Open source model of the AMRU5 hexapod robot

O. Verlinden*, **J.-C. Habumuremyi**** and **G. Kouroussis***
*Department of Theoretical Mechanics, Dynamics and Vibrations
Faculté polytechnique de Mons, Bd Dolez 31, B-7000 Mons, Belgium
**Department of Theoretical and Applied Mechanical Systems
Royal Military Academy
Avenue de la Renaissance 30, B-1000 Bruxelles, Belgium
e-mail: Olivier.Verlinden@fpms.ac.be
web page: http://www.mecara.fpms.ac.be/EasyDyn

ABSTRACT

The walking robot AMRU5 is the result of collaborative work of RMA (Royal Military Academy), ULB (Université libre de Bruxelles) and VUB (Vrije Universiteit Brussel). It is an hexapod robot whose each leg has 3 degrees of freedom and corresponds to a pantograph mechanism which can rotate with respect to the central body about a vertical axis. A central controller manages the global motion of the robot, and delegates the detail of the motion control to the 6 controllers installed on each leg. The latter have been set up according to a neuro-fuzzy approach [1].

In collaboration with FPMs (Faculté polytechnique de Mons), a dynamic model has been initiated, on the basis of `EasyDyn`. Basically developed for teaching, `EasyDyn` is a framework mixing symbolic and numerical tools in order to simulate the dynamic behaviour of a mechanical system. It is based on tools available for free (for non-commercial purposes) and can be downloaded freely, along with well documented examples and problems for students. The basic idea of `EasyDyn` is to provide automatic tools for the mathematical part (time derivation and vector calculus) without hiding the mechanical part. It is important to note that `easyDyn` is not a program but a library allowing to easily build a program dedicated to the simulation of a particular system. The approach has the advantage to very flexible as it allows to add any peculiarity of the system, eventually not available in a commercial software. In particular, walking robots involve a unilateral contact with the ground and eventually particular controllers.

Practically, the simulation of a mechanical system with `EasyDyn` begins with the `CaGEM` utility, which is a script running under the symbolic engine `MuPAD`. From a basic description of the system (number of bodies, number of degrees of freedom, inertia data and expression of position matrices in terms of the chosen configuration parameters), `CAGeM` generates a C++ application. The latter relies on the numerical part of `EasyDyn`, an object-oriented C++ library, providing routines to build and integrate the equations of motion of a mechanical system. The equations of motion are built according to the d'Alembert's principle, from the kinematics of the body and the expression of the efforts applied on each body. The interest of the symbolic tool is that the velocities and accelerations are built automatically by symbolic derivation of the positions. The C++ code can be compiled as is but is the most often completed, namely for the description of the efforts. For that purpose, several routines are provided for classical elements (springs, dampers,...) as well as vector classes allowing the usage of vector expressions in C++. Eventually, supplementary differential equations, related for example to the dynamics of actuators or sensors can also be added to the ones related to the dynamics of the mechanical system. A visualization of the system can also be easily realized (figure 9.

The present model of AMRU5 involves 50 bodies and 18 degrees of freedom. The contact between the legs and the ground is modelled by a unilateral elastic contact with friction. The dynamical

behaviour of the DC motors is included as either a first or second-order model (with inductance). The numerical controllers of the legs are implemented in a realistic way as discrete-time devices acting on the system at regular intervals. So far, only a simple PID controller has been tested. However, the neuro-fuzzy controller should be implemented easily as the actual C code used to program the microcontroller can be retrieved as is in the simulation program. Some important phenomena like friction in the transmissions still have to be considered to get a completely representative model. The work of the central microcontroller can also be envisaged although it is less dependent on the dynamic response of the robot.

Although it was developed for teaching, the `EasyDyn` framework offers a good foundation for collaborative work or unsual mechanical systems like walking robots.

1 INTRODUCTION

That's the reason why the authors developed a framework called `EasyDyn`, which allows the student to perform simulations quite easily but only if he has a sufficient knowledge of the discipline. We kept in mind the following considerations during the development

- the tool must be open source [5], in the hope that it can be useful not only to our students but also to anybody interested in the subject;
- existing libraries or programs will be used for particular problems as far as they are freely available, at least for non commercial usage;
- it shouldn't be limited to simplistic 2D systems;
- it should offer the possibility to be extended to mechanical systems involving particular components (controllers, hydraulic or electric actuators, ...);
- it should be portable and run under Windows and Unix;
- it should be composed of several independent elements with a specific task; more especially, the computational part won't be mixed with any graphical result.

2 Description of AMRU5

The walking robot AMRU5[9] (figure 1) has been built from a collaboration between different belgian universities: RMA (Royal Miltitary Academy), ULB (Université libre de Bruxelles) and VUB (Vrije Universiteit van Brussel). It is an hexapod robot (figure 1), whose each leg owns 3 degrees of freedom. The 18 motions are actuated by DC electric motors.

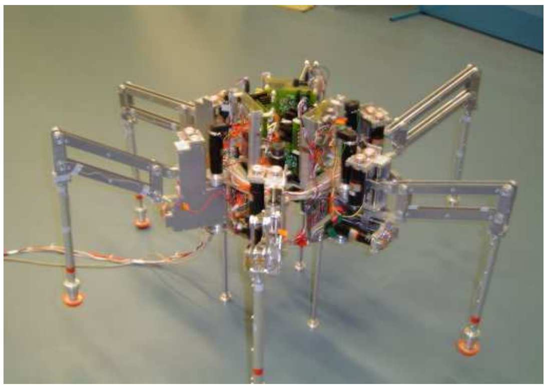


Figure 1: AMRU5 robot

To assure a gravitationally decoupled actuation, known as an important property for energy savings in walking robots, as well as an easy control of the foot position, each leg consists of a pantograph mechanism (figure 2). In the plane of the pantograph, the horizontal (vertical) coordinate of the foot (point F) is controlled independently by the horizontal (vertical) motion of the driving point C (A). The third degree of freedom of the leg corresponds to a rotation of the pantograph about a vertical axis with respect to the central body.

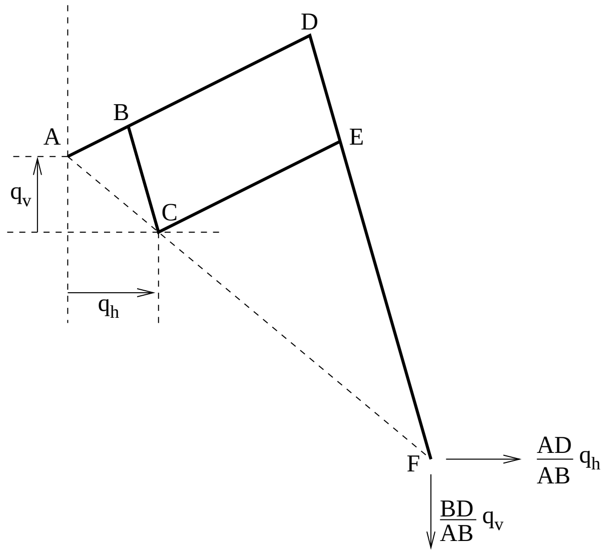


Figure 2: Kinematics of the pantograph mechanism

The mechanical structure of the leg, as well as the active and passive joints, are illustrated in figure 3. The pantograph mechanism is attached to a support part, which rotates with respect to the central body. This rotation is driven by a first DC motor through a gearbox. Two other DC motors lead, through a gearbox and a nut-screw system, the translational motion of the driving points.

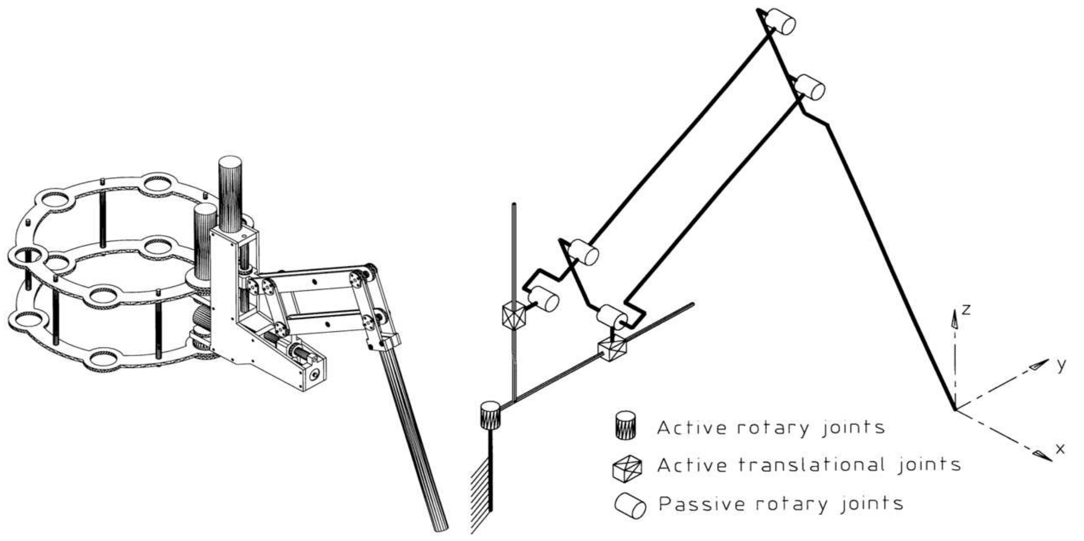


Figure 3: Mechanical structure of a leg

A central controller manages the global motion of the robot, and delegates the detail of the motion control to the 6 controllers installed on each leg. The latter have been set up according to a neuro-fuzzy approach [1]. The central controller is not considered in this work.

Disposing of a good dynamic model of such a walking robot is interesting on several points of view: the model can not only be used to set up the controller but also to test its robustness with respect to various nonlinearities that are difficult to take into account when developping the controller.

3 EasyDyn

3.1 Why EasyDyn ?

`EasyDyn` is a framework dedicated to the simulation of dynamic systems, and especially multibody systems, which has been developed for teaching purposes. It is not a simulation program but a C++ library, combined with a symbolic utility, freely available from the internet [1]. `EasyDyn` comes along with documentation, examples and a series of problems. Under `EasyDyn`, the mechanical system is not defined in a data file but as a C++ program describing the kinematics and the applied forces. The equations of motion can then be built and integrated numerically with the help of particular routines of the library. To make the task easier and the code particularly readable, the object oriented features of the C++ language have been exploited to redefine a comprehensive vector algebra built from classes like vectors, rotation or inertia tensors and homogeneous transformation matrices, combined with the overloading of the usual operators. In such a way, vector expressions can be written as is in the C++ program, like a matrix expression under Matlab-alike programs. Last but not least, `EasyDyn` provides a utility which generates symbolically a complete C++ code, where the kinematics (velocities and accelerations) has been derived symbolically from only the position information. This last module is actually a script running under `MuPAD`, developed at the university of Paderborn in Germany. Although available for free for non commercial usage, `MuPAD` offers features comparable to its commercial counterparts.

4 THE C++ Library of `EasyDyn`

The C++ library of `EasyDyn` library provides 4 modules with a specific task

- the `vec` module defines classes related to vector calculus (vectors, rotation tensors, inertia tensors, and homogeneous transformation matrices), and the related algebra so that vector expressions can be written in natural form in a C++ program; this module is completely independent;
- the `sim` module provides routines for the integration of second-order differential equations written in residual form; this module can be used on its own;
- the `mbs` module is a frontend to `sim` which automatically builds the differential equations of motion of a multibody system from the kinematics and the applied forces; the `mbs` module also features routines for the application of usual forces; this module relies on the `vec` and `sim` modules.
- the `visu` module provides routines to build 3D scenes, composed of moving objects, that can be viewed by external programs; the `visu` largely uses the `vec` module.

4.1 The `vec` module

The need for vector algebra is evident in multibody systems. Forces, position vectors, velocities, accelerations are all vectors. In practice, vector calculus needs the projection in an arbitrarily chosen frame (coordinate system). A frame i is composed of 3 unit vectors $\boldsymbol{x}_i$, $\boldsymbol{y}_i$ and $\boldsymbol{z}_i$ perpendicular to each other and positively oriented. Let us denote $\{\boldsymbol{a}\}_i$ the 3x1 vector gathering

[1]http://www.mecara.fpms.ac.be

the 3 components of vector $\boldsymbol{a}$ in frame i

$$\{\boldsymbol{a}\}_i = \begin{pmatrix} a_{x_i} \\ a_{y_i} \\ a_{z_i} \end{pmatrix} \leftrightarrow \boldsymbol{a} = a_{x_i} \cdot \boldsymbol{x}_i + a_{y_i} \cdot \boldsymbol{y}_i + a_{z_i} \cdot \boldsymbol{z}_i \tag{1}$$

The transfer between different coordinate systems i and j is easily written in the following matrix form

$$\{\boldsymbol{a}\}_i = [\boldsymbol{R}_{i,j}] \cdot \{\boldsymbol{a}\}_j \quad \text{with} \quad [\boldsymbol{R}_{i,j}] = \left[\{\boldsymbol{x}_j\}_i \ \{\boldsymbol{y}_j\}_i \ \{\boldsymbol{z}_j\}_i\right] \tag{2}$$

where $\boldsymbol{R}_{i,j}$ is the rotation tensor describing the orientation of frame j with respect to frame i.

Frames are also used to localize points in space. We will denote $\boldsymbol{r}_{P/i}$ the coordinate vector of point P with respect to frame i, that's to say the vector running from frame i to point P (figure 4). Generally, a global reference exists and is referenced as frame 0. The coordinate vector of a point P with respect to the global reference frame, that's to say $\boldsymbol{r}_{P/0}$ can be denoted for the sake of simplicity by $\boldsymbol{e}_P$.

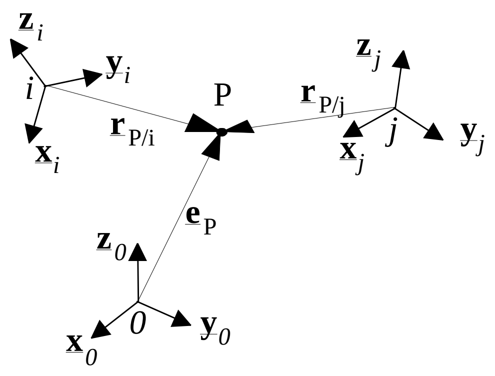


Figure 4: Points, frames and coordinate vectors

In the same way, we need to be able to localize the situation of a body with respect to a reference frame. A frame j is then attached to the body and its situation with respect to the reference frame i will be expressed from the 4x4 homogeneous transformation matrix $\boldsymbol{T}_{i,j}$ defined as

$$\boldsymbol{T}_{i,j} = \begin{pmatrix} \boldsymbol{R}_{i,j} & \{\boldsymbol{r}_{j/i}\}_i \\ 0\ 0\ 0 & 1 \end{pmatrix} \tag{3}$$

The homogeneous transformation matrices also have the advantage to enjoy the following two properties

$$\begin{pmatrix} \{\boldsymbol{r}_{P/i}\}_i \\ 1 \end{pmatrix} = \boldsymbol{T}_{i,j} \cdot \begin{pmatrix} \{\boldsymbol{r}_{P/j}\}_j \\ 1 \end{pmatrix} \qquad \boldsymbol{T}_{i,k} = \boldsymbol{T}_{i,j} \cdot \boldsymbol{T}_{j,k} \tag{4}$$

In particular, the last relationship allows to develop the motion along a kinematic chain as the product of successive elementary matrices.

Taking these considerations into account, the `vec` module offers the following classes

- vectors (class `vec`), consisting of 3 public variables, `x`, `y` and `z` representing the coordinates of the vector in an orthonormal dextrosum coordinate system;
- rotation tensors (class `trot`), consisting of the 9 coefficients of the matrix (public variables `r11`, `r12`, ..., `r33`);

- inertia tensors (class `tiner`), equivalent to the 3x3 symmetric matrix built from the 3 moments of inertia `Ixx` `Iyy`, `Izz` and the 3 inertia products `Ixy`, `Ixz` and `Izz`;
- homogeneous transformation matrices (class `mth`) built from a vector and a rotation tensor.

The classical operators are overloaded and allow namely vector addition, scalar and vector products, multiplication by a rotation tensor or a homogeneous transformation matrix. Several methods also exist to assign the objects. In particular, a transformation matrix can be initialized from the multiplication of elementary matrices corresponding to displacements or rotations about a given axis. The following piece of code is given for the purpose of illustration

```
%#include<EasyDyn/vec.h>
vec a(1,2,3),b,c;
b.put(3,2,1);
c=a+b; c=a-b; c=a^b;
c=(2*a+(b^vcoord(3,4,5)))/3;
cout << c << endl;
double d=a*b;
trot R=Rrotz(0.2);
c=R*a; c=R.inv()*a;
mth T=Troty(0.1)*Tdisp(1,0,0);
vec r(0,0.5,0),e=T*r;
```

The vector algebra defined in the `vec` module assumes that the vectors are projected in the right coordinate system. It is the responsability of the user to verify this condition.

4.2 The `sim` module

The `sim` module consists of routines to integrate second-order differential equations of the form

$$\boldsymbol{f}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, t) = 0 \tag{5}$$

expressed in terms of time t, the state variables $\boldsymbol{q}$ and their first and second time derivatives $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$.

The only integration method available so far is based on the implicit Newmark formulas, classically used in finite elements and characterized by a an accuracy order of 2. The integration process implements an adpative time step and can deal with stiff equations. More details about the implementation can be found in [?], as well as the example of a hydraulic system showing that the tool is not limited to multibody systems.

The utilization of the module, illustrated in figure 5, generally consists in a call to the routine `NewmarkIntegration` which will perform the integration of the differential equations as far as the residuals $\boldsymbol{f}$ are described in the routine `ComputeResidual`. At regular intervals, the integration process calls the routine `SaveData` provided by the user. Most of the time, it will come down to a call to the `SaveStateVariables` but the user is free to save some more data or to perform any other action expected to be done at regular intervals (a digital controller for example). The results are saved in text form and can be plotted by a tool like `gnuplot`.

Besides the integration, `mbs` provides a routine to determine the state of static equilibrium and a routine which determines the poles of the system after linearization of the differential equations about the current configuration.

The numerical methods implemented in `mbs` are largely based on matrices. All matrix operations rely on the Gnu Scientific Library (GSL), except for the eigenvalue problem which uses a routine from LAPACK.

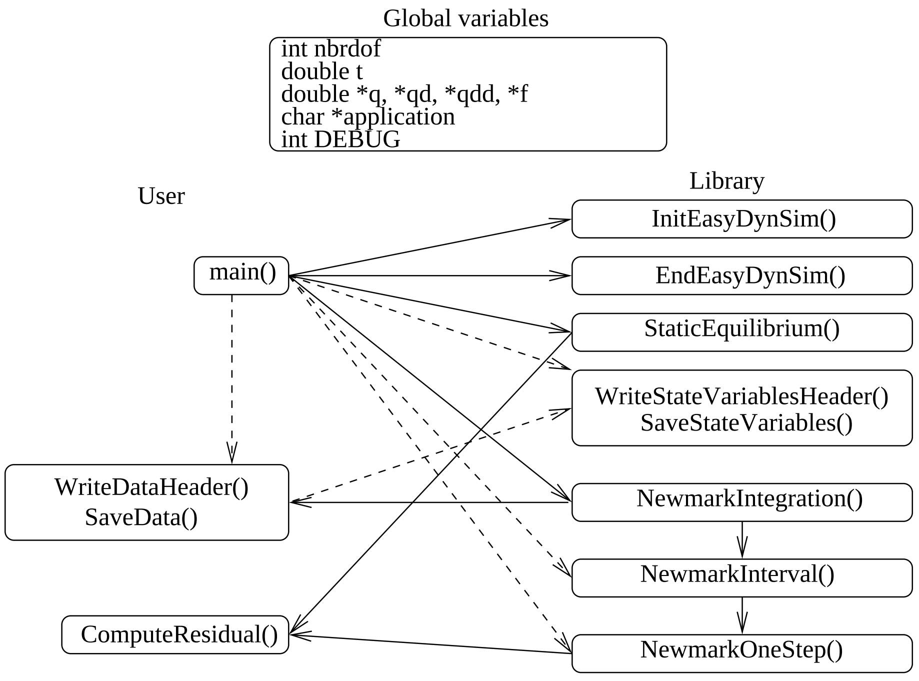


Figure 5: Using the `sim` module of `EasyDyn`

4.3 The `mbs` module

The `mbs` module is a frontend to `sim` which automatically builds the residuals of the motion equations of a multibody system as far as the user formulates, for each body, the kinematics (position, velocities and accelerations) of the center of gravity and the resultant wrench of the applied efforts. The kinematics is expressed in terms of the chosen configuration parameters $\boldsymbol{q}$ and their first and second time derivatives $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$. This implies that only generalized coordinates can be used, whose number corresponds exactly to the number of degrees of freedom. The module `mbs` cannot deal with constraint equations.

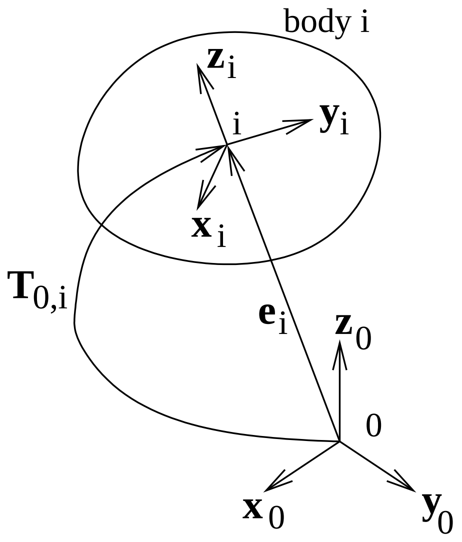


Figure 6: Reference frame of a body

The differential equations governing the motion are constructed by application of the d'Alembert's principle [7], which has the advantage to eliminate automatically all the joint forces. If the system comprises n_B bodies and n_{cp} configuration parameters (degrees of freedom), the n_{cp}

equations can be built from

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

with

- m_i and $\boldsymbol{\Phi}_{G_i}$ the mass and the central inertia tensor of body i;
- $\boldsymbol{R}_i$ and $\boldsymbol{M}_{Gi}$ the resultant force and moment, at the center of gravity G_i, of all applied efforts exerted on body i;
- $\boldsymbol{a}_i$ the acceleration of the center of gravity of body i;
- $\boldsymbol{d}^{i,j}$ the partial contributions of $\dot{\boldsymbol{q}}_j$ in the velocity $\boldsymbol{v}_i$ of the center of gravity of body i

$$\boldsymbol{v}_i = \sum_{j=1}^{n_{cp}} \boldsymbol{d}^{i,j} \cdot \dot{\boldsymbol{q}}_j \tag{7}$$

- $\boldsymbol{\theta}^{i,j}$ the partial contributions of $\dot{\boldsymbol{q}}_j$ in the rotational velocity $\boldsymbol{\omega}_i$ of body i such that

$$\boldsymbol{\omega}_i = \sum_{j=1}^{n_{cp}} \boldsymbol{\theta}^{i,j} \cdot \dot{\boldsymbol{q}}_j \tag{8}$$

The n_{cp} resulting equations of motion have the well known following form

$$\boldsymbol{M}(\boldsymbol{q}) \cdot \ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) = 0 \tag{9}$$

with

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

$$\boldsymbol{M}_{jk} = \textstyle\sum_{i=1}^{n_B} \left[m_i \boldsymbol{d}^{i,j} \cdot \boldsymbol{d}^{i,k} + \boldsymbol{\theta}^{i,j} \cdot (\boldsymbol{\Phi}_{G_i} \cdot \boldsymbol{\theta}^{i,k})\right] \tag{10}$$

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

Practically, the user provides three principal routines

- `SetInertiaData` expected to initialize the inertia properties of all the bodies of the system;
- `ComputeMotion` which describes, in terms of the configuration parameters and their time derivatives, the kinematics of each body, that's to say, the position matrix $\boldsymbol{T}_{0,i}$, the translation and rotation velocities $\boldsymbol{v}_i$ and $\boldsymbol{\omega}_i$ and the corresponding accelerations $\boldsymbol{a}_i$ and $\dot{\boldsymbol{\omega}}_i$.
- `AddAppliedEfforts` which builds the resultant force $\boldsymbol{R}_i$ and the resultant moment $\boldsymbol{M}_{Gi}$, of the applied efforts exerted on each body.

Once these routines are available, the residuals of the equations of motion can be built by a call to `ComputeResidualmbs`. By calling the latter from `ComputeResidual`, the integration routines of the `sim` module can be called directly. The separation of the two routines was made on purpose. The equations of the multibody part are built by `ComputeResidualmbs` and supplementary differential equations can be defined in `ComputeResidual`, to describe the electrical or hydraulic parts of the complete system.

Let us remark that the user doesn't have to provide the expression of the partial contributions. It is indeed clear that the partial contribution $\boldsymbol{d}_{i,j}$ ($\boldsymbol{\theta}_{i,j}$) is equal to the variation of the translation (rotation) velocity of body i for a unit variation of $\dot{\boldsymbol{q}}_j$. Let us note that in the development version of EasyDyn, the user can provide a direct expression for the partial contributions which allows to considerably accelerate the computational process, especially with treelike systems.

The mbs module also provides routines to apply efforts related to classical force elements: sping, damper, gravity, contact point plane with friction and also a tire element. The latter is built on the model of the university of Arizona [?, ?] and is limited so far to a flat ground.

4.4 The visu module

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

Figures ?? and ?? are examples of scenes built with the help of the visu module.

4.5 Example

Let us consider the double pendulum illustrated in figure 7. It is composed of 2 bodies, each one with its own frame. Two revolute joints constraint the motion of the bodies, on O between the ground and body 1 and on A between bodies 1 and 2. It is easy to figure out that the system has 2 degrees of freedom so that the configuration of the system can be univoquely defined from angles q_1 and q_2 indicated on the figure.

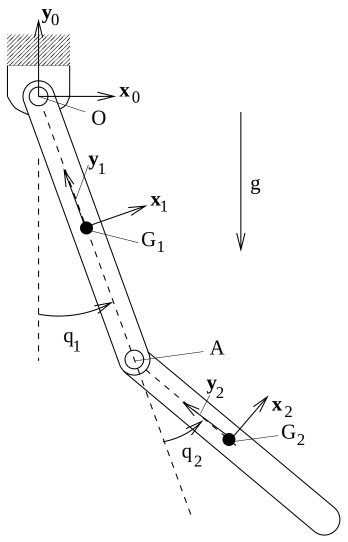


Figure 7: Double pendulum

Writing the routine SetInertiaData is just the matter of assigning the right variables

```
void SetInertiaData()
{
// Inertia data for body 0
body[0].mass=1.1; body[0].PhiG.put(1,1,body[0].mass*l1*l1/12);
// Inertia data for body 1
body[1].mass=0.9; body[1].PhiG.put(1,1,body[1].mass*l2*l2/12);
}
```

The complete kinematics can be described in about ten explicit lines (q[0] and q[1] refer to $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$)

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

This small piece of code illustrates some interesting utilizations of the vec module

- the position matrix is built from the multiplication of predefined forms corresponding to rotations or displacements (Trotz and Tdisp);
- the homogeneous transformation matrix (TOG) comprises a rotation tensor (R) which can be used to build a vector expressed in the global frame (OG or AG2) from its local coordinates.

In this case where only the gravity is considered, the efforts would be easily described in the following way

```
void AddAppliedEfforts()
{
// Contribution of external applied forces
vec gravity(0,-9.81,0);
AddGravityForces(gravity);
}
```

The main part of an EasyDyn application is always based on the same model: specification of the number of degrees of freedom and bodies, initialization, call to the integration routine and closing. In the case of the double pendulum, we have

```
int main()
{
  // Initialisation and memory allocation
  nbrdof=2;  nbrbody=2; application="dp2";
  InitEasyDynmbs();
  // Initial configuration
  q[1]=1;
  // Let's go !
  NewmarkIntegration(5,0.01,0.005);
  // The clean way to finish !
  EndEasyDynmbs();
}
```

5 The symbolic part of EasyDyn

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

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

$$\{\boldsymbol{v}_i\}_0 = \frac{d}{dt}\{\boldsymbol{e}_i\}_0 = \sum_{j=1}^{n_{cp}} \frac{\partial\{\boldsymbol{e}_i\}_0}{\partial q_j} \cdot \dot{q}_j = \sum_{j=1}^{n_{cp}} \{\boldsymbol{d}_{i,j}\}_0 \cdot \dot{q}_j \tag{11}$$

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

$$\{\tilde{\boldsymbol{\omega}}_i\}_0 = \begin{pmatrix} 0 & -\omega_{z_i} & \omega_{y_i} \\ \omega_{z_i} & 0 & -\omega_{x_i} \\ -\omega_{y_i} & \omega_{x_i} & 0 \end{pmatrix}_0 = \dot{\boldsymbol{R}}_{0,i} \cdot \boldsymbol{R}_{0,i}^T \tag{12}$$

One further derivation then naturally leads to the accelerations

$$\{\boldsymbol{a}_i\}_0 = \frac{d}{dt}\{\boldsymbol{v}_i\}_0 \qquad \{\dot{\boldsymbol{\omega}}_i\}_0 = \frac{d}{dt}\{\boldsymbol{\omega}_i\}_0 \tag{13}$$

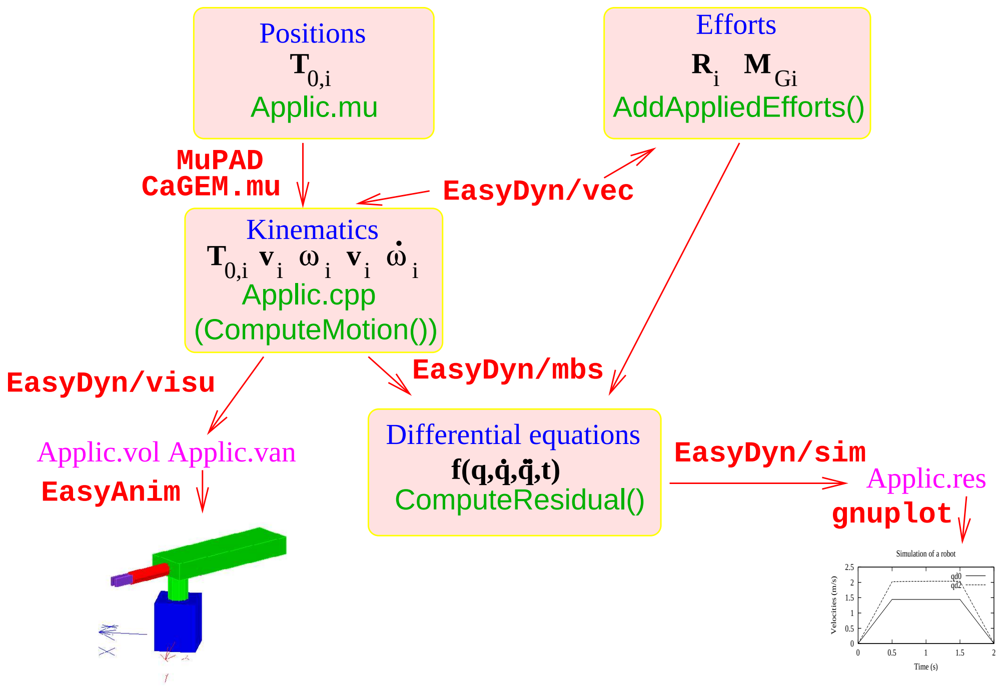


Figure 8: Data flow when combining `CAGeM` and the C++ library

A supplementary tool, called `CAGeM` (Computer Aided Generation of Motion) has been developed to help the users of `EasyDyn`. Practically, `CAGeM` is a MuPAD script which builds the core of a C++ application using `mbs`, to simulate the behaviour of a multibody system. The complete data flow when using `CAGeM` is illustrated in figure 8.

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

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

- some option flags.

The script CAGeM uses the symbolic derivation features of MuPAD to build the expressions of velocities and accelerations from the position matrices.
The following code illustrates the user file related to the example of the double pendulum

```
// Title of the application
title:="Simulation of a double pendulum":
nbrdof:= 2:
nbrbody:= 2:
// Gravity vector
gravity[1]:=0:
gravity[2]:=-9.81:
gravity[3]:=0:
// Eventual constants
l0:=1.2: l1:=1.1:
// Inertia characteristics
mass[0]:=1.1: mass[1]:=0.9:
Ixx[0]:=1: Ixx[1]:=1:
Iyy[0]:=1: Iyy[1]:=1:
Izz[0]:=l0^2/12*mass[0]:
Izz[1]:=l1^2/12*mass[1]:
// Definition of the position matrices
T0G[0] := Trotz(q[0]) * Tdisp(0,-l0/2,0):
T0G[1] := Trotz(q[0]) * Tdisp(0,-l0,0) * Trotz(q[1]) * Tdisp(0,-l1/2,0):
// Initial conditions
qi[1]:=1:
// Simulation parameters
FinalTime:=5:
StepSave:=0.01:
StepMax:=0.005:
```

In this case, where the gravity is the only applied effort, no other task but compiling the resulting code is necessary. In some cases, the user will have to complete the procedure AddAppliedEfforts.

The CAGeM utility offers different options, activated by flags in the code describing the system. Let us mention the most significant ones

- PLOT: if this option is activated, a generic script file is generated to be used by gnuplot; the history of the configuration parameters and their first and second time derivatives is displayed on the screen and exported in postcript files;
- ANIM: if activated, the generated C++ code comprises the statements needed to generate a basic animation during the integration process (a box is attached to each body);
- LATEX_FR: allows the creation of a LaTeX report in French about the built application: inertia properties, expression of homogeneous transformation matrices, velocities, accelerations, some listings and the curves generated by the gnuplot script [2];
- LATEX_EN: same as LATEX_FR but the report is in English;

6 EasyDyn Model of AMRU5

The walking robot AMRU5[9] illustrated in figure 9 is the most comprehensive example treated so far with the help of EasyDyn. All recent improvements of EasyDyn are used: relative motion, dependent parameters and the direct determination of the partial velocities from MuPAD.

[2] this option is particularly appreciated by the students

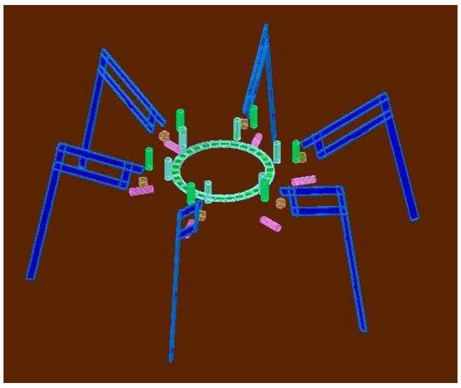


Figure 9: EasyDyn model of the walking robot AMRU5

Moreover, the model includes the dynamics of the DC motors and their digital controllers and a contact with friction between the legs and the ground. The model is actually driven by the position signals sent to the leg controllers by the central unit. The purpose of the model is to help in tuning new types of controllers and testing their robustness.

6.1 Kinematics

As we will see later, the contact with the ground is not considered in our model as a kinematic constraint, but as an external force. Kinematically, the robot is then completely free, which leads to a total number of 24 configuration parameters: 6 degrees of freedom to represent the spatial motion of the central body and 3 degrees of freedom to express the motion of each leg with respect to the central body. The distribution of the configuration parameters is indicated in table 1.

For the sake of illustration, the position matrix of the central body (body 0) will be specified in only one line in the MuPAD script

```
TOG[0]=Tdisp(q[0],q[1],q[2])*Trotx(q[4])*Troty(q[4])*Trotz(q[5])
```

Subsystem	Bodies	Configuration parameters
Central body	0	q_0 to q_5
Leg 1	1 to 8	q_6 to q_8
Leg 2	9 to 16	q_9 to q_{11}
Leg 3	17 to 24	q_{12} to q_{14}
Leg 4	25 to 32	q_{15} to q_{17}
Leg 5	33 to 40	q_{18} to q_{20}
Leg 6	41 to 48	q_{21} to q_{23}

Table 1: Distribution of configuration parameters

Each leg actually consists of 8 bodies: the support part, the four links of the pantograph and the rotors of the 3 motors. The latter have been introduced for a more consistent representation of the inertia effects. Taking into account the central body, the total number of bodies is then equal to 49. The legs are numbered as indicated in figure 10.

The fact that the pantograph mechanism involves kinematic loops doesn't prevent the system to be analysed by EasyDyn. However, a manual work must be done so as to express the situation of each body from the chosen configuration parameters, which correspond to the displacements of the driving points, and are denoted q_7 and q_8 for leg 1 (figure 11).

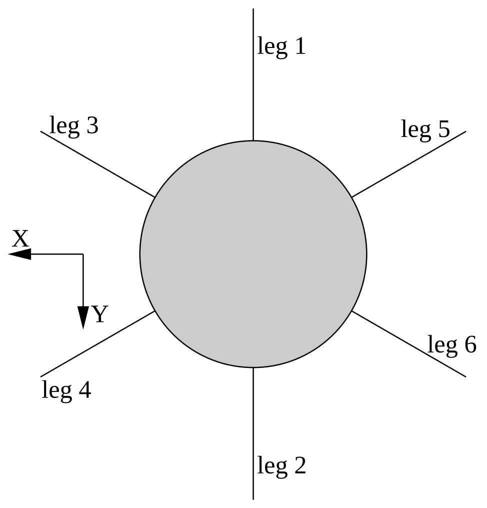


Figure 10: Numbering of the legs of AMRU5

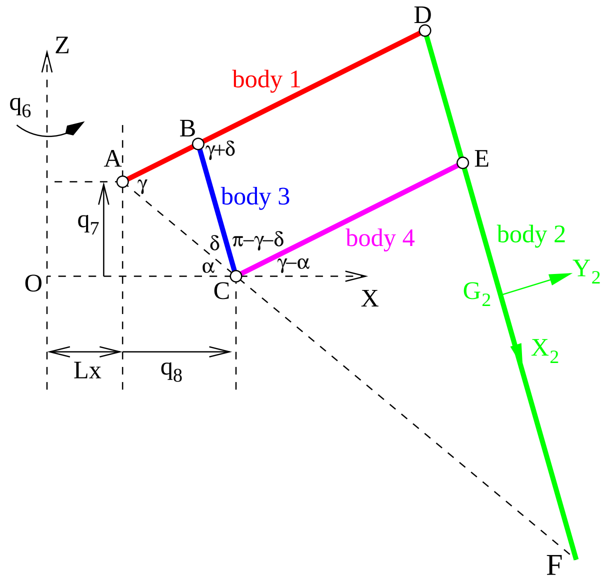
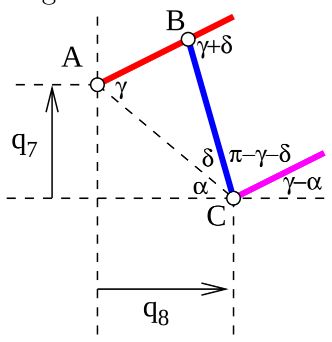


Figure 11: Kinematic analysis of leg 1

The work begins by the determination of the angles α, β and γ, in terms of q_7, q_8 and known lengths



$$AC = \sqrt{q_7^2 + q_8^2} \tag{14}$$

$$\alpha = \arctan(\frac{q_7}{q_8}) \tag{15}$$

$$\delta = \arccos(\frac{AC^2 + BC^2 - AB^2}{2 \cdot AC \cdot BC}) \tag{16}$$

$$\gamma = \arccos(\frac{AB^2 + AC^2 - BC^2}{2 \cdot AB \cdot AC}) \tag{17}$$

Angles α, β and γ are defined in `EasyDyn` as dependent parameters (vs configuration parameters). Once their expressions are known in terms of the configuration parameters, they can be used to express the position matrices. For example, the position matrix of body 2 (figure 11) will be expressed with respect to the one of the central body (body 0), will be written under `MuPAD`

```
TrefG[2]=TRotz(q[6])*Tdisp(Lx,0,q7)*Troty(α − γ)*Tdisp(AD,0,0)
                    *Troty(γ + δ)*Tdisp(DF/2,0,0)
```

where q_6 represents the rotation angle, about the vertical axis, of the leg with respect to the central body.

The complete kinematics of the 8 bodies of each leg is specified in the MuPAD file in only about 20 lines. After symbolic treatment by the `CAGeM` utility under `MuPAD`, the generated C++ file provides namely the routine `ComputeMotion` which calculates positions, velocities and accelerations of all bodies in terms of $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and, $\ddot{\boldsymbol{q}}$. Only this routine can already be very useful in robotics: it solves the direct kinematic problem and provides, from the speed expressions, the jacobian matrix which can be used for solving the inverse problem.

6.2 Expression of efforts

To complete the model, the efforts must be specified. Apart from gravity which is already treated by `CAGeM`, the efforts to be considered are the motor torques and the contact forces

A preliminary study has shown that the first-order model of a DC motor is sufficient due to the slowness of the mechanical motions. The motor torque T_m applied on a rotor is then determined according to

$$T_m = \frac{K_m}{R}(u - K_b\omega) \tag{18}$$

with u the applied voltage (imposed by the controller), R the terminal resistance, K_b the back-electromotive constant and K_m the torque constant.

Introducing a contact force following the Coulomb laws in a simulation is always problematic. In a rigourous manner, the contact should be introduced either in the form of kinematic constraints, or in the form of external forces, according to the contact conditions: contact or non contact, adhesion or slipping. This approach makes the simulation very difficult as transitions must be managed with changes in the number of degrees of freedom. Another solution consists in introducing some compliance in the contact so as to be able to express the contact forces only in terms of relative position and velocity. In our case, the contact between the foot and the ground is modelled as the contact between a point and a plane. The normal force is given by

$$\begin{aligned} F_n &= K_n pen^{p_k} + C_{damp} pen^{p_d} \frac{d\ pen}{dt} \text{ if } pen > 0 \\ &= 0 \text{ if } pen <= 0 \end{aligned} \tag{19}$$

where pen is the penetration of the point in the plane and K_n, p_k, C_{damp} and p_d are constants chosen according to the nature of the contact.

The tangential contact force is calculated by

$$\begin{aligned} F_t &= -fF_n \frac{\boldsymbol{v}_g}{v_g lim} \text{ if } ||\boldsymbol{v}_g|| < v_{glim} \\ &= -fF_n \frac{\boldsymbol{v}_g}{||\boldsymbol{v}_g||} \text{ if } ||\boldsymbol{v}_g|| > v_{glim} \end{aligned} \tag{20}$$

where f is the friction coefficient, $\boldsymbol{v}_g$ the slip velocity and v_{glim} a limit slip velocity from which the friction force fully corresponds to the law of Coulomb. The parameter v_{glim} represents some tolerance with respect to adhesion and is chosen small enough to represent the adhesion in a satisfying way with respect to the considered application (0.1 mm/s here).

7 Modelling the controllers

In this first attempt, only a PID controller (with a low pass filter on the derivative term to limit the influence of noise) has been considered. In Laplace form, it is written

$$U(s) = K_p E(s) + K_i \frac{E(s)}{s} + K_d \frac{E(s)}{1 + s\frac{K_d}{N}} \tag{21}$$

with U(s) and E(s) the Laplace transforms of voltage u and control error ϵ, and K_p, K_i, K_d and N the constants of the controller.

For simulation, the previous relationship is rewritten in its equivalent differential form

$$u + \dot{u}\frac{K_d}{N} = \left(K_p + \frac{K_i K_d}{N})\right)\epsilon + K_i \int \epsilon\, dt + \left(K_d + \frac{K_p K_d}{N})\right)\dot{\epsilon} \tag{22}$$

which can be added naturally to the differential equations of motion. A new differential variable q_i is introduced as well, such that $\dot{q}_i$=u to get a set of consistent second-order equations of motion.

Let us note that one supplementary differential equation has to be considered

$$\ddot{q}_j + \omega_0 \dot{q}_j = \omega_0 \epsilon \tag{23}$$

from which $\dot{\epsilon}$ and $\int \epsilon\, dt$ are given by $\ddot{q}_j$ and q_j. In this equation, ω_0 is chosen sufficently large with respect to the frequency bandwidth of the closed-loop system.

The model of AMRU5 with all controllers then involves 60 second-order differential equations from which 24 represent the equations of motion while the 36 other ones represent the controllers.

This continuous model has been considered for verification only. Practically, the controllers work in discrete-time and evaluate the latter expressions from finite differrences. Only the 24 equations of motion are then retained and the determination of the voltages by the controller is programmed in a routine called a given procedure at regular time intervals during the integration process. Non linearities like the integrator windup and the saturation (maximum voltage of 12 V) are also implemented in the model.

8 Some typical results

The simulation considered here is a straight line walk in the Y direction (figure 10) on a flat ground. A delay of 1 s is left to the system to reach its static equilibrium (contact deformation and controller stabilization), after which the maneuver begins. The reference signals are defined in the same way as it is actually programmed in the microcontroller. Namely, the inverse kinematics is not exact as some trigonometric functions, not available on the microcontroller, have been replaced by polynomials. The walking is also divided in typical sequences between which some velocity discontinuities can appear.

Figure 12 and 13 show the evolution of the position and velocity of the center of gravity of the central body. The global motion is regular but some discontinuities can be observed on the velocities. It is of interest to note the vertical displacement, due to the contact elasticity.

Figures 14 and 15 show the evolution of contact forces on legs 1 and 4. The evolution of the vertical force perfectly shows the load transfer from rear (leg 1) to front legs (leg 4) during a step. Figures show that the lateral component (X) is different from zero, due to the approximations in the inverse kinematics which induce some slips of the feet. Figures also show clearly when legs are in the air or in contact with the ground.

Finally, the evolution of the voltages as calculated by the controller are given for legs 1 and 4, respectively on figures 16 and 17. Voltages u_1, u_2 and u_3 refer respectively to the rotation of the leg with respect to the main body, and the vertical and horizontal motions of the pantograph.

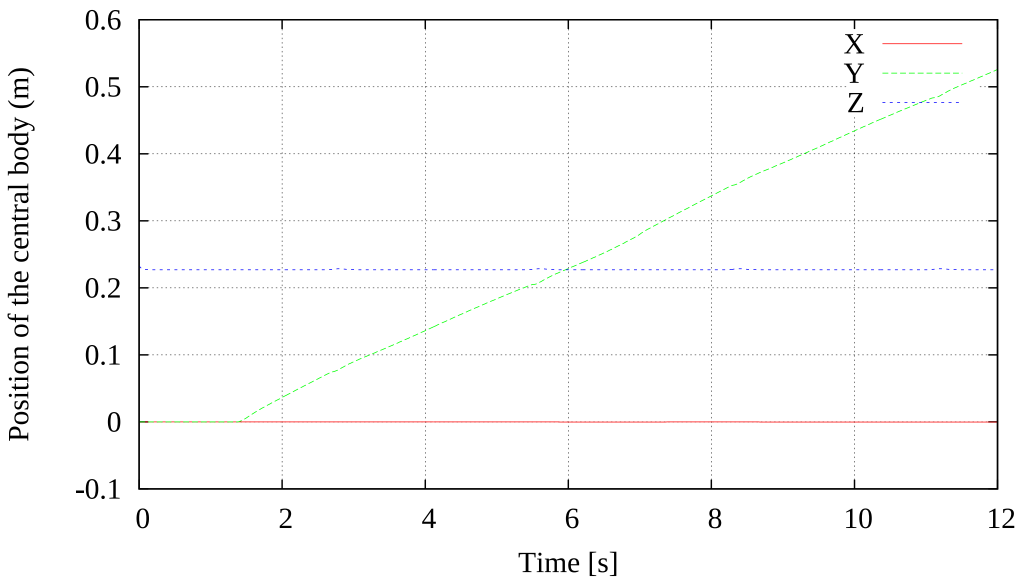


Figure 12: Position of the central body

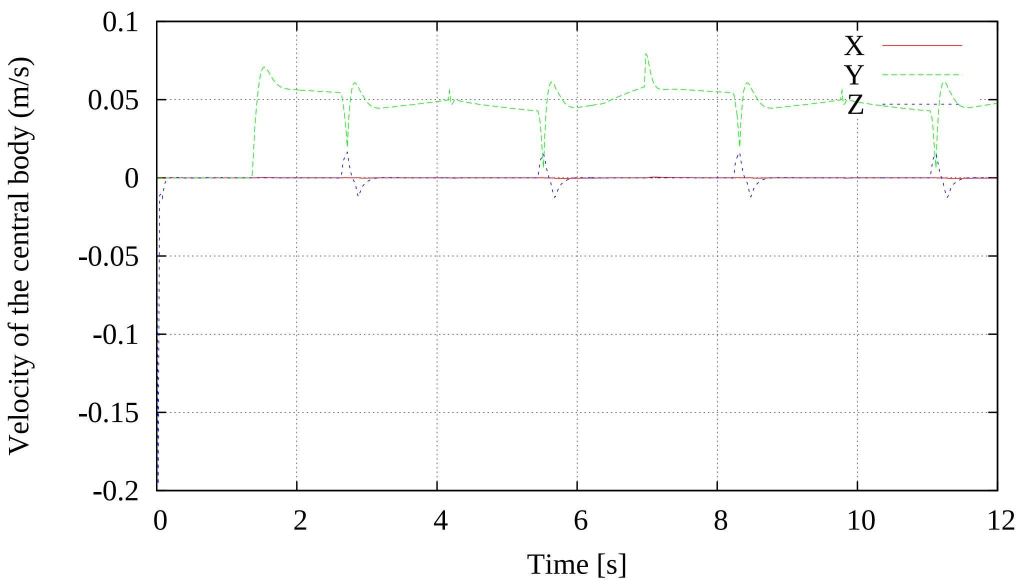


Figure 13: Velocity of the central body

9 Conclusion

We have presented the model of the walking robot AMRU5, which allows the simulation of walking maneuvers from the orders of the central controller. The model involves a comprehensive mechanical model accounting for the unilateral nature of the contact and includes a first-order model of the electric motors as well as the controllers.

The presented approach, based on the `EasyDyn` freamework, has the advantage to lead to a dedicated simulation program in C++ which can be modified at will by the user to introduce new features in the model, independently from any software editor. The kinematics of the mechanical model is generated symbolically from only the position information given in the compact form of homogeneous transformation matrices. The differential equations of motion are on the other hand constructed numerically from the kinematics and the efforts applied on each body.

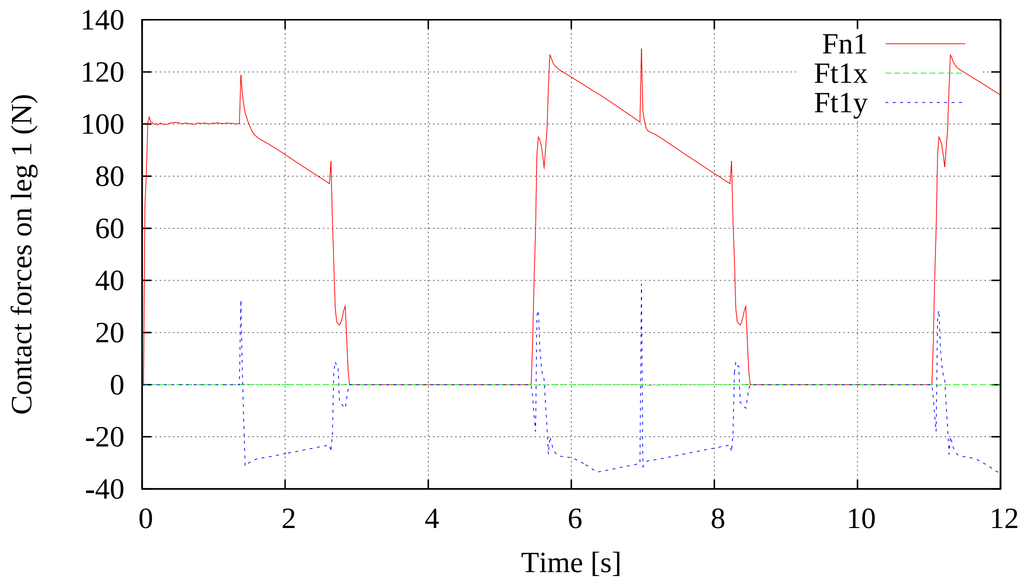


Figure 14: Contact forces for leg 1

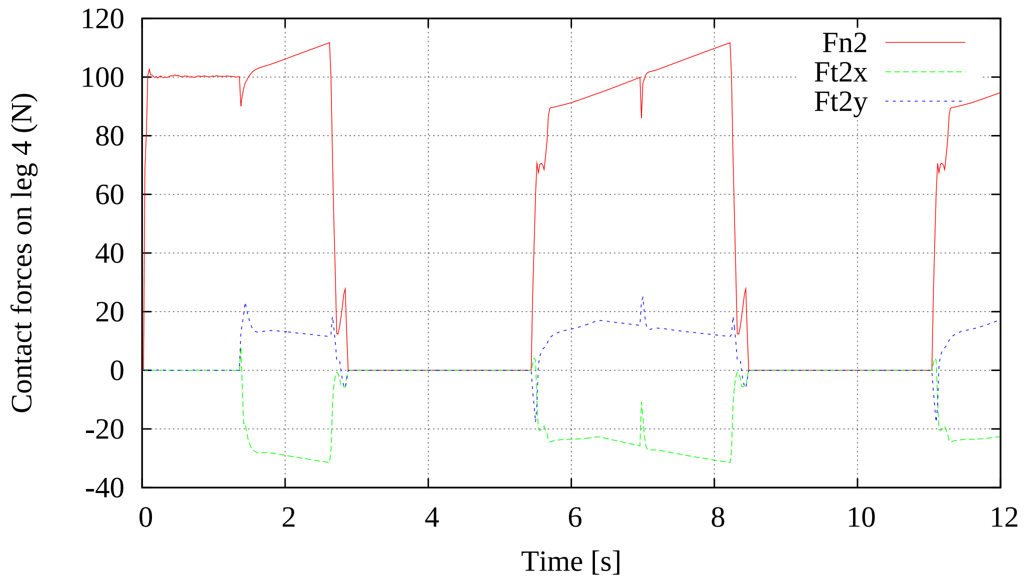


Figure 15: Contact forces for leg 4

The development prospectives concern the implementation of the actual neuro-fuzzy controller and friction losses in the transmissions. At end, the complete model is expected to help in testing and tuning other types of controllers or general behavioral strategies related to the actual mission of the robot.

References

[1] J.-C. Habumuremyi, *Adaptive Neuro-Fuzzy Control for a Walking Robot*, Phd Thesis, Vrije Universiteit Brussel, 2004

[2] C. Conti, P. Dehombreux, O. Verlinden and S. Datoussaid, *ACIDYM, a Modular Software for Computer-Aided Learning of Kinematic and Dynamic Analyis of Multibody Systems* in

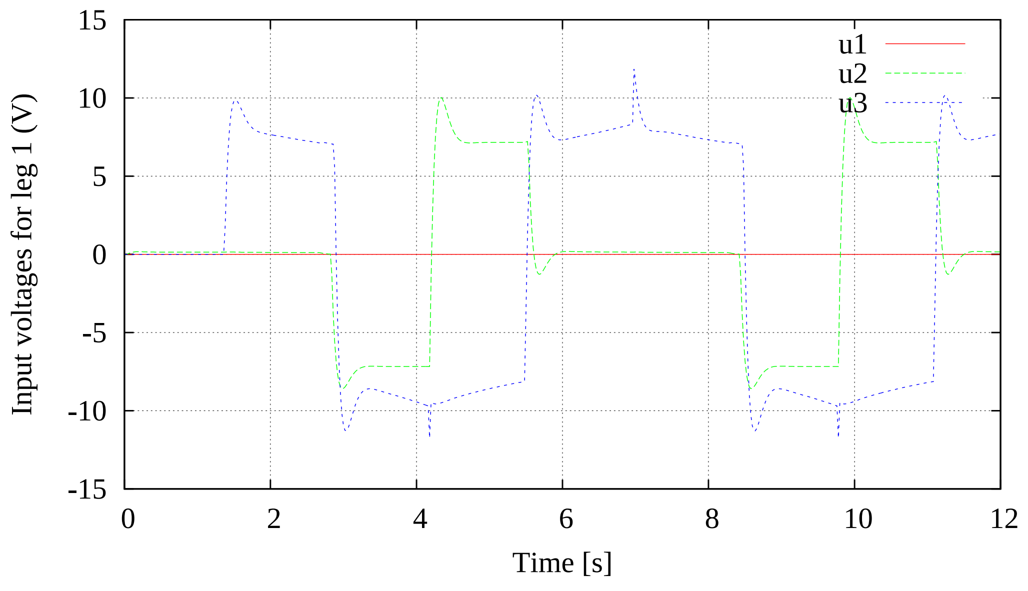


Figure 16: Input voltages for leg 1

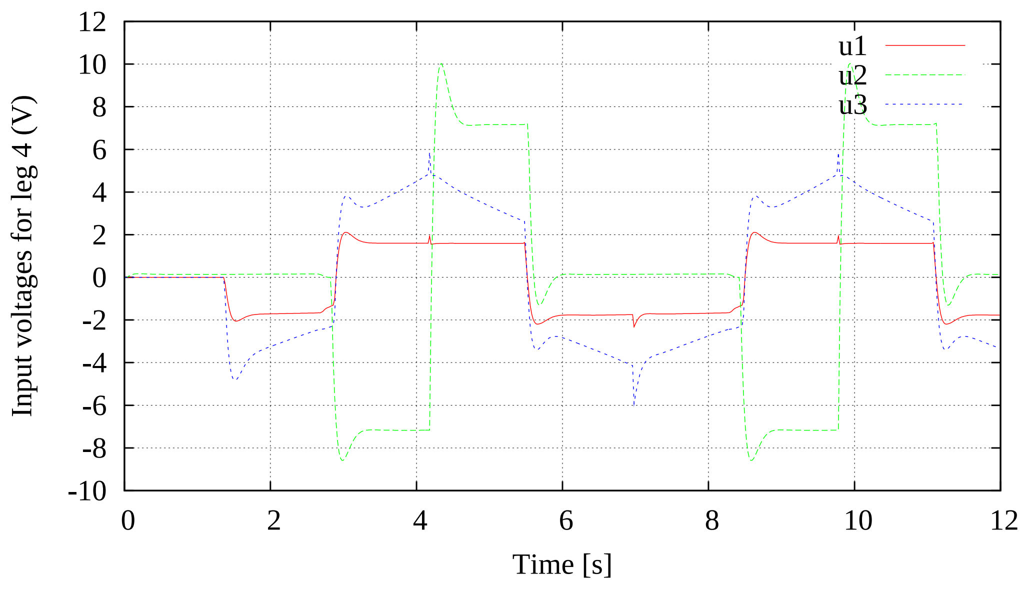


Figure 17: Input voltages for leg 4

Advanced Multibody System Dynamics: Simulation and Software Tools, W. Schiehlen,Kluwer Academic Publishers, 1993

[3] O. Verlinden, G. Kouroussis and C. Conti, *Open source symbolic and numerical tools for the simulation of multibody systems*, Electronic proceedings of the 6th National Congress on Theoretical and Applied Mechanics (on CD), Ghent, Belgium, 2003

[4] O. Verlinden, G. Kouroussis and C. Conti, *EasyDyn, a framework based on free symbolic and numerical tools for teaching multibody sytems*, Electronic proceedings of the ECCOMAS Thematic Conference “Multibody Dynamics 2005”, Madrid, Spain, 21-24 June 2005

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

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

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

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

[9] J.-C. Habumuremyi, Y.Baudoin and P.Kool, *Adaptative Neuro-fuzzy Control of AMRU-5, a six-legged walking robot*, IARP Workshop Hudem 2004, Brussels, 16-18 june, 2004