)$$

where the `show()` procedure may be used for displaying it in a more concise and elegant form.

A first advantage in using structured objects is that all the object properties are embedded in a unique variable, thus any misunderstanding about the coordinates of a point and the reference frame in which they are defined is avoided. Secondly, new methods may be easily introduced to handle new objects. Indeed, the homogeneous representation makes it possible to easily convert the coordinates expressed in a reference frame  $j$  into the coordinates expressed in a reference frame  $i$  simply by pre-multiplying the 4x1 matrix of a generic point  $\mathbf{P}$  or a vector  $\mathbf{v}$  by the 4x4 transformation matrix  $\mathbf{T}_j^i$ , as follows:

$$\mathbf{P}^i = \mathbf{T}_j^i \mathbf{P}^j, \quad \mathbf{v}^i = \mathbf{T}_j^i \mathbf{v}^j \quad (3)$$

This operation may be converted into the following Maple procedure:

```
> project := proc (A::{POINT,VECTOR},TT::frame)
    local B,cc:
    B:=copy(A):
    if B[frame] <> TT then
        B[frame]:=TT:
        if type(B,'POINT') then cc:= coords else cc:=comps: end if:
        B[cc]:=map(simplify,evalm(inv_frame(TT)*A[frame]*A[cc]),trig):
    end if:
    copy(B):
end:
```

The procedure above operates only on the ‘POINT’ and ‘VECTOR’ objects, which are defined in Maple using the Boolean function ‘type’ as follows:

```
> `type/POINT` := x -> has({indices(x)},{obj}) and evalb(x[obj]='POINT'):
> type(A,POINT), type(A,VECTOR):
```

*true, false*

Moreover, the sum between two vectors or between a vector and a point may be defined by rebinding the ‘+’ operator:

```
> `+` := proc(A,B)
    if type([A,B],{['POINT','VECTOR'],['VECTOR','POINT'],['VECTOR','VECTOR']}) then
        if A[frame]<>B[frame] then PB:=project(B,A[frame]) else PB:=copy(B): end if:
        if type([A,B],['VECTOR','VECTOR']) then
            make_VECTOR(A[frame],op([all_comps(A)]+[all_comps(PB)]));
        else make_POINT(A[frame],op([all_comps(A)]+[all_comps(PB)])); end if:
    else A+B: end if: # standard addition
end:
```

The procedure above does not only the sum of the coordinates of points and vectors, but also performs a change of reference frame if necessary, then it creates a new vector from the sum of two vectors or a new point from the sum of a point and a vector. On the contrary, the sum of two points is not allowed. For example, the point  $\mathbf{G}_r$  introduced above may also be defined by adding the vector  $\mathbf{AC}$  to the origin of the frame  $\mathbf{T}_2$ , as follows:

```
> origin(T2) + make_VECTOR(T2,0,0,-h): show(%);
```

$$\text{POINT} \left( \begin{array}{l} \text{frame} = T2, \text{coords} = \begin{bmatrix} 0 \\ 0 \\ -h \\ 1 \end{bmatrix} \end{array} \right)$$

In the expressions above, the time dependency of the variables is explicitly indicated. This trick makes it possible to use the standard differentiation procedures available in Maple (or any other CAS) to obtain velocities and accelerations simply by derivation with respect to  $t$ . Indeed, the velocity of a point (or a vector) may be calculated by deriving expressions (3) with respect to the time. By referring to the inertial frame  $i=0$ , one obtains the following expressions

$$\dot{\mathbf{P}}^0 = \dot{\mathbf{T}}_j^i \mathbf{P}^j + \mathbf{T}_j^i \dot{\mathbf{P}}^j \quad (4)$$

$$\dot{\mathbf{v}}^i = \dot{\mathbf{T}}_j^i \mathbf{v}^j + \mathbf{T}_j^i \dot{\mathbf{v}}^j \quad (5)$$

By examining the terms of equation (4), it may be seen that the left hand side is the absolute velocity of the point  $\mathbf{P}$ . The first term on the right hand side corresponds to the velocity of a point  $\mathbf{Q}$  fixed to the reference frame  $\mathbf{T}_j^i$ , which is instantaneously coincident with the point  $\mathbf{P}$  (this term is also called ‘eulerian’ velocity). The second term of the right hand side corresponds to its the relative velocity. These concepts may be easily included in a Maple procedure as follows:

```
> velocity := proc(P::{POINT,VECTOR},veltype)
  if veltype='absolute' then
    PP:=project(P,ground):
    make_VECTOR(ground, op(map(diff,[all_comps(PP)],t))):
  elif veltype='relative' then
    make_VECTOR(P[frame], op(map(diff,[all_comps(P)],t))):
  elif veltype='eulerian' then
    VE := map(simplify,evalm( map(diff,P[frame],t) &* P[coords] ),trig);
    make_VECTOR(ground, VE[1,1],VE[2,1],VE[3,1]):
  else ERROR ("invalid velocity specification") end if:
end:
```

Similarly to the above, the angular velocity of a reference frame  $\mathbf{T}$  may be calculated by extracting its components from the angular velocity matrix<sup>20</sup> which stem from the rotational submatrix  $\mathbf{R} = \mathbf{T}_{1..3,1..3}$  as follows:

$$\dot{\mathbf{R}} \mathbf{R}^T = \begin{bmatrix} 0 & -\mathbf{w}_z & +\mathbf{w}_y \\ +\mathbf{w}_z & 0 & -\mathbf{w}_x \\ -\mathbf{w}_y & +\mathbf{w}_x & 0 \end{bmatrix} \quad (6)$$

## 2.3 Bodies

The properties of a rigid body can be collected in a table containing the reference frame which the body is attached to, the coordinates of the center of mass and the inertia tensor. Rigid bodies can be defined in *MBSymba* using the `make_BODY()` command, for example the rear chassis may be defined as follows:

```
> rear_chassis := make_BODY(Gr,mr,Irx,Iry,Irz,0,Crxz,0): show(rear_chassis);
```

$$\text{rear\_chassis} = \text{BODY} \left( \text{frame} = T2, \text{CM} = \begin{bmatrix} 0 \\ 0 \\ -h \\ 1 \end{bmatrix}, \text{mass} = mr, \text{inertia} = \begin{bmatrix} I_{rx} & 0 & C_{rxz} \\ 0 & I_{ry} & 0 \\ C_{rxz} & 0 & I_{rz} \end{bmatrix} \right)$$

In order to complete the model of the motorcycle it is necessary to define also the front chassis and the wheels, as follows:

```
> TWR := T2 * translate(-b,0,-Rr) * rotate('Y',thetal(t)): # rear wheel frame
rear_wheel := make_BODY(TWR,0,0,iry,0):
> Gf := make_POINT(T4,e,0,f): # front chassis center of mass
front_chassis := make_BODY(Gf,mf,Ifx,0,Ifz):
> E := project( make_POINT(T4,lx,0,lz) ,ground): # front wheel center
TWF := translate(all_comps(E)) * rotate('Z',psi4(t)) * rotate('X',phi4(t)) *
rotate('Y',theta4(t)): #front wheel frame
front_wheel := make_BODY(TWF,0,0,ify,0):
```

when coordinates  $\mathbf{q}_1, \mathbf{q}_4$  represent the wheel spin rotations. It should be observed that the definition of the front wheel does not make use of the kinematic chain from frame  $\mathbf{T}_1$  to frame  $\mathbf{T}_{WF}$ , but uses the new coordinates  $\mathbf{j}_4, \mathbf{y}_4$  which correspond respectively to the front camber and the front yaw angles. This choice makes it possible to obtain much simpler kinematic expressions, but has the (slight) disadvantage of requiring two additional relations for describing the dependency among coordinates, as will be explained in detail below.

## 2.4 Forces and torques

A force is defined by a vector, which describes its magnitude and orientation, plus the application point. In order to calculate the power (or the generalized components) of a force it is also necessary to know the bodies to which the force is acting. All this information may be packed in a table using the `make_FORCE()` command. For example, tire forces can be defined as follows:

```
> Cr := make_POINT(TWR, -Rr*sin(thetal(t)) , 0 , Rr*cos(thetal(t))): #contact point
rear_tire_forces := make_FORCE (rotate('Z',psi(t)), 0,Yr(t),Zr, Cr, rear_wheel):
show(rear_tire_forces);
```

$$\text{rear\_tire\_forces} = \text{FORCE} \left( \text{frame} = \begin{bmatrix} c(\psi(t)) & -s(\psi(t)) & 0 & 0 \\ s(\psi(t)) & c(\psi(t)) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{applied} = Cr, \text{acting} = \text{rear\_wheel}, \text{comps} = \begin{bmatrix} 0 \\ Yr(t) \\ Zr \\ 0 \end{bmatrix} \right)$$

```
> Cf := make_POINT(TWF, -Rf*sin(theta4(t)) , 0 , Rf*cos(theta4(t))): #contact point
front_tire_forces := make_FORCE (rotate('Z',psi4(t)), 0,Yf(t),Zf, Cf, front_wheel):
```

A torque can be defined similarly, for example the torque exerted by the rider on the handlebar plus the torque due to the steering damper are defined as follows:

```
> steering_torque := make_TORQUE(T4, 0,0, tau(t)-C[delta]*diff(delta(t),t),
front_chassis,rear_chassis): show(steering_torque);
```

$$\text{steering\_torque} = \text{TORQUE} \left( \text{frame} = T4, \text{reacting} = \text{rear\_chassis}, \text{acting} = \text{front\_chassis}, \text{comps} = \begin{bmatrix} 0 \\ 0 \\ \tau(t) - C_{\delta} \left( \frac{\partial}{\partial t} \delta(t) \right) \\ 0 \end{bmatrix} \right)$$

## 2.5 Constraints

There are several techniques for the automating managing of constraints in multibody models<sup>1-15,18,21</sup>, but their review exceeds the aim of this paper. However, a CAS system always offers the possibility of defining constraints by using algebraic or differential equations, respectively for holonomic and non-holonomic constraints. For example, the dependency among coordinates  $\mathbf{y}, \mathbf{j}, \mathbf{d}, \mathbf{y}_4, \mathbf{j}_4$  may be explicated by considering two equivalent definitions of the front spin axis, as follows:

```
> spin_axis := project( make_VECTOR(TWF,0,1,0) , ground):
spin_axis_copy := project( make_VECTOR(T4 ,0,1,0) , ground):
```

by equaling both X and Z components of the unit vectors above, two independent holonomic constraint equations may be obtained as follows:

```
> spin_x := Xcomp(spin_axis)- Xcomp(spin_axis_copy);
spin_z := Zcomp(spin_axis)- Zcomp(spin_axis_copy);
```

$$\begin{aligned} \text{spin}_X &:= s(\delta(t)) c(\psi(t)) c(\varepsilon) - s(\delta(t)) s(\psi(t)) s(\phi(t)) s(\varepsilon) + s(\psi(t)) c(\phi(t)) c(\delta(t)) - s(\psi_4(t)) c(\phi_4(t)) \\ \text{spin}_Z &:= -c(\phi(t)) s(\varepsilon) s(\delta(t)) - s(\phi(t)) c(\delta(t)) + s(\phi_4(t)) \end{aligned}$$

whereas the third condition on the Y components would be redundant.

Two additional equations are also needed to correlate the wheel spin rotations  $\mathbf{q}_1, \mathbf{q}_4$  to the remaining coordinates. The absence of the longitudinal slip condition may be easily made explicit by calculating the eulerian velocity of each contact point, and the following non-holonomic constraint equations are obtained:

```
> VCr_euler := project(velocity(Cr, 'eulerian'), T1):
no_slip_rear := Xcomp(VCr_euler);
```

$$\text{no\_slip\_rear} := c(\psi(t)) \left( \frac{\partial}{\partial t} x(t) \right) + Rr \left( \frac{\partial}{\partial t} \theta_1(t) \right) + s(\psi(t)) \left( \frac{\partial}{\partial t} y(t) \right)$$

```
> Vcf_euler := project(velocity(Cf, 'eulerian'), rotate('Z', psi4(t))):
no_slip_front := Xcomp(Vcf_euler):
```

The main advantage of this approach is that the user has the maximum flexibility in defining the constraint equations. This requires skill and experience, but makes it possible to reach a great simplicity and efficiency in the description of the system. Anyway, it will not be difficult to define some procedures for automatic handling the most common constraints, such as coincident points and axes, orthonormal properties of frame matrices, basic kinematic pairs, etc. In all cases, constraint equations can be collect in a set of algebraic (for holonomic constraints) and differential (for non holonomic constraints) equations as follows:

$$\mathbf{f}_j(\mathbf{q}, \dot{\mathbf{q}}, t) \quad j = 1..c \quad (7)$$

where  $\mathbf{q}$  is the vector of coordinates and  $c$  is the number of constraint equations.

## 2.6 Additional Equations

It is not uncommon to include some additional variables and equations in the model, which may define the behavior of particular sub-system (for example tire and suspension in the field of vehicle dynamics), or for defining a control sub-system. These equations may be both algebraic or differential and can be expressed as follows

$$h_k(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{p}, \dot{\mathbf{p}}, t) = 0 \quad k = 1..c' \quad (8)$$

$\mathbf{p}$  is a vector of new variables, which obviously are in the same number as the new equations. These equations are very similar to constraint equations (7), but they are treated in a separated way since the auxiliary variables  $\mathbf{p}$  do not affect the expression of kinetic or potential energy and thus they do not originate any Lagrange's equation, as shown below.

In the present motorcycle model, equations (8) are used for modeling the tire behavior, in particular the following couple of well know relaxation equations is used for describing the relationship between tire lateral forces and the generalized coordinates:

```
> VCr_abs:=project(velocity(Cr,'absolute'),Tl);
Vsr := Xcomp(VCr_abs): Vnr := Ycomp(VCr_abs):
equ_Yr := sigma/'Vsr'*diff(Yr(t),t)+Yr(t) - (Cr1*arctan('-Vnr'/'Vsr')+Cr2*phi(t));
```

$$\frac{\sigma \left( \frac{\partial}{\partial t} Yr(t) \right)}{c(\psi(t)) \left( \frac{\partial}{\partial t} x(t) \right) + s(\psi(t)) \left( \frac{\partial}{\partial t} y(t) \right)} + Yr(t) + Cr1 \arctan \left( \frac{- \left( \frac{\partial}{\partial t} \psi(t) \right) b - s(\psi(t)) \left( \frac{\partial}{\partial t} x(t) \right) + c(\psi(t)) \left( \frac{\partial}{\partial t} y(t) \right)}{c(\psi(t)) \left( \frac{\partial}{\partial t} x(t) \right) + s(\psi(t)) \left( \frac{\partial}{\partial t} y(t) \right)} \right) - Cr2 \phi(t)$$

```
> Vcf_abs := project(velocity(Cf,'absolute'),rotate('Z',psi4(t)));
VSf := Xcomp(Vcf_abs): VNf := Ycomp(Vcf_abs):
equ_Yf := sigma/'VSf'*diff(Yf(t),t)+Yf(t) - (Cf1*arctan('-VNf'/'VSf')+Cf2*phi4(t));
```

$$equ\_Yf := \frac{\sigma \left( \frac{\partial}{\partial t} Yf(t) \right)}{VSf} + Yf(t) + Cf1 \arctan \left( \frac{VNf}{VSf} \right) - Cf2 \phi4(t)$$

## 3 DERIVATION OF THE EQUATIONS OF MOTION

All procedures defined above serve mainly in the description of the system. They are very flexible and leave the user several possibilities for describing the same system, according to their skill and experience. The next step is the definition of commands for deriving the equations of motion on the basis of the information stored in the objects defined above. Once the system has been described and a set of coordinates has been established, there are still many methods for deriving the equations of motion<sup>20-24</sup>, which range from direct application of Newton's laws to methods of analytical mechanics, like Lagrange's or Hamilton's equations. However, each method can be codified in a standard sequence of operations and therefore the equations of motion can be automatically derived. In this paper a method based on the Lagrange's equations is illustrated and the sections below describe a minimum set of commands for deriving the model equations.

### 3.1 Kinetic Energy

The procedure for calculating the kinetic energy of a body is quite simple, it requires the calculation of the velocity of the center of mass and the angular velocity of the frame fixed to

the body, which can be computed using the procedures illustrated above. For example, the kinetic energy of the rear wheel can be calculated as:

```
> kinetic_energy(rear_wheel);
```

$$\frac{1}{2} \left( \left( \frac{\partial}{\partial t} \theta_1(t) \right) + s(\phi(t)) \left( \frac{\partial}{\partial t} \psi(t) \right) \right)^2 i_{ry}$$

obtaining a very simple expression. This expression shows the great superiority of the symbolic approach with respect to the numeric one in obtaining an optimized formulation of the equations of motion. Indeed, the symbolic approach makes it possible to perform expression simplifications at two levels: during the definition of the model, when the non-significant properties of objects are eliminated at the origin by the user (in the case above only the wheel inertia  $i_{ry}$  around the spin is taken into account) and during the manipulation of objects, when mathematical simplifications (i.e. trigonometry relations and many other) are automatically performed by the CAS.

### 3.2 Generalized Forces

The information stored in a 'FORCE' object are sufficient for calculating the generalized force with respect to a given coordinate. Basing on the concept of virtual work<sup>17,20-24</sup> the following Maple procedures calculate the generalized force of a force FT with respect a coordinate  $q$ :

```
> generalized_force := proc(FT,q)
  FV := make_VECTOR(FT[frame],Comps(FT)): #force vector
  TA := FT[acting][frame]: # acting frame
  PA := project(FT[applied],TA #project P to the acting body frame
  DTA := evalm( map(diffF,TA,q)*PA[coords]):
  DPA := make_VECTOR(ground,DTA[1,1],DTA[2,1],DTA[3,1]): # virtual displacement
  if has({indices(FT)},[reacting])then
    TR := FT[reacting][frame]: else TR := ground: fi: # reacting frame
  PR := project(FT[applied],TR): #project P to the reacting body frame
  DPR := evalm( map(diffF,TR,q)*PR[coords]): # virtual displacement
  DPR := make_VECTOR(ground,DPR[1,1],DPR[2,1],DPR[3,1]):
  FF := dot_prod(DPA,FV) - dot_prod(DPR,FV): #generalized force
end:
```

The generalized force associated to a torque may be calculated in a similar way, obtaining a general and powerful procedure which makes it possible to calculate, for example, the following expression of the generalized force of the front tire force with respect to the lateral displacement of the vehicle, and the generalized force of the steering torque with respect to the steering angle:

```
> generalized_force(front_tire_forces,y(t)),
  generalized_force(steering_torque,delta(t));
```

$$c(\psi_4(t)) Yf(t), \tau(t) - C_\delta \left( \frac{\partial}{\partial t} \delta(t) \right)$$

Once again, these expressions are very simple because of the careful choice of the set of the coordinates (simplification due to the user) and the symbolic simplification of trigonometric terms (simplification due to the CAS).

### 3.3 Lagrange's Equations

The Lagrangian function of a mechanical system subject to constraint equations (7) is defined as follows<sup>24</sup>:

$$L = T(\mathbf{q}, \dot{\mathbf{q}}, t) - V(\mathbf{q}) + \sum_{j=1}^c \mathbf{l}_j \mathbf{f}_j(\mathbf{q}, \dot{\mathbf{q}}, t) \quad (9)$$

where  $\mathbf{q}$  is the vector of  $n$  generalized coordinates,  $K$  is the kinetic energy,  $V$  is the potential of conservative forces and  $\mathbf{l}_j$  are the Lagrange's multipliers. The motion of the system is described by means of the following set of 2<sup>nd</sup> order Lagrange's equations:

$$\frac{d}{dt} \left( \frac{\partial L}{\partial \dot{q}_i} \right) - \frac{\partial L}{\partial q_i} = Q_i \quad i = 1..n \quad (10)$$

where  $Q_i$  are the generalized forces. Equations above may be automatically calculated using the following Maple procedure:

```
> lagrange_equations := proc(_bodies,_forces,_constraints,coords,t,gamma)
  local KE,GE,Qforce,LL:
  KE := kinetic_energy(_bodies):           #Kinetic Energy
  GE := gravity_energy(_bodies):           #Gravitational Energy
  Qforce := generalized_force(_forces,_coords): #Generalized Forces
  # Lagrangian Functions and Lagrange's Equations:
  LL := KE-GE + sum('gamma'[k(t)]*_constraints['k'],'k'=1..nops(_constraints)):
  [seq(lagrange(LL,_coords[i],t) - Qforce[i] , i=1..nops(_coords))]:
end:
```

For the motorcycle model considered, is it only necessary to collect bodies, forces and constraints as follows:

```
> bodies := [rear_wheel,rear_chassis,front_chassis,front_wheel]:
forces := [rear_tire_forces,front_tire_forces,steering_torque]:
constraints := ['no_slip_rear','no_slip_front','front_camber','front_yaw']:
```

to define the generalized coordinates:

```
> qq := [x(t),y(t),psi(t),delta(t),phi(t),psi4(t),phi4(t),theta1(t),theta4(t)]:
```

$$qq := [x(t), y(t), \psi(t), \delta(t), \phi(t), \psi4(t), \phi4(t), \theta1(t), \theta4(t)]$$

and then the Lagrange's equations may be automatically derived, avoiding any long, and error susceptible hand calculations:

```
> leqns := lagrange_equations(bodies,forces,constraints,qq,t,gamma):
```

An index of the complexity of the equations of motion may be represented by the number of the operations needed for their evaluation, which are:

```
> codegen[cost](leqns);
```

*531 additions + 4110 functions + 2983 multiplications + subscripts*

This level of complexity is much greater than the level which may be reached by the hand derivations of the equations. At the same time, this number of operations is much smaller than the number of operations that would be required using a numerical approach, because the latter cannot perform simplification.

### 3.4 Non Linear Model Equations

The union of Lagrange's equations (10), constraint equations (7) and additional equations (8) constitute a set of  $m=n+c+c'$  2<sup>nd</sup> order differential-algebraic equations (DAE's), having a corresponding number of unknown  $\mathbf{y}=\{\mathbf{q}, \boldsymbol{\lambda}, \mathbf{p}\}$ :  $n$  generalized coordinates,  $c$  Lagrange's multipliers and  $c'$  additional variables, as follows<sup>18, 21-24</sup>

$$\mathbf{G}(\mathbf{y}, \dot{\mathbf{y}}, t) = \left\{ \begin{array}{l} \frac{d}{dt} \left( \frac{\partial L}{\partial \dot{q}_i} \right) - \left( \frac{\partial L}{\partial q_i} \right) = Q_i \\ \mathbf{f}_j \\ p_k \end{array} \right\} \quad (11)$$

In the example being considered, there are altogether  $m=9+4+2=15$  equations and the unknowns are:

$$[x(t), y(t), \psi(t), \delta(t), \phi(t), \psi_4(t), \phi_4(t), \theta_1(t), \theta_4(t), \gamma_1(t), \gamma_2(t), \gamma_3(t), \gamma_4(t), Y_r(t), Y_f(t)] \quad (12)$$

There are several techniques available for integrating the above set of equations<sup>25</sup> or for converting them into a set of ordinary differential equations<sup>18</sup> (ODEs), but this question exceeds the aim of this paper. In all cases, the CAS makes it possible to easily handle the equations and automatically convert them into a C or Fortran source.

## 4 LINEAR ANALYSIS

The linearization of the equations of motion makes it possible to obtain a model of reasonable complexity, which may be profitably used for stability investigation, analysis in frequency domain, control design and optimization. A basic set of commands for linearizing and manipulating equations is illustrated below; as an example they are used for simplifying the above equations of the motorcycle model and for re-deriving the well known Sharp's motorcycle model<sup>19</sup>.

### 4.1 Linearization of the Equations of Motion

The `linearize()` command makes it possible to linearize the given expression with respect to the given expansion point, taking into account also 1<sup>st</sup> and 2<sup>nd</sup> derivative of the variables with respect to the independent variable. For the motorcycle, it is first necessary to define the equilibrium configuration as the straight running motion at constant speed  $V$ , as follows:

```
> straight_run := [ x(t)=V*t , y(t)=0, psi(t)=0, phi(t)=0, delta(t)=0, psi4(t)=0,
  phi4(t)=0, thetal(t)=-V*t/Rr, theta4(t)=-V*t/Rf, Yr(t)=0, Yf(t)=0,
  seq(gamma[i](t)=0, i=1..nops(constraints)) ];
```

$$\text{straight\_run} := \left[ \begin{array}{l} x(t) = Vt, y(t) = 0, \psi(t) = 0, \phi(t) = 0, \delta(t) = 0, \psi_4(t) = 0, \phi_4(t) = 0, \theta_1(t) = -\frac{Vt}{Rr}, \\ \theta_4(t) = -\frac{Vt}{Rf}, Y_r(t) = 0, Y_f(t) = 0, \gamma_1(t) = 0, \gamma_2(t) = 0, \gamma_3(t) = 0, \gamma_4(t) = 0 \end{array} \right]$$

then the constraint equations (7) may be linearized as follows:

```
> lin_constraints := linearize(constraints, qq, t);
```

$$\text{lin\_constraints} := \left[ \left( \frac{\partial}{\partial t} x(t) \right) + Rr \left( \frac{\partial}{\partial t} \theta_1(t) \right), \left( \frac{\partial}{\partial t} x(t) \right) + Rf \left( \frac{\partial}{\partial t} \theta_4(t) \right), -\psi_4(t) + \psi(t) + \delta(t) c(\varepsilon), \phi_4(t) - s(\varepsilon) \delta(t) - \phi(t) \right]$$

It may be observed that, even if the non-linear constraints equations were non-holonomic, the linearization procedure yields to holonomic expressions. The utilization of a CAS tool gives the possibility of exploiting this property by integrating the first and second constraint equations, as follows:

```
> lin_constraints[1] := int(lin_constraints[1],t):
lin_constraints[2] := int(lin_constraints[2],t):
```

Therefore, constraint equations have been simplified as follows:

$$\text{lin\_constraints} = [x(t) + Rr \theta_1(t), x(t) + Rf \theta_4(t), -\psi_4(t) + \psi(t) + \delta(t) c(\epsilon), \phi_4(t) - s(\epsilon) \delta(t) - \phi(t)]$$

In a similar manner, Lagrange's equations (9) and additional equations (10) can also be linearized:

```
> lin_leqns := linearize(leqns, straight_run, t): codegen[cost](lin_leqns);
lin_tire_eqns := subs(change_vars, linearize([equ_Yr,equ_Yf],straight_run,t) );
```

Their evaluation requires:

$$119 \text{ additions} + 186 \text{ functions} + 290 \text{ multiplications} + 4 \text{ divisions} + \text{subscripts}$$

By comparing the number of operations required for evaluating non-linear and linear equations, it may be noted that the complexity of the latter is about 10% of the complexity of the former.

## 4.2 Independent Coordinates Formulation

Even if the linear analysis can be performed in the dependent coordinates formulation, the advantages of this formulation, that has been explained above for a non-linear model, are no longer present in a linear model. On the contrary, the complexity of a model can be minimized by converting it into an independent coordinates formulation. There is more than one method for determining the independent coordinates<sup>18</sup>, for example the coordinates partitioning or the velocity projection matrix method. In presence of only linear constraint equations, the following simple approach may be applied to the motorcycle model. A subset of  $n-c=5$  independent coordinates should be extracted from the set of lagrangian coordinates, then the remaining  $c=4$  coordinates may be calculated from the constraint equations, as follows:

```
> zz := [x(t), y(t), psi(t), delta(t), phi(t)]; #independent coordinates
> yy := [phi4(t),psi4(t),thetal(t),theta4(t)]; #remaining coordinates
change_vars := solve( convert(lin_constraints,set), [phi4(t),psi4(t),thetal(t),
theta4(t)] );
```

$$zz := [x(t), y(t), \psi(t), \delta(t), \phi(t)] \quad (13)$$

$$\text{change\_vars} := \left\{ \theta_1(t) = -\frac{x(t)}{Rr}, \psi_4(t) = \psi(t) + \delta(t) c(\epsilon), \phi_4(t) = s(\epsilon) \delta(t) + \phi(t), \theta_4(t) = -\frac{x(t)}{Rf} \right\}$$

The user has great flexibility in choosing the independent coordinates, even if he erroneously chooses a set of dependent coordinates, this mistake is immediately discovered because the calculation of the remaining coordinates as a function of the former fails.

The next step is the removal of Lagrange's multipliers from Lagrange's equations (10), which may be performed by pre-multiplying them by the matrix  $\mathbf{R}$  defined below<sup>18</sup>:

$$\mathbf{F}_q \mathbf{R} = \mathbf{0} \quad (14)$$

where  $\mathbf{F}_q$  is the Jacobian matrix of the constraint equations. The matrix  $\mathbf{R}$  is also called velocity projection matrix because, in the case of holonomic constraints, it correlates dependent velocities  $\dot{\mathbf{q}}$  to the independent ones  $\dot{\mathbf{z}}$  as follows:

$$\dot{\mathbf{q}} = \mathbf{R} \dot{\mathbf{z}} \quad (15)$$

The matrix  $\mathbf{R}$  is often calculated from SVD decomposition of Jacobian  $\mathbf{F}_q$ , but for linear models matrix  $\mathbf{R}$  is constant and thus the integration of equation (15) makes it possible to also correlate positions as  $\mathbf{q} = \mathbf{R} \mathbf{z}$ . Therefore, matrix  $\mathbf{R}$  can be immediately calculated by expressing the correlation between dependent coordinates  $\mathbf{q}$  (12) and independent ones  $\mathbf{z}$  (13) in matrix form, as follows:

```
> RR := linalg[genmatrix]( subs(change_vars,qq) , zz );
```

$$RR := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & c(\epsilon) & 0 \\ 0 & 0 & 0 & s(\epsilon) & 1 \\ -\frac{1}{Rr} & 0 & 0 & 0 & 0 \\ -\frac{1}{Rf} & 0 & 0 & 0 & 0 \end{bmatrix}$$

After this, the independent Lagrange's equations can be easily computed:

```
> lin_independent_leqns:=evalm(linalg[transpose](RR) &*subs(change_vars,lin_leqns)) :
```

The motorcycle is now described by means of  $n-c=5$  independent Lagrange's equation plus  $c'=2$  auxiliary tire equations, having a corresponding number of unknowns:

```
> all_eqns := lin_independent_leqns union lin_aux_eqns:
all_vars := zz union [Yr(t),Yf(t)];
```

$$all\_vars := [x(t), y(t), \psi(t), \delta(t), \phi(t), Yr(t), Yf(t)]$$

The symbolic approach offers the possibility of further manipulating the model equations. For example, if the absolute components  $\dot{x}, \dot{y}$  of the velocity of the point  $\mathbf{A}$  are replaced by its components  $\dot{x}_1, \dot{y}_1$  expressed in the moving frame  $\mathbf{T}_1$ , the Sharp's equations<sup>19</sup> are re-derived, as follows (the first Lagrange's equation, which describes the motion in the X direction, is trivial and therefore it has been dropped):

```
> VA := make_VECTOR(T1,x1[dot](t),y1[dot](t),0):
VA := linearize( project(VA,ground) , [x1[dot](t)=V,y1[dot](t),psi(t)] ,t):
moving_vars := [ diff(x(t),t)=Xcomp(VA), diff(y(t),t)=Ycomp(VA) ];
> sharp_vars := [y1[dot](t), psi(t), delta(t), phi(t), Yr(t), Yf(t)];
sharp_leqns := subs(global_properties,moving_vars,all_eqns[2..nops(all_vars)]):
```

$$moving\_vars := \left[ \frac{\partial}{\partial t} x(t) = xI_{dot}(t), \frac{\partial}{\partial t} y(t) = \psi(t) V + yI_{dot}(t) \right]$$

$$sharp\_vars := [yI_{dot}(t), \psi(t), \delta(t), \phi(t), Yr(t), Yf(t)]$$

Moreover, the following relations among dependent geometrical parameters give the possibility of performing additional simplifications, which are not essential but give more

compact equations:

```
> global_properties := {
    f = s(epsilon)*k-c(epsilon)*j,      lx = -an+Rf*s(epsilon),
    a = c(epsilon)*k-e+j*s(epsilon),    lz = -1/c(epsilon)*(-s(epsilon)
    *k*c(epsilon)+c(epsilon)^2*Rf+s(epsilon)*ej+c(epsilon)^2*j+s(epsilon)*an)  }:
> sharp_leqns := map(expand,map(simplify,sharp_leqns,trig)):
```

$$f = s(\epsilon) k - c(\epsilon) j \quad a = c(\epsilon) k - e + j s(\epsilon)$$

$$l_z = -\frac{-s(\epsilon) k c(\epsilon) + c(\epsilon)^2 Rf + s(\epsilon) e - j + c(\epsilon)^2 j + s(\epsilon) an}{c(\epsilon)} \quad l_x = -an + Rf s(\epsilon)$$

### 4.3 State Space Formulation

The state space equations and variables can be derived from Lagrange's 2<sup>nd</sup> order differential equations simply by reducing them into a set of 1<sup>st</sup> order equations and by eliminating redundant variables, i.e. by applying the `first_order()` procedure as follows:

```
> (xx,state_eqns) := first_order(sharp_leqns,sharp_vars,t):
```

State variables are:

$$xx = [yI_{dot}(t), \delta(t), \phi(t), Yr(t), Yf(t), \psi_{dot}(t), \delta_{dot}(t), \phi_{dot}(t)] \quad (16)$$

In a symbolic approach, instead of the standard state space formulation it is preferable to use the following implicit formulation, where  $\mathbf{x}$  is the state variables vector and  $\mathbf{u}$  is the inputs vector:

$$\begin{aligned} \mathbf{A}\dot{\mathbf{x}} &= \mathbf{B}\mathbf{x} + \mathbf{C}\mathbf{u} \\ \mathbf{y} &= \mathbf{D}\mathbf{x} + \mathbf{E}\mathbf{u} \end{aligned} \quad (17)$$

Indeed, the formulation above does not require the symbolic inversion of the matrix  $\mathbf{A}$ , which would be a very onerous task.

For the motorcycle model, the only one input is the steering torque  $t$ ; matrices  $\mathbf{A}$ ,  $\mathbf{B}$  and  $\mathbf{C}$  can be easily computed using the `state_space()` command. It can be verified that the following matrices correspond *exactly* to the equations in the Sharp's paper<sup>19</sup>.

```
> uu := [tau(t)]; #driving vector
(AA,BB,CC) := state_space(leqns1,coords1,t,uu):
j*s(epsilon) = a -c(epsilon)*k+e; # further geometrical simplification
BB := map(expand,subs(%,map(simplify,BB)))
```

$$AA = \begin{bmatrix} mf + mr & 0 & 0 & 0 & 0 & mf k & mf e & mf j + mr h \\ mf k & 0 & 0 & 0 & 0 & mf k^2 + Irz + Ifx - Ifx c(\epsilon)^2 + Ifz c(\epsilon)^2 & mf e k + Ifz c(\epsilon) & -Ifx c(\epsilon) s(\epsilon) + mf j k + Ifz c(\epsilon) s(\epsilon) + Crxz \\ mf e & 0 & 0 & 0 & 0 & mf e k + Ifz c(\epsilon) & Ifz + mf e^2 & Ifz s(\epsilon) + mf e j \\ mf j + mr h & 0 & 0 & 0 & 0 & -Ifx c(\epsilon) s(\epsilon) + mf j k + Ifz c(\epsilon) s(\epsilon) + Crxz & Ifz s(\epsilon) + mf e j & Irx + mr h^2 - Ifz c(\epsilon)^2 + Ifx c(\epsilon)^2 + Ifz + mf j^2 \\ 0 & 0 & 0 & \frac{\sigma}{V} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\sigma}{V} & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$BB = \begin{bmatrix} 0, & 0, & 0, & 1, & 1, & -mfV - mrV, & 0, & 0 \\ 0, & 0, & 0, & -b, & -\frac{an}{c(\epsilon)} + \frac{a}{c(\epsilon)}, & -mfVk, & \frac{ifyVs(\epsilon)}{Rf}, & \frac{iryV}{Rr} + \frac{ifyV}{Rf} \\ 0, & -Zfs(\epsilon)an + mfe g s(\epsilon), & -Zfan + mfe g, & 0, & -an, & -mfeV - \frac{ifyVs(\epsilon)}{Rf}, & -C_{\delta}, & \frac{c(\epsilon)ifyV}{Rf} \\ 0, & -Zfan + mfe g, & mfgj + mrhg, & 0, & 0, & -mfVj - \frac{iryV}{Rr} - \frac{ifyV}{Rf} - mrhV, & -\frac{c(\epsilon)ifyV}{Rf}, & 0 \\ -\frac{Cr1}{V}, & 0, & Cr2, & -1, & 0, & \frac{Cr1b}{V}, & 0, & 0 \\ -\frac{Cf1}{V}, & c(\epsilon)Cf1 + Cf2s(\epsilon), & Cf2, & 0, & -1, & \frac{Cf1an}{c(\epsilon)V} - \frac{Cf1a}{c(\epsilon)V}, & \frac{Cf1an}{V}, & 0 \\ 0, & 0, & 0, & 0, & 0, & 0, & -1, & 0 \\ 0, & 0, & 0, & 0, & 0, & 0, & 0, & -1 \end{bmatrix}, \quad CC = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

#### 4.4 Numerical Results

The utilization of a CAS offers the further possibility of doing numerical calculation. Despite this might not be an efficient way of making simulations, this capability does not require any additional handling of the model (such as C or Fortan code generation and compilation) and hence may be very useful for preliminary calculation. As an example, eigenvalues are calculated using the motorcycle's characteristics reported in the Sharp's paper<sup>19</sup> for a velocity  $V=20$  m/s, furthermore the complex frequency response functions (FRFs) between the steering torque and state variables (16) are calculated at a frequency  $f=2$  Hz (each row of the FRF's vector corresponds to a state variable). Moreover, figure 2 represents the Nyquist plot for velocity from 1 to 40 m/s and figure 3 represents the FRFs of steering angle  $\mathbf{t}$  and roll angle  $\mathbf{j}$  for a velocity of 20 m/s.

```
> Sharp71data := {...}: #motorcycle's characteristics as in the Sharp's paper
Sharp71data := {Zf=-1005.3, h=.6157, Irx=31.18, Irz=21.07, Crxz=1.7355, iry=1.0508, Ifz=.4420,
ify=.7186, mr=217.45, b=.4798, e=.0244, f=.0283, Ifx=1.2338, an=.1158, ε=.4715, l=.93468,
a=.9485376, Cδ=6.77, mf=30.65, rtf=0, Rf=.3048, Cf1=11174, Cf2=938.6, σ=.2438, rtr=0,
Rr=.3048, Cr1=15831, Cr2=1325.6, g=9.81, j=.4167186306, k=.8796329043}
> # convert matrices to numerical form for improving calculation speed
AAN := convert(evalf(subs(Sharp71data,evalm(AA))),Matrix):
BBN := convert(evalf(subs(Sharp71data,evalm(BB))),Matrix):
# Eigenvalues
eivals := LinearAlgebra[Eigenvalues](subs(V=20,BBN),subs(V=20,AAN)):
# Frequency Responce Functions
GG := evalf(subs(V=20,freq=2,I*2*Pi*freq*AAN-BBN)):
FRFs := evalm(linalg[inverse](GG) &* DD):
```

$$eivals = \begin{bmatrix} -79.5972034800000046 + 0. I \\ -59.9518662699999964 + 0. I \\ -5.82560971300000041 + 54.1820526200000003 I \\ -5.82560971300000041 - 54.1820526200000003 I \\ -20.4335085999999997 + 0. I \\ -4.04230133100000000 + 15.8013662700000008 I \\ -4.04230132999999991 - 15.8013662700000008 I \\ .0926615959300000070 + 0. I \end{bmatrix}, \quad FRFs = \begin{bmatrix} y^l_{dor}(t) \\ \delta(t) \\ \phi(t) \\ Yr(t) \\ Yf(t) \\ \Psi_{dor}(t) \\ \delta_{dor}(t) \\ \phi_{dor}(t) \end{bmatrix}, \quad FRFs = \begin{bmatrix} -.005962934704 + .0009441006142 I \\ .0008071948120 + .001106083910 I \\ .001878696559 + .0008710549705 I \\ 12.72890345 + 1.739597837 I \\ 6.415211800 + 6.932925052 I \\ .01382901938 + .008641966714 I \\ -.01389946035 + .01014350917 I \\ -.01094599959 + .02360839724 I \end{bmatrix}$$

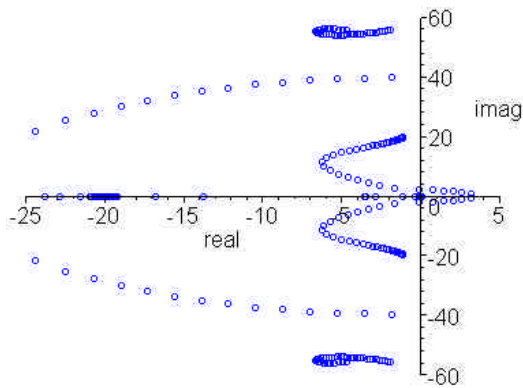


Figure 2 – Nyquist Plot of the motorcycle  
(speed from 1 up to 40 m/s)

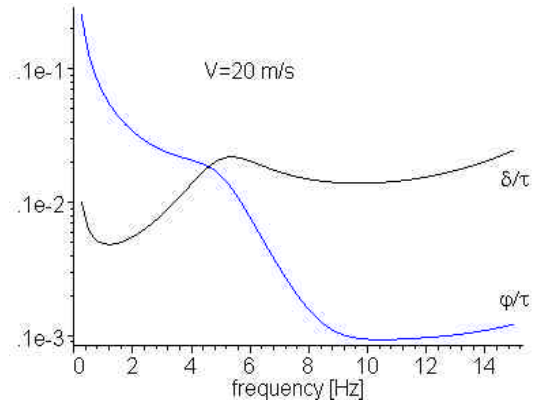


Figure 3 – Frequency Response Functions  
of the steering and roll angles

## 5 CONCLUSIONS

Some procedures for building mathematical models of multibody systems using Maple have been described. A first set of procedures is dedicated to the description of the system: procedures make it possible to easily define objects (point, vectors, bodies forces and torques), to manipulate them and describe relationships between them. The user is free to select the most suitable coordinate system, and it is possible to describe both open and closed kinematic chains, both holonomic and non holonomic constraints. Moreover, it is possible to introduce new special objects, such as tires, active actuators and more besides. A second set of procedures is devoted to the derivation of the equations of motion and is based on the Lagrange approach. These procedures make it possible to automatically derive the system's equations of motion, however it is possible to maintain a step by step control of these operations. The model equations consist of a set of 2<sup>nd</sup> order differential algebraic equations, which may be further manipulated and converted into an ordinary set of differential equations. A further set of commands is dedicated to the linearization equations and to their conversion into state space formulation. All the above procedures are designed to be used in conjunction with the built-in procedures of the CAS system, which may be profitably used for simplifying expressions, combining equations, calculating eigenvalues, integrating equations and in particular for generating C or Fortran code, which constitute the starting point of building optimized simulations code.

As an example, a mathematic model of a motorcycle has been derived. The model has only five degrees of freedom, but it has complex kinematic chains. The non-linear equations of motion have been derived using a dependent coordinates formulation. Next, they have been linearized and converted into a independent coordinates formulation, re-obtaining a model of the straight running of the motorcycle, which has been well-documented. This example demonstrates the power of the symbolic approach in multibody modeling, which make it possible to obtain optimized models because of the possibility of choosing the most suitable coordinates set, the essential description of the system and the utilization of symbolic simplifications of equations.

## 6 ACKNOWLEDGMENT

Methods for symbolic modeling of multibody systems described here have been partially developed with the financial support from the “Provincia Autonoma di Trento” (PAT) in the framework of project MEPROME (a research project for the years 2001-2004 aiming at the development of tools for modeling simulation and design of mechatronic systems).

## 7 APPENDIX: LIST OF PROCEDURES FOR MULTIBODY MODELING

### 7.1 Building objects

<code>translate(x,y,z)</code>	create a 4x4 frame matrix as a <code>(x,y,z)</code> translation
<code>rotate ('axis',angle)</code>	create a 4x4 frame matrix as a rotation <code>angle</code> around the 'X', 'Y' or 'Z' axis
<code>inv_frame(frame)</code>	create a 4x4 frame matrix as the inverse of the given <code>frame</code>
<code>make_POINT(frame, x,y,z)</code>	create a point having <code>x,y,z</code> coordinates with respect to the given <code>frame</code>
<code>origin(frame)</code>	create a point as the origin of the given <code>frame</code>
<code>make_VECTOR(frame, ux,uy,uz)</code>	create a vector having <code>ux,uy,uz</code> components with respect to the given <code>frame</code>
<code>make_BODY(CoM,mass, Ix,Iy,Iz,Cyz,Cxz,Cxy)</code>	create a body having center of mass <code>CoM</code> , whit <code>mass</code> and inertia tensor defined in the same frame of <code>CoM</code>
<code>make_FORCE(frame, ux,uy,uz, acting_point, acting_body, reacting_body)</code>	create a force having <code>ux,uy,uz</code> components with respect to the given <code>frame</code> , applied on the given <code>acting_point</code> and acting between <code>acting_body</code> and <code>reacting_body</code>
<code>make_TORQUE(frame, ux,uy,uz, acting_body, reacting_body)</code>	create a torque, similarly to above

### 7.2 Kinematics Procedure

<code>project(point ,new_frame)</code>	project the given <code>point</code> (or vector, or force, or torque), which is defined respect its own frame, into the <code>new_frame</code>
<code>velocity(point, 'absolute')</code> <code>velocity(point, 'relative')</code> <code>velocity(point, 'eulerian')</code>	compute the velocity of a point (or a vector)
<code>angular_velocity(frame)</code>	compute the angular velocity of reference frame
<code>dot_prod(u,v)</code> <code>cross_prod(u,v)</code>	dot and cross product between vectors
<code>Comps({point,vector,force,torque})</code> <code>Xcomp()</code> , <code>Ycomp()</code> , <code>Zcomp()</code>	extract the components of the give <code>point</code> (or vector, or force, or torque)

### 7.3 Energetic and Dynamics Procedures

<code>kinetic_energy(body)</code> <code>gravity_energy(body)</code>	compute Kinetic and Gravitational Energy of a <code>body</code>
<code>generalized_force(force,q)</code>	compute the Generalized Force of the given applied <code>force</code> (or torque) with respect the coordinate <code>q</code>
<code>lagrange(L,q,t)</code>	derive the Lagrange's Equation respect the coordinate <code>q</code>
<code>(vars,eqns) := lagrange_equations</code> <code>(bodies,forces,constraints,coords,t,lambda)</code>	derive Lagrange's Equations for the given multibody system
<code>down_order(eqns,vars,t)</code> <code>(fo_eqns,fo_vars):=first_order(eqns,vars,t)</code>	convert a set of 2 <sup>nd</sup> order differential equations into a set of 1 <sup>st</sup> order equations

### 7.4 Linear Analysis

<code>linearize(expression, expansion_point,t)</code>	linearize the given <code>expression</code> basing on the given <code>expansion_point</code> , considering also the derivative of variables respect the independent coordinate <code>t</code>
<code>Q := quadratic_form(F,q)</code> <code>(M,C,K,F) := dynamic_form(legns,lcoords,t)</code>	extract the <b>Q</b> matrix from expression $F = \mathbf{q}^T \mathbf{Q} \mathbf{q}$ convert equations to the dynamic matrix form $\mathbf{M}\ddot{\mathbf{q}} + \mathbf{C}\dot{\mathbf{q}} + \mathbf{K}\mathbf{q} = \mathbf{F}$
<code>(A,B,C) := state_space(fo_eqs,fo_vars,t,uu)</code>	convert equations to the state space matrix form $\mathbf{A}\dot{\mathbf{x}} = \mathbf{B}\mathbf{x} + \mathbf{C}\mathbf{u}$

## 8 REFERENCES

- [1] J. Lih, I. Haque, *Symbolic Closed-Form Modeling and Linearization of Multibody Systems Subject to Control*, Journal of Mechanisms, Transmissions, and Automation in Design, 113-2, pp. 124-132 (1991).
- [2] M.W. Sayers, *Symbolic Computer Language for Multibody Systems*, Journal of Guidance, Control, and Dynamics, 14-6, pp. 1153-1163 (1991).
- [3] P.E. Nikravesh, J.A.C. Ambrosio, *Automatic Construction of Equations of Motion for Rigid-Flexible Multibody Systems*, Winter Annual Meeting of the American Society of Mechanical Engineers, Nov. 8-13 1992, Anaheim, CA, USA, Publ. by ASME, New York, NY, USA, ISSN 0160-8835, pp. 125-132 (1992).
- [4] N. Sreenath, *Hybrid Computation Environment For Multibody Simulation*, *Mathematics and Computers in Simulation*, 34-2, pp. 121-140 (1992).
- [5] J. Lih, *Computer-Oriented Closed-Form Algorithm for Constrained Multibody Dynamics for Robotics Applications*, Mechanism and Machine Theory, 29-3, pp. 357-371 (1994).
- [6] K. Cui, I. Haque, M. Thirumalai, *On Configurations of Symbolic Equations, of Motion for Rigid Multibody Systems*, Mechanism and Machine Theory, 30-8, pp. 1149-1170, (1995).
- [7] J. McPhee, M.G. Ishac, G.C. Andrews, *Wittenburg's Formulation of Multibody Dynamics Equations from a Graph-Theoretic Perspective*, Mechanism and Machine Theory, 31-2, pp. 201-213 (1996).

- [8] J.M. Pagalday, A. Avello, *Optimization of Multibody Dynamics Using Object Oriented Programming and a Mixed Numerical-Symbolic Penalty Formulation*, Mechanism and Machine Theory, 32-2, pp. 161-174 (1997).
- [9] K. Cui, I. Haque, *Symbolic Equations of Motion for Hybrid Multibody Systems Using a Matrix-Vector Formulation*, Mechanism and Machine Theory, 32-6, pp. 743-763 (1997).
- [10] G.L. Blankenship, H.G. Kwatny, C. La Vigna, V. Polyakov, Integrated Modeling and Design of Nonlinear Control Systems, 1997 American Control Conference, Jun. 4-6 1997, Albuquerque, NM, USA, Publ. by IEEE, Piscataway, NJ, USA, pp. 1395-1399.
- [11] A. Kecskemethy, T. Krupp, M. Hiller, Symbolic Processing of Multiloop Mechanism Dynamics Using Closed-Form Kinematics Solutions, Multibody System Dynamics, 1, pp. 23-45 (1997).
- [12] J. McPhee, Automatic Generation of Motion Equations for Planar Mechanical Systems Using the New Set of "Branch Coordinates", Mechanism and Machine Theory, 33-6, pp. 805-823, (1998).
- [13] F.C. Park, J. Choi, S.R. Ploen, Symbolic Formulation of Closed Chain Dynamics in Independent Coordinates, Mechanism and Machine Theory, 34, pp.731-751 (1999).
- [14] P. Shi, J. McPhee, Dynamics of Flexible Multibody Systems Using Virtual Work and Linear Graph Theory, Multibody System Dynamics, 4-4, pp. 355-381 (2000).
- [15] P. Fisette, T. Postiau, L. Sass, J.C. Samin, Fully Symbolic Generation of Complex Multibody Models, Mechanics of Structures and Machines, 30-1, pp. 31-82 (2002).
- [16] M. Da Lio, D. Bortoluzzi: Symbolic Derivation of Open-Loop Dynamic Models of Multibody Mechatronic Systems for Control Purpose. A Case Study: the Lisa Technology Package, 5th ESA International Conference on Spacecraft Guidance, Navigation and Control Systems, 22-25 October 2002, Frascati, Rome, Italy, SP-516
- [17] V. Cossalter and R. Lot: A Motorcycle Multi-Body Model for Real Time Simulations Based on the Natural Coordinates Approach, Vehicle System Dynamics Vol 37, n.6, pp. 423-448, (2002)
- [18] J.G. de Jalon and E. Bayo, Kinematic and Dynamic Simulation of Multibody Systems, Springer Verlag (1994)
- [19] R.S. Sharp: The Stability and Control of Motorcycles, Journal of Mechanical Engineering Science, 13 (1971).
- [20] H. Sush and C.W. Radcliffe, Kinematics and Mechanism Design, J. Wiley & sons, New York, (1978).
- [21] E. Pennestrì, *Dinamica Tecnica e Computazionale*, Vol.2, Editrice Ambrosiana (2001).
- [22] D. T. Greenwood, *Classical Mechanics*, Prentice-Hall: Englewood Cliffs, (1977).
- [23] L. Meirovitch, *Methods of Analytical Dynamics*, McGraw-Hill: New York, (1970).
- [24] J. S. Torok: *Analytical Mechanics* Wiley (1999)
- [25] E. Hairer, G. Wanner : Solving Ordinary and Differential Equations II, Stiff and Differential- Algebraic Problems, Springer Verlag (1991)