A MULTIBODY USER-SPACE HARD REAL-TIME ENVIRONMENT FOR THE SIMULATION OF SPACE ROBOTS

M. Attolico and P. Masarati
Dipartimento di Ingegneria Aerospaziale,
Politecnico di Milano
via La Masa 34, 20156 Milano Italy
{attolico,masarati}@aero.polimi.it

Abstract

The paper presents the interface of an existing open source multibody/multidisciplinary simulation framework, MBDyn, to RTAI/NEWLXRT for hardware/software-in-the-loop simulations of flexible space manipulators. The adaptation of the multibody software to the needs of real-time simulation required a minimal effort, basically focused on specializing existing input/output facilities to the use of RTAI/NEWLXRT mailboxes and on eliminating or wrapping few system calls. Preliminary applications to a simulation similar to that of the space robots control are presented, with different levels of sophistication and of integration into real-time control and visualization environments such as Matlab/Simulink Real-Time Workshop through the use of RTAILab.

1 Introduction

A general purpose, scalable multibody analysis framework, with generic and specialized element libraries for structural dynamics, aeroelasticity, hydraulics and electric circuits modeling, is usually believed to be less than ideal for real-time simulation. Typical realizations of mechanical systems real-time simulation are based on a dedicated modeling approach, based on model reduction by means of hierarchical definition of the kinematic degrees of freedom, and on exploitation of topological features of the system, e.g. tree structures, to achieve very efficient system solution (1, 2).

However, when detailed and sophisticated analyses are required, e.g. structural and joint deformability nonlinearities, motor and control dynamics incorporated into the simulation, a general purpose software can save the large implementation efforts required by dedicated, minimal set multibody models.

A modern approach to this class of problems, deeply rooted in the golden era of scientific computing when memory and time constraints were very stringent even without considering real-time execution, is represented by the symbolic manipulation of equations. It consists in symbolically eliminating the redundant unknowns from the beginning, to generate minimal set object code for later execution (3, 4). However, it is not yet clear how this approach can efficiently handle models entailing the dynamics of large deformable parts, characterized by arbitrary constitutive laws and subjected to non-conservative, nonlocal force fields.

This challenging task has been accomplished by exploiting the good performances of MBDyn in implicitly integrating complex systems with appreciable accuracy at a relatively limited computational cost. The adaptation of the multibody software to the needs of real-time simulation required a minimal effort, basically focused on specializing existing input/output facilities to the use of RTAI/NEWLXRT mailboxes and on eliminating inessential and wrapping essential non hard real-time system calls. Applications to the simulation of space robots control are presented, with different levels of sophistication and integration in real-time control and visualization environments as Matlab/Simulink Real-Time Workshop. The control is graphically designed in Simulink; the connectivity with the real-time layer occurs through S-functions; the controller is then automatically converted into a compiled program by Real-Time Workshop. All the hardware/software-in-the-loop simulation and process control is monitored and interfaced to the user by RTAILab. The real-

time extensions to MBDyn for RTAI/NEWLXRT will be available in a future release of the software package.
The paper is organized as follows: a description of the multibody software MBDyn is given in section 2; the MBDyn-RTAI/NEWLXRT interface is described in section 3; section 4 illustrates how the control program is produced; the simulated tests are presented in section 5, and conclusions are drawn in section 6.

2 MBDyn Software

MBDyn is an open source multibody simulation software, developed by the "Dipartimento di Ingegneria Aerospaziale" of the University Politecnico di Milano (`http://www.aero.polimi.it/~mbdyn/` or `http://www.mbdyn.org/`), that solves initial value multiphysics problems by numerical integration. The modeling is essentially based on connecting multiphysics nodes, the entities that make degrees of freedom available, by means of elements, the entities that write equations, that can be chosen in a predefined library, which can be extended by the user in many ways, e.g. by directly coding new elements or by writing run-time loaded modules.
MBDyn started as a general purpose multibody analysis software, originating from the need of an open tool for the analysis of rotorcraft dynamics (5), and eventually evolved in a multiphysics simulation software for the integrated analysis of multidisciplinary problems (6).
The formulation is classified as "redundant", in the sense that all the unknowns for each independent field (displacements and rotations, electric potential, hydraulic pressure, abstract scalar degrees of freedom) are directly considered during the simulation, each generating a differential equation. When algebraic constraints are required, they are explicitly added to the system, which becomes a Differential-Algebraic Equations (DAE) system, and the corresponding reaction unknowns (forces and couples, electric currents, hydraulic flows, abstract scalar efforts) are added to the list of degrees of freedom; thus no model reduction occurs.
The problem is solved by integrating in time the resulting nonlinear DAE system as is, by means of implicit A/L-stable integration schemes that allow to control the stability of the integration by means of tunable algorithmic dissipation (7).
This approach is far from ideal for real-time applications, because the resulting underlying linear algebra problem to be solved at each iteration can become quite large. For instance, a careless modeling of a 6-dof rigid robot can result in $7 \times 12 = 84$ dofs for the structural nodes (12 per node) and $6 + 6 \times 5 = 36$ dofs for the constraints (6 for the ground constraint and 5 for each revolute joint), not considering those required by the control; by carefully crafting the model, e.g. by neglecting the inertia of some components, if acceptable, and by carefully implementing the ground constraint, one can save a couple of dozens degrees of freedom, but the size of the problem remains in the order of 100 dof.
Nonetheless, if model sophistication requires specific features, not available, or not worth the implementation effort in dedicated real-time software, and time constraints are not too stringent, it is efficient enough to allow software/hardware in the loop simulation for a wide range of problems.

3 MBDyn Interface to NEWLXRT

In order to run a program like MBDyn in real-time, the simplest and most versatile way is to use the NEWLXRT module of RTAI. This module allows to use all the services made available by RTAI in user space. The changes the program requires are minimal, basically consisting in (a) inserting the essential calls to NEWLXRT functions; if the program must be run in hard real-time, namely under RTAI scheduler, (b) any kind of Linux operating system calls must be eliminated, or anticipated before any hard real-time activity takes place, or (c) wrapped around NEWLXRT calls; finally, (d) appropriate I/O must be provided to the simulation process. Point (d) may be seen as a subcase of (c), but, owing to its importance in the control loop, and to the significant implications both in terms of cost, versatility and generality issues that may arise, it deserves special attention.
The first step is to initialize the real time task support struct(ure) (i.e. the *buddy*) by calling `rt_task_init()`, which links RTAI services to the standard Linux processes and threads. Then, the invocation of `start_rt_timer()` starts the real-time timer. The program execution can be made periodic, by calling `rt_task_make_periodic()`; otherwise the synchronization can be delegated to the controller, by making the multibody simulation task wait on a semaphore or by making the first control output mailbox blocking. At this point the program correctly runs in soft-real time, namely under Linux scheduler, but, if the latency must be reduced, and the real-time execution is to be made safe, the process needs to run in hard real-time, namely `rt_make_hard_real_time()` needs be invoked. This inhibits Linux processes and interrupts from preempting the execution.

The elimination and the wrapping of system calls in the multibody simulation process followed a few guidelines based on common sense. First of all, it is left to the user's experience to determine how much stack and heap the code will be required during regular execution; the desired amount of stack is reserved by automatically defining a dummy array of the desired size, by allocating anything needed before going to hard real-time mode, and then invoking

```
#include <sys/mman.h>
    /* snip */
    mlockall(MCL_CURRENT | MCL_FUTURE);
```

to reserve stack and heap and thus prevent page swapping during the execution. MBDyn already allowed to disable all the output on screen by means of command line switches, and the file dump of bulk results could be disabled by configuration file statements. This "soft" approach provided a very preliminary and rough disabling of I/O system calls. At this point, all the output had to occur through RTAI mailboxes (described later), which might be critical when tight timing constraints are in place, because although they perform very well for small message passing, they are not the right tool for massive bulk streaming to data storage media. Consider that, on typical architectures and with typical models not run in real-time, full results dumping takes over more than 50% of the execution time. A much more elegant solution has been implemented by adding a soft real-time messaging proxy, that is fed by MBDyn by means of mailboxes. The proxy runs at very low priority level, and message delivering is not critical.
Memory allocation is another potential place for system calls. During regular execution MBDyn does not require any dynamical allocation, it occurs all at startup. However, care must be given to third party libraries, e.g. sparse solvers. If memory allocation is required at run-time, one needs to wrap each memory manager call to kernel memory management routines.
To execute MBDyn in real-time, another problem to be solved is how the multibody simulation process communicates with the other ones to monitor and control the simulation. RTAI allows different means of inter-process communications, both local and remote (8). The use of RTAI mailbox services has been preferred over more sophisticated solutions because of their simplicity and generality. This allows MBDyn to send and/or receive data safely in hard real-time. The policy used in the data transmission never blocks the calling process, but the transmission is permitted only if the mailbox is available to receive/send data.
The communication facilities, namely arbitrary measures on the model and arbitrary input in terms of forces, control tensions or currents, have been implemented by exploiting the object-oriented design of MBDyn. This allowed to extend existing I/O devices to use RTAI mailboxes, hiding the implementation details in a very limited portion of code, and simultaneously making real-time I/O available to all elements and degrees of freedom in a broadly general manner.

4 The Control Program

The control process is used in the simulation with MBDyn to close the control loop. It is obtained using RTAILab tools, which allow to create, directly from Simulink or Scilab models, the control program with all the utilities required to monitor the simulation and to communicate with the multibody simulation process. This allows to simulate the entire control loop in a very realistic manner. In this work the Simulink/RTAILab interface was used to generate the control law. Two C-program S-functions have been implemented to handle the input/output mailboxes. These portions of code can be simply inserted into the Simulink model, resulting in a control program that is interfaced with the multibody simulation task. In detail, at the beginning of the execution the S-function looks for the MBDyn real-time task and for the input/output mailboxes. It syncronizes the two real-time programs, while performing the input/output operations during the simulation. The input S-function keeps the data read in a step in a buffer, so that if a mailbox read fails, e.g. because of a overrun, the last known data is available for the controller, representing a sort of best estimate of the actual measures. This policy was selected because sometimes, when using close to the time limit accurate models, MBDyn can overrun the time step, without excessive loss in the meaning of the simulation. On the controller output side the same policy has not been adopted for the read facility of MBDyn, because the controller is not expected to incur in time-overrun with the usual sample rates. The start up of the simulation is performed by starting MBDyn first, followed by the control program, which synchronizes itself with the multibody task.

5 Numerical Applications

The MBDyn-RTAI interface has been tested by modeling two different mechanical systems, controlled by an external real-time control process. The tests were run on a dual Athlon 2100 MP (1733 MHz) PC.

5.1 Simple Pendulum

The first system is a pendulum; the measures simulated by MBDyn are the angular position θ and its rate ω; it is controlled by a couple. The control law is simply proportional to the measures, so the controller does not have internal dynamics. In the proposed simulation the pendulum starts from its stable equilibrium position and is placed in the unstable equilibrium position. The sample frequency is 1 kHz. The results are shown in Figures 2, 1. Notice that the control loop works correctly even in adverse conditions, e.g. unstable system configurations.

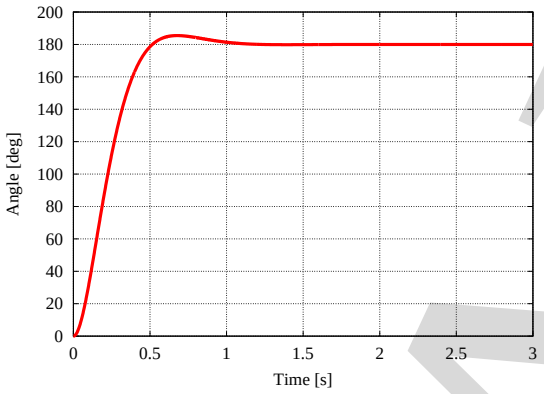

FIGURE 1: *Pendulum angular position*

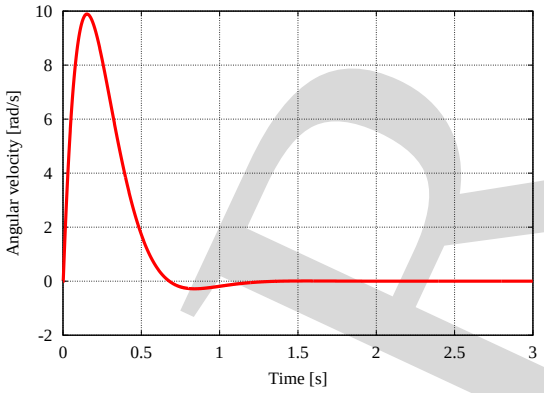

FIGURE 2: *Pendulum angular velocity*

5.2 Deformable Space Robot

The second system is a planar robot for aerospace applications, with two flexible arms, as shown in Figure 3, which has been built in the laboratories of the "Dipartimento di Ingegneria Aerospaziale" of the University Politecnico di Milano (9). The workspace is a horizontal plane.

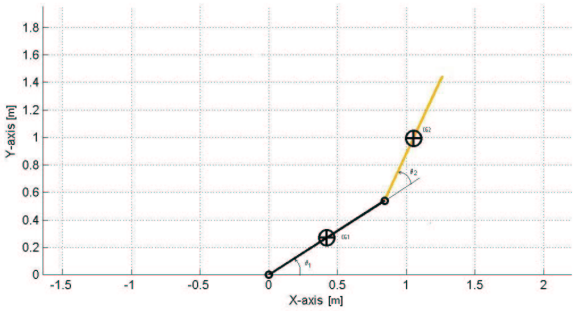

FIGURE 3: *Robot*

Length	1.0	m
Height	0.1	m
Link 1		
Material	Steel	
Thickness	5.0	mm
Mass	3.9	kg
Bending Stiffness	215.625	Nm^2
Link 2		
Material	Aluminum alloy	
Thickness	2.5	mm
Mass	0.685	kg
Bending Stiffness	9.336	Nm^2

TABLE 1: *Robot data*

This model is controlled by two couples applied at the two hinges, while the measures are the angular positions of the two joints θ_1, θ_2. This model is much more complicated, and consequently more computationally demanding than the previous one: the multibody program must solve 84 equations; furthermore, the model contains deformable beams, whose assembly is significantly time consuming. For this reason, the sample frequency had to be reduced to about 330Hz; this allowed to reduce the number of time overruns during the execution of MBDyn to a reasonable value, thus preserving the significance of the real-time simulation. The control task implements a PID controller for each joint. The presented simulation requires the robot to follow a prescribed path, consisting in driving each joint from 0 to 90 degrees by two parabola arcs in about 5 seconds. Figures 4 and 5, show the angular position and positioning errors of the two joints along the path; Figures 6–8 show the path of the end nodes of the two arms.
In order to reduce the execution latency, one of the two processors is dedicated to the execution of the MBDyn simulation task, whereas the other one executes the control task, RTAILab's simulation monitor, and manages the system interrupts.

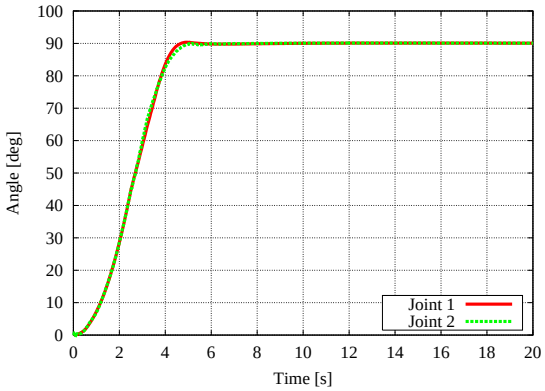

FIGURE 4: *Robot joint angular position*

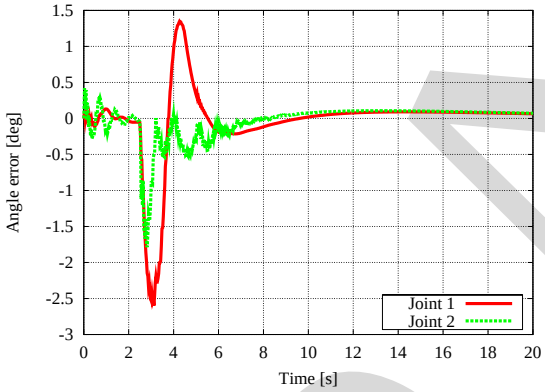

FIGURE 5: *Robot joint angular position error*

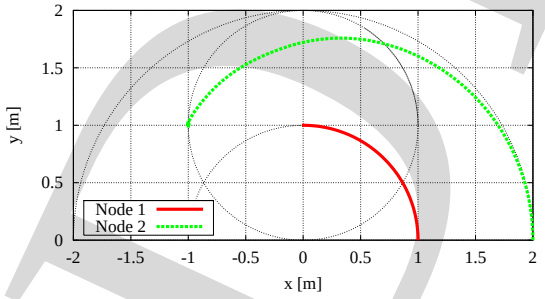

FIGURE 6: *robot end node path*

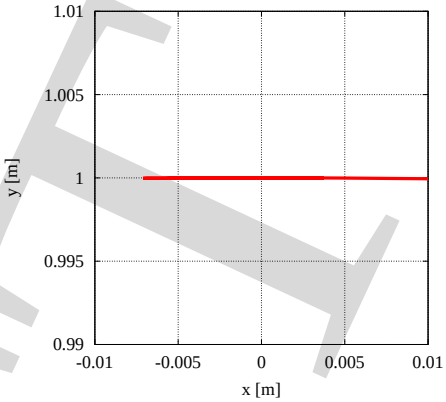

FIGURE 7: *Robot end node 1 path detail*

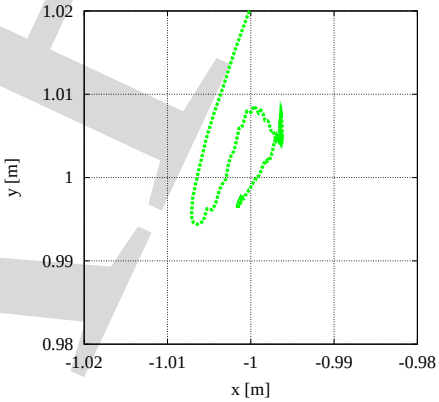

FIGURE 8: *Robot end node 2 path detail*

Note that the trajectory of node 1 basically implies no axial extension of the first link, as expected, and the positioning error at the extremity is essentially tangential to the trajectory, related to joint positioning dynamics and very limited link flexural deflection; the trajectory of node 2 reflects the superimposition of the two angular positioning errors, plus some flexural deflection of the second link, which is significantly less rigid than the first one. No friction in the joints has been modeled, nor any electric motor internal dynamics has been considered; these aspects of robot simulation will be addressed in future developments, pending substantial improvements of MBDyn in these areas.

6 Conclusions

A general purpose multibody/multiphysics simulation software has been modified to allow hard real-time scheduling by means of RTAI/NEWLXRT. Its application to simple control problems, in a Matlab/Simulink Real-Time Workshop environment, under RTAILab supervision, has been presented by

means of simple robot problems. The test cases described above show how a general purpose simulation software can be easily and proficiently transformed into a real-time simulator, thus providing the real-time simulation world a general analysis tool. A significant advantage of this approach is that the real-time simulations in principle might benefit of all the improvements of the general simulation tool. However, the intrinsic performance limitations of a non-optimized tool are apparent; although the proposed approach will definitely benefit from the expected continuous hardware development, the real-time execution of multibody simulations will require some additional software developments. The code MBDyn already allows some parallelization of the assembly and solution phases, by partitioning the problem, spawning the subdomain solutions on different machines and collecting the interface problem on a master node (10). While this strategy may not be very promising for real-time, because of transmission latency uncertainties, a multithreaded approach for SMP architectures might bring substantial performance improvements.

References

[1] Dae-Sung Bae, Ruoh-Shih Hwang, and Edward J. Haug. A recursive formulation for real-time dynamic simulation of mechanical systems. *Journal of Mechanical Design*, 113:158–166, June 1991.

[2] Javier Garcia de Jalon and Eduardo Bayo. *Kinematic and Dynamic Simulation of Multibody Systems: the Real Time Challenge*. Springer-Verlag, New York, 1994.

[3] Martin Otter and Hilding Elmqvist. Languages, libraries, tools, workshop and eu-project realsim. *Simulation News Europe*, pages 3–8, December 2000.

[4] Michael Tiller. *Introduction to Physical Modeling with Modelica*. Kluwer Academic Publishers, 2001.

[5] Gian Luca Ghiringhelli, Pierangelo Masarati, Paolo Mantegazza, and Mark W. Nixon. Multi-body analysis of a tiltrotor configuration. *Nonlinear Dynamics*, 19(4):333–357, August 1999.

[6] Pierangelo Masarati, Gian Luca Ghiringhelli, Massimiliano Lanz, and Paolo Mantegazza. Integration of hydraulic components in a multibody framework for rotorcraft analysis. In 26^{th} *European Rotorcraft Forum*, pages 57.1–10, The Hague, The Netherlands, 26–29 September 2000.

[7] Pierangelo Masarati, Massimiliano Lanz, and Paolo Mantegazza. Multistep integration of ordinary, stiff and differential-algebraic problems for multibody dynamics applications. In *XVI Congresso Nazionale AIDAA*, pages 71.1–10, Palermo, 24–28 Settembre 2001.

[8] Lorenzo Dozio and Paolo Mantegazza. Real time distributed control systems using RTAI. In *Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, Hakodate, Hokkaido, Japan, May 14–16 2003.

[9] Marcello Romano, Franco Bernelli-Zazzera, and Mauro Massari. Numerical and experimental results on closed loop tracking control of flexible manipulators. In *XVI Congresso Nazionale AIDAA*, Palermo, 24–28 Settembre 2001.

[10] Giuseppe Quaranta, Pierangelo Masarati, and Paolo Mantegazza. Multibody analysis of controlled aeroelastic systems on parallel computers. *Multibody System Dynamics*, 8(1):71–102, 2002.