SMARTMOBILE – AN ENVIRONMENT FOR GUARANTEED MULTIBODY MODELING AND SIMULATION

Ekaterina Auer and Wolfram Luther
IIIS, University of Duisburg-Essen, Lotharstr. 63, Duisburg, Germany
{auer,luther}@inf.uni-due.de

Keywords: Validated method, interval, Taylor model, initial value problem, guaranteed multibody modeling and simulation.

Abstract: Multibody modeling and simulation is important in many areas of our life from a computer game to space exploration. To automatize the process for industry and research, a lot of tools were developed, among which the program MOBILE plays a considerable role. However, such tools cannot guarantee the correctness of results, for example, due to possible errors in the underlying finite precision arithmetic. To avoid such errors and simultaneously prove the correctness of results, a number of so called validated methods were developed, which include interval, affine and Taylor form based arithmetics. In this paper, we present the recently developed multibody modeling and simulation tool SMARTMOBILE based on MOBILE, which is able to guarantee the correctness of results. The use of validated methods there allows us additionally to take into account the uncertainty in measurements and study its influence on simulation. We demonstrate the main concepts and usage with the help of several mechanical systems, for which kinematical or dynamic behavior is simulated in a validated way.

1 INTRODUCTION

Modeling and simulation of kinematics and dynamics of mechanical systems is employed in many branches of modern industry and applied science. This fact contributed to the appearance of various tools for automatic generation and simulation of models of multibody systems, for example, MOBILE (Kecskeméthy, 1993). Such tools produce a model (mostly a system of differential or algebraic equations or both) from a formalized description of the goal mechanical system. The system is then solved using a corresponding numerical algorithm. However, the usual implementations are based on finite precision arithmetic, which might lead to unexpected errors due to round off and similar effects. For example, unreliable numerics might ruin an election (German Green Party Convention in 2002) or cost people lives (Patriot Missile failure during the Golf War), cf. (Huckle, 2005).

Aside from finite precision errors, possible measurement uncertainties in model parameters and errors induced by model idealization encourage the employment of a technique called interval arithmetic and its extensions in multibody modeling and simulation tools. Essential ideas of interval arithmetic were developed simultaneously and independently by several people whereas the most influential theory was formulated by R. E. Moore (Moore, 1966). Instead of providing a point on the real number axis as an (inexact) answer, intervals supply the lower and upper bounds that are guaranteed to contain the true result. These two numbers can be chosen so as to be exactly representable in a given finite precision arithmetic, which cannot be always ensured in the usual finite precision case. The ability to provide a guaranteed result supplied a name for such techniques – "validated arithmetics". Their major drawback is that the output might be too uncertain (e.g. $[-\infty;+\infty]$) to provide a meaningful answer. Usually, this is an indication that the problem might be ill conditioned or inappropriately formulated, and so the finite precision result wrong.

To minimize the possible influence of overestimation on the interval result, this technique was extended

with the help of such notions as affine (de Figueiredo and Stolfi, 2004) or Taylor forms/models (Neumaier, 2002). Besides, strategies and algorithms much less vulnerable to overestimation were developed. They include rearranging expression evaluation, coordinate transformations, or zonotopes (Lohner, 2001).

The second focus in this paper is MOBILE, an object oriented C++ environment for modeling and simulation of kinematics and dynamics of mechanical systems based on the multibody modeling method. Its central concept is a transmission element which maps motion and force between system states. For example, an elementary joint modeling revolute and prismatic joints is such a transmission element. Mechanical systems are considered to be concatenations of these entities. In this way, serial chains, tree type or closed loop systems can be modeled. With the help of the global kinematics, the transmission function of the complete system chain can be obtained from transmission functions of its parts. The inverse kinematics and the kinetostatic method (Kecskeméthy and Hiller, 1994) help to build dynamic equations of motion, which are solved with common initial value problem (IVP) solvers. MOBILE belongs to the numerical type of modeling software, that is, it does not produce a symbolic description of the resulting model. Only the values of output parameters for the user-defined values of input parameters and the source code of the program itself are available.

SMARTMOBILE (Simulation and Modeling of dynAmics in MOBILE: Reliable and Template based) enhances the usual, floating point based MOBILE with validated arithmetics and IVP solvers (Auer et al., 2006a). In this way, it can model and perform validated simulation of the behavior of various classes of mechanical systems including non-autonomous and closed-loop ones as well as provide more realistic models by taking into account the uncertainty in parameters.

In this paper, we give an overview of the structure and the abilities of SMARTMOBILE. First, the main validated techniques and software are referenced briefly in Section 2. In Section 3, the main features of MOBILE are described in short to provide a better understanding of the underlying structure of SMARTMOBILE. In Section 4, we describe the implementation of SMARTMOBILE in some detail and validate kinematical and dynamic behavior of several example systems with the help of this environment. We summarize the paper in Section 5. On the whole, we give an overview of the potential of validated methods in mechanical modeling, and, in particular, the potential of SMARTMOBILE.

2 VALIDATED METHODS AND SOFTWARE

To guarantee the correctness of MOBILE results, it is necessary to enhance this tool with validated concepts. Fortunately, we do not need to implement these concepts from scratch. In the last decades, various libraries were implemented that supported different aspects of validated calculus. In the first Subsection, we name several of these tools. After that, we give a very brief overview of interval arithmetic and an algorithm for solving IVPs (initial value problems) to provide a reference about the difference of validated methods to the usual floating point ones.

2.1 Validating Multibody Tools of the Numerical Type

To validate the results of multibody modeling and simulation software of the numerical type, the following components are necessary. First, the means are required to work with arithmetic operations and standard functions such as sine or cosine in a guaranteed way. Here, the basic principles of interval calculus and its extensions are used. Interval arithmetic is implemented in such libraries as PROFIL/BIAS (Knüppel, 1994), FILIB++ (Lerch et al., 2001), C-XSC (Klatte et al., 1993). LIBAFFA (de Figueiredo and Stolfi, 2004) is a library for affine arithmetic, whereas COSY (Berz and Makino, 2002) implements Taylor models.

Second, validated algorithms for solving systems of algebraic, differential or algebraic-differential equations are necessary. C-XSC TOOLBOX (Hammer et al., 1995) offers a general means of solving different classes of systems as well as an implementation in C-XSC. For IVP solving in interval arithmetic, there exist such packages as AWA (Lohner, 1988), VNODE (Nedialkov, 2002), and recently developed VALENCIA-IVP (Auer et al., 2006a). In the framework of Taylor models, the solver COSY VI (Berz and Makino, 1998) was developed.

Finally, almost all of the above mentioned solvers need means of computing (high order) derivatives automatically. Some of them, for example, COSY VI, use the facilities provided by the basis arithmetic in COSY. Interval implementations do not possess this facility in general; external tools are necessary in this case. The symbolic form of the mathematical model, which is necessary to be able to obtain derivatives automatically, is not available in case of software of the numerical type. However, a method called algorithmic differentiation (Griewank, 2000) offers a possibility to obtain the derivatives using the code of the

program itself.

There are two main techniques to implement algorithmic differentiation of a piece of program code: overloading and code transformation. In the first case, a new data type is developed that is capable of computing the derivative along with the function value. This new data type is used instead of the simple one in the code piece. The drawback of this method is the lack of automatic optimization during the derivative computation. FADBAD++ (Stauning and Bendtsen, 2005) is a generic library implementing this approach for arbitrary user-defined basic data types. The technique of code transformation presupposes the development of a compiler that takes the original code fragment and the set of differentiation rules as its input and produces a program delivering derivatives as its output. This approach might be difficult to implement for large pieces of code which are self-contained programs themselves. However, derivatives can be evaluated more efficiently with this technique. An implementation is offered in the library ADOL-C (Griewank et al., 1996).

This list of tools is not supposed to be complete. All of the above mentioned packages are implemented (or have versions) in C++, an important criterium from our point of view since MOBILE is also implemented in this language.

2.2 Theory Overview

In this Subsection, we consider the basic principles of validated computations using the example of interval arithmetic. First, elementary operations in this arithmetic are described. Then a basic interval algorithm for solving IVPs is outlined to give an impression of the difference to floating point analogues. In particular, the latter passage makes clear why automatic differentiation is unavoidable while simulating dynamics of mechanical systems, that is, solving systems of differential equations.

An interval $[\underline{x};\overline{x}]$, where $\underline{x}$ is the lower, $\overline{x}$ the upper bound, is defined as $[\underline{x};\overline{x}] = \{x \in \mathbb{R} : \underline{x} \le x \le \overline{x}\}$. For any operation $\circ = \{+,-,\cdot,/\}$ and intervals $[\underline{x};\overline{x}]$, $[\underline{y};\overline{y}]$, the corresponding interval operation can be defined as $[\underline{x};\overline{x}] \circ [\underline{y};\overline{y}] =$

$$[min(\underline{x}\circ\underline{y},\underline{x}\circ\overline{y},\overline{x}\circ\underline{y},\overline{x}\circ\overline{y}); max(\underline{x}\circ\underline{y},\underline{x}\circ\overline{y},\overline{x}\circ\underline{y},\overline{x}\circ\overline{y}))] \ .$$

Note that the result of an interval operation is also an interval. Every possible combination of $x \circ y$, where $x \in [\underline{x};\overline{x}]$ and $y \in [\underline{y};\overline{y}]$, lies inside this interval. (For division, it is assumed that $0 \notin [\underline{y};\overline{y}]$.)

To be able to work with this definition on a computer using a finite precision arithmetic, a concept of a machine interval is necessary. The machine interval has machine numbers as the lower and upper bounds. To obtain the corresponding machine interval for the real interval $[\underline{x};\overline{x}]$, the lower bound is rounded down to the largest machine number equal or less than $\underline{x}$, and the upper bound is rounded up to the smallest machine number equal or greater than $\overline{x}$.

Consider an algorithm for solving the IVP

$$\dot{x}(t) \quad = \quad f(x(t)), \quad x(t_0) \quad \in \quad [x_0], \qquad (1)$$

where $t \in [t_0,t_n] \subset \mathbb{R}$ for some $t_n > t_0$, $f \in C^{p-1}(\mathcal{D})$ for some $p > 1$, $\mathcal{D} \subseteq \mathbb{R}^m$ is open, $f : \mathcal{D} \mapsto \mathbb{R}^m$, and $[x_0] \subset \mathcal{D}$. The problem is discretized on a grid $t_0 < t_1 < \cdots < t_n$ with $h_{k-1} = t_k - t_{k-1}$. Denote the solution with the initial condition $x(t_{k-1}) = x_{k-1}$ by $x(t;t_{k-1},x_{k-1})$ and the set of solutions $\{x(t;t_{k-1},x_{k-1}) \mid x_{k-1} \in [x_{k-1}]\}$ by $x(t;t_{k-1},[x_{k-1}])$. The goal is to find interval vectors $[x_k]$ for which the relation $x(t_k;t_0,[x_0]) \subseteq [x_k]$, $k = 1,\ldots,n$ holds.

The (simplified) kth time step of the algorithm consists of two stages (Nedialkov, 1999) :

1. Proof of existence and uniqueness. Compute a step size h_{k-1} and an a priori enclosure $[\tilde{x}_{k-1}]$ of the solution such that

(i) $x(t;t_{k-1},x_{k-1})$ is guaranteed to exist for all $t \in [t_{k-1};t_k]$ and all $x_{k-1} \in [x_{k-1}]$,

(ii) the set of solutions $x(t;t_{k-1},[x_{k-1}])$ is a subset of $[\tilde{x}_{k-1}]$ for all $t \in [t_{k-1};t_k]$.

Here, Banach's fixed-point theorem is applied to the Picard iteration.

2. Computation of the solution. Compute a tight enclosure $[x_k] \subseteq [\tilde{x}_{k-1}]$ of the solution of the IVP such that $x(t_k;t_0,[x_0]) \subseteq [x_k]$. The prevailing algorithm is as follows.

2.1. Choose a one-step method

$$x(t;t_k,x_k) = x(t;t_{k-1},x_{k-1}) + h_{k-1}\varphi(x(t;t_{k-1},x_{k-1})) + z_k \ ,$$

where $\varphi(\cdot)$ is an appropriate method function, and z_k is the local error which takes into account discretization effects. The usual choice for $\varphi(\cdot)$ is a Taylor series expansion.

2.2. Find an enclosure for the local error z_k. For the Taylor series expansion of order $p-1$, this enclosure is obtained as $[z_k] = h_{k-1}^p f^{[p]}([\tilde{x}_{k-1}])$, where $f^{[p]}([\tilde{x}_{k-1}])$ is an enclosure of the pth Taylor coefficient of the solution over the state enclosure $[\tilde{x}_{k-1}]$ determined by the Picard iteration in Stage One.

2.3. Compute a tight enclosure of the solution. If mean-value evaluation for computing the enclosures of the ranges of $f^{[i]}([x_k])$, $i = 1,...,p-1$, instead of the direct evaluation of $f^{[i]}([x_k])$ is used, tighter enclosures can be obtained.

Note that Taylor coefficients and their Jacobians (used in the mean-value evaluation) are necessary to be able to use this algorithm.

3 MOBILE

A transmission element, the basis of MOBILE, maps motion and loads between state objects (coordinate frames or variables) according to

$$\begin{array}{rclcrcl} q' & = & \phi(q), & & \dot{q}' & = & \mathbf{J}_\phi\,\dot{q}, \\ \ddot{q}' & = & \mathbf{J}_\phi\,\ddot{q}+\dot{\mathbf{J}}_\phi\,\dot{q}, & & Q & = & \mathbf{J}_\phi^{\mathrm{T}}\,Q' \ . \end{array} \qquad (2)$$

$\mathbf{J}_\phi$ is the Jacobian of ϕ, the mapping for the motion transmission. Other characteristics are vectors of dimension depending on the degrees of freedom of a mechanical system. Here, q and q' are the generalized positions, $\dot{q}$ and $\dot{q}'$ the velocities, $\ddot{q}$ and $\ddot{q}'$ the accelerations, as well as Q and Q' the forces of the transmission element in the original and final states, respectively. The transmission of force is assumed to be ideal. That is, power is neither generated nor consumed.

Models in MOBILE are concatenations of transmission elements. The overall mapping of this concatenation from the original state into the final one is obtained by the composition of the corresponding mappings of the intermediate states. Concatenated elements are considered as a single transmission element. This helps to solve the task of the global kinematics: to obtain the positions, the orientations, the velocities, and the accelerations of all bodies of a mechanical system from the given q, $\dot{q}$, and $\ddot{q}$.

All transmission elements are derived from the abstract class `MoMap`, which supplies their main functionality including the methods `doMotion()` and `doForce()` for transmission of motion and force. For example, elementary joints are modeled by the `MoMap`-derived class `MoElementaryJoint`. Besides, there exist elements for modeling mass properties and applied forces. The corresponding representations of the mapping (2) for these elements are described in (Kecskeméthy, 1993).

Transmission elements are assembled to chains implemented by the class `MoMapChain`. The methods `doMotion()` and `doForce()` can be used for a chain representing the system to determine the corresponding composite transmission function.

To model dynamics of a mechanical system, the equations of motion have to be built and solved. Their minimal form is given by

$$M(q;t)\,\ddot{q}+b(q,\dot{q};t)=Q(q,\dot{q};t)\ , \qquad (3)$$

where $M(q;t)$ is the generalized mass matrix, $b(q,\dot{q};t)$ the vector of generalized Coriolis and centrifugal forces, and $Q(q,\dot{q};t)$ the vector of applied forces. The class `MoEqmBuilder` is responsible for generation of such equations, that is, computation of M, b, and Q for each given q, $\dot{q}$, and t.

After the introduction of a state vector $x=\left[q^T,\dot{q}^T\right]^T$, the state-space form of the state equations is obtained as

$$\dot{x}=\left[\begin{array}{c}\dot{q}\\ \ddot{q}\end{array}\right]=\left[\begin{array}{c}\dot{q}\\ -M^{-1}\widehat{Q}\end{array}\right]\ , \qquad (4)$$

where $\widehat{Q}(q,\dot{q};t)=b(q,\dot{q};t)-Q(q,\dot{q};t)$. This is the responsibility of the class `TMoMechanicalSystem`.

Finally, an IVP corresponding to (4) is solved by an appropriate integrator algorithm, for example, Runge–Kutta's using the class `MoRungeKuttaIntegrator` derived from the basic class `MoIntegrator`.

MOBILE models and simulates mechanical systems directly as executable programs. This allows the user to embed the resulting modules in existing libraries easily. Besides, the core of MOBILE is extendable owing to its open system design.

As already mentioned, MOBILE belongs to the numerical type of the modeling software, by which we mean that it does not produce the symbolic description of the resulting mathematical model, as opposed to the symbolical type. In the latter case, the process of validation of the model is basically reduced to the application of the validated methods to the obtained system of equations. In the former case, it is necessary to integrate verified techniques into the core of the software itself, the task which cannot always be solved since many modeling tools are not open source.

4 SMARTMOBILE

In this Section, we first describe the main features of the recently developed multibody modeling and simulation tool SMARTMOBILE which produces guaranteed results in the constraints of the given model. There, the modeling itself can be enhanced by taking into account the uncertainty in parameters, which might result, for example, from measurements. After that, we demonstrate the possibilities SMARTMOBILE offers by simulating kinematical and dynamic behavior of two example systems in a validated way.

4.1 Main Features

The focus of SMARTMOBILE is to model and simulate dynamics of mechanical systems in a guaranteed way. The concept behind MOBILE, however, presupposes that kinematics is also modeled (almost as a by-product) and so it is easy to simulate it afterwards. That is why SMARTMOBILE is one of the rare validated tools that possess both functionalities.

To simulate dynamics, it is necessary to solve an IVP for the differential(-algebraic) equations of motion of the system model in the state space form. As already mentioned, validated IVP solvers need derivatives of the right side of these equations. They can be obtained using algorithmic differentiation, the method that is practicable but might consume a lot of CPU time in case of such a large program as MOBILE. An alternative is to make use of the system's mechanics for this purpose. This option is not provided by MOBILE developers yet and seems to be rather difficult to algorithmize for (arbitrary) higher orders of derivatives. That is why it was decided to employ the first possibility in SMARTMOBILE.

To obtain the derivatives, SMARTMOBILE uses the overloading technique. In accordance with Subsection 2.1, all relevant occurrences of `MoReal` (an alias of `double` in MOBILE) have to be replaced with an appropriate new data type. Almost each validated solver needs a different basic validated data type. Therefore, the strategy in SMARTMOBILE is to use pairs type/solver. To provide interval validation of dynamics with the help of VNODE-based solver `TMoAWAIntegrator`, the basic data type `TMoInterval` including data types necessary for algorithmic differentiation should be used. The data type `TMoFInterval` enables the use of `TMoValenciaIntegrator`, an adjustment of the basic version of VALENCIA-IVP. The newly developed `TMoRiotIntegrator` is based on the IVP solver from the library RIOT, an independent C++ version of COSY and COSY VI, and requires the class `TMoTaylorModel`, a SMARTMOBILE-compatible wrapper of the library's own data type `TaylorModel`. Analogously, to be able to use an adjustment of COSY VI, the wrapper `RDAInterval` is necessary. Modification of the latter solver for SMARTMOBILE is currently work in progress.

In general, kinematics can be simulated with the help of all of the above mentioned basic data types. However, other basic data types might become necessary for more specific tasks such as finding of equilibrium states of a system since they require specific solvers. SMARTMOBILE provides an option of modeling equilibrium states in a validated way with the help of the interval-based data type `MoFInterval` and the class `MoIGradientStaticEquilibriumFinder`, a version of the zero-finding algorithm from the C-XSC TOOLBOX.

The availability of several basic data types in SMARTMOBILE points out its second feature: the general data type independency through its template structure. That is, `MoReal` is actually replaced with a placeholder and not with a concrete data type. For example, the transmission element `MoRigidLink` from MOBILE is replaced with its template equivalent `TMoRigidLink`, the content of the placeholder for which (e.g. `TMoInterval` or `MoReal`, cf. Figure 1) can be defined at the final stage of the system assembly. This allows us to use a suitable pair consisting of the data type and solver depending on the application at hand. If only a reference about the form of the solution is necessary, `MoReal` itself and a common numerical solver (e.g. Runge-Kutta's) can be used. If a relatively fast validation of dynamics without much uncertainty in parameters is of interest, `TMoInterval` and `TMoAWAIntegrator` might be the choice. For validation of highly nonlinear systems with a considerable uncertainty, the slower combination of `TMoTaylorModel` and `TMoRiOTIntegrator` can be used.

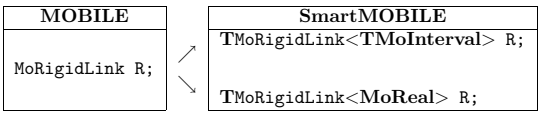


Figure 1: Template usage.

A MOBILE user can easily switch to SMARTMOBILE because the executable programs for the models in both environments are similar. In the validated environment, the template syntax should be used. The names of transmission elements are the same aside from the preceding letter `T`. The methods of the classes have the same names, too. Only the solvers are, of course, different, although they follow the same naming conventions.

4.2 Examples

First, we give an example of the guaranteed simulation of kinematics of a five arm manipulator, the system defined in detail in (Traczinski, 2006). The modeling of the system itself can be enhanced in SMARTMOBILE by using so called sloppy joints (Traczinski, 2006) instead of usual revolute ones. In the transmission element responsible for the modeling of the

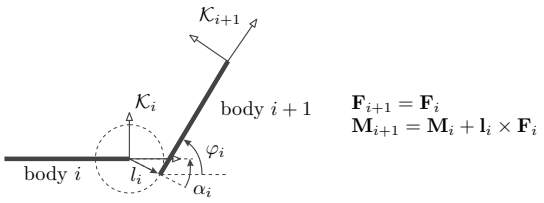


Figure 2: A sloppy joint.

Table 1: Widths of the position enclosures.

	x	y	CPU (s)
`TMoInterval`	1.047	1.041	0.02
`TMoTaylorModel`	0.163	0.290	0.14

sloppy joint, it is no longer assumed that the joint connects the rotational axes of two bodies exactly concentrically. Instead, the relative distance between the axes is supposed to be within a specific (small) range. Two additional parameters are necessary to describe the sloppy joint (cf. Figure 2): radius $l_i \in [0; l_{max}]$ and the the relative orientation angle $\alpha_i \in [0; 2\pi)$ (the parameter φ_i that describes the relative orientation between two connected bodies is the same both for the sloppy and the usual joint).

The considered system consists of five arms of the lengths $l_0 = 6.5$m, $l_1 = l_2 = l_3 = 4$m, and $l_4 = 3.5$m, each with the uncertainty of $\pm 1\%$. The arms are connected with five sloppy joints, for which $l_{max} = 2$mm and initial angle constellation is $\varphi_0 = 60^\circ$, $\varphi_1 = \varphi_2 = \varphi_3 = -20^\circ$, and $\varphi_4 = -30^\circ$. Each of these angles has an uncertainty of $\pm 0.1^\circ$.

In Figure 3, left, the (abridged) SMARTMOBILE model of the manipulator is shown (the system geometry described above is omitted). First, all the necessary coordinate frames are defined with the help of the array `K`. Analogously, the required rigid links `L` and sloppy joints `R`, with which arms and their connections are modeled, are declared. They are defined later inside the `for` loop. The array `l` characterizes the lengths of the rigid links, and `phi` is used to define the angles of sloppy joints. All elements are assembled into one system using the element `manipulator`. By calling the method `doMotion()` for this element, we can obtain the position of the tip of the manipulator, which equals the rotational matrix `R` multiplied by the translational vector `r`, both stored in the last coordinate frame `K[10]`.

We simulate kinematics with the help of intervals and Taylor models. That is, the placeholder `type` is either `TMoInterval` or `TMoTaylorModel`. Both position enclosures are shown in Figure 3, right. Note that enclosures obtained with intervals are wider than whose obtained with Taylor models (cf. also Table 1, where the widths of the corresponding intervals are shown). Taylor models are bounded by intervals to provide a comparison. Numbers are rounded up to the third digit after the decimal point. CPU times are measured on a Pentium 4, 3.0 GHz PC under CYGWIN.

The statistic analysis of the same system with the help of the Monte-Carlo method (Metropolis and Ulam, 1949) carried out in (Hörsken, 2003) shows

Table 2: Performance of `TMoAWAIntegrator`, `TMoRiOTIntegrator`, and `TMoValenciaIntegrator`. for the double pendulum over the time interval $[0; 0.4]$.

Strategy	`AWA`	`RiOT`	`Valencia`
Break-down	0.424	0.820	0.504
CPU time	1248	9312	294

that the results, especially those for Taylor models, are acceptable. Although the set of all possible positions obtained statistically with the help of 50,000 simulations (solid black area in Figure 3) is not as large as even the rectangle obtained with `TMoTaylorModel`, there might exist parameter constellations which lead to the results from this rectangle. Besides, statistical simulations require a lot of CPU time, which is not the case with SMARTMOBILE. Additionally, the results are proven to be correct there through the use of validated methods.

The next example is the double pendulum with an uncertain initial angle of the first joint from (Auer et al., 2006a). The lengths of both massless arms of the pendulum are equal to 1m and the two point masses amount to 1kg each with the gravitational constant $g = 9.81 \frac{\mathrm{m}}{\mathrm{s}^2}$ The initial values for angles (specified in rad) and angular velocities (in $\frac{\mathrm{rad}}{\mathrm{s}}$) are given as

$$\left[\frac{3\pi}{4} \quad -\frac{11\pi}{20} \quad 0.43 \quad 0.67\right]^T ,$$

where the initial angle of the first joint has an uncertainty of $\pm 1\%$ of its nominal value.

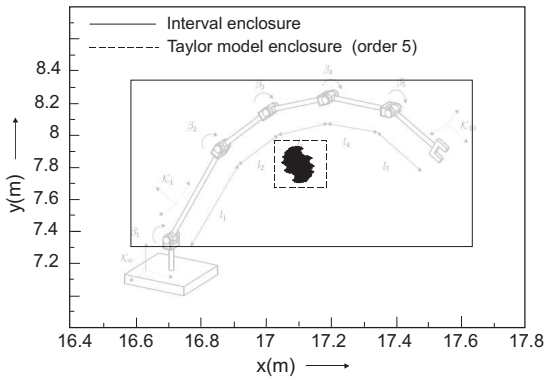
The interval enclosures of the two angles β_1 and β_2 of the double pendulum are shown for identical time intervals in Figure 4. Besides, Table 2 summarizes the results. The line "Break-down" contains the time in seconds after which the corresponding method no longer works. That is, the correctness of results cannot be guaranteed after that point. This happens here due to both the chaotic character of the considered system and the resulting overestimation. The last line indicates the CPU time (in seconds) which the solvers take to obtain the solution over the integration interval $[0; 0.4]$. Note that the CPU times are provided only as a rough reference since the solvers can be further optimized in this respect.

The use of `TMoValenciaIntegrator` improves both the tightness of the resulting enclosures and the CPU time in comparison to `TMoAWAIntegrator` for this example. Although `TMoRiOTIntegrator` breaks down much later than the both former solvers, it needs a lot of CPU time.

The double pendulum is a simple example of the opportunities that SMARTMOBILE offers for dy-

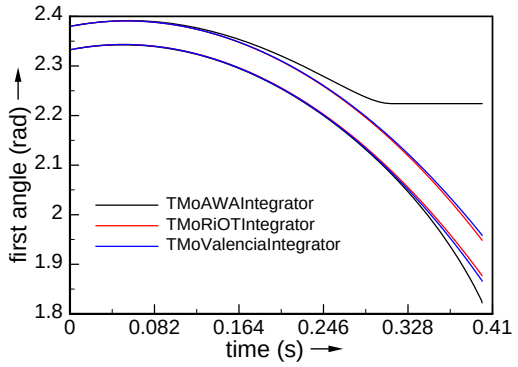
```
TMoFrame<type>     *K=new TMoFrame<type>[2*5+1];
TMoAngularVariable<type> *phi=new
                    TMoAngularVariable<type>[5];
TMoVector<type>       *l=new TMoVector<type>[5];
TMoSlacknessJoint<type> *R=new
                     TMoSlacknessJoint<type>[5];
TMoRigidLink<type> *L=new TMoRigidLink<type>[5];
TMoMapChain<type>                   manipulator;
double l_max;
for(int i=0;i<5;i++){
    TMoSlacknessJoint<type> joint(K[2*i],K[2*i+1],
                                  phi[i],zAxis,l_max);
    TMoRigidLink<type> link(K[2*i],K[2*i+1],l[i]);
    R[i]=joint; L[i]=link;
    Manipulator<<R[i]<<L[i];
}
Manipulator.doMotion();
cout<<"Position="<<K[10].R*K[10].r;
```

(a) SMARTMOBILE model.

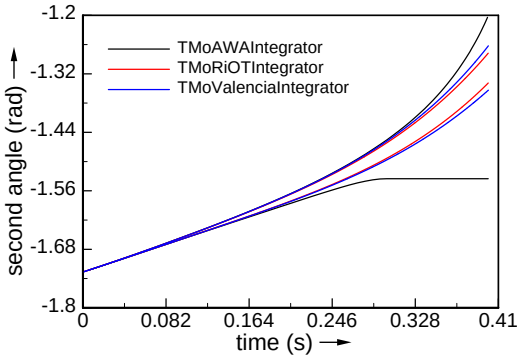


(b) Enclosures of the position of the tip.

Figure 3: Kinematics of the five arm manipulator with uncertain parameters.



(a) Enclosure of the first joint angle.



(b) Enclosure of the second joint angle.

Figure 4: Interval enclosures for the first and second state variable of the double pendulum.

namic simulation. More close-to-life examples are treated in (Auer et al., 2006b), (Auer, 2006), (Auer et al., 2004).

At last, consider the previous example once again but without the uncertainty in β_1. To find equilibrium states of this system, we use the basic data type `MoFInterval` instead of `TMoInterval` and apply `MoIGradientStaticEquilibriumFinder` to the element `manipulator` instead of using `TMoMechanicalSystem` and an integrator. All four possible equilibria (stable and unstable) are found by the validated solver `MoIGradientStaticEquilibriumFinder` in the starting interval $[-3.15;1]$ for all coordinates (shown rounded up to the third digit after the decimal point):

```
1: ([-3.142;-3.142];[-3.142;-3.142])
2: ([-3.142;-3.142];[-0.000; 0.000])
3: ([-0.000; 0.000];[-3.142;-3.142])
4: ([-0.000; 0.000];[-0.000; 0.000])
```

Since we do not have any uncertainties in the model, the intervals obtained are very close to point intervals, that is, $\beta_i \approx \overline{\beta}_i$. The difference is noticeable only after the 12-th digit after the decimal point. However, if the same problem is modeled using the non-verified model in MOBILE, only one (unstable) equilibrium state $[\beta_1,\beta_2]$ =`[3.142;-3.142]` is obtained (using the identical initial guess).

5 CONCLUSIONS

In this paper, we presented a recently developed tool SMARTMOBILE for guaranteed modeling and simu-

lation of kinematics and dynamic of mechanical systems. With its help, the behavior of different classes of systems including non-autonomous and closed-loop ones can be obtained with the guarantee of correctness, the option which is not given in tools based on floating point arithmetics. Besides, the uncertainty in parameters can be taken into account in a natural way. Moreover, SMARTMOBILE is flexible and allows the user to choose the kind of underlying arithmetics according to the task at hand. The functionality of the tool was demonstrated using three examples.

The main directions of the future development will include enhancement of validated options for modeling and simulation of closed-loop systems in SMARTMOBILE as well as integration of further verified solvers into its core.

REFERENCES

Auer, E. (2006). Interval Modeling of Dynamics for Multibody Systems. In *Journal of Computational and Applied Mathematics*. Elsevier. Online.

Auer, E., Kecskeméthy, A., Tändl, M., and Traczinski, H. (2004). Interval Algorithms in Modeling of Multibody Systems. In Alt, R., Frommer, A., Kearfott, R., and Luther, W., editors, *LNCS 2991: Numerical Software with Result Verification*, pages 132 – 159. Springer, Berlin Heidelberg New York.

Auer, E., Rauh, A., Hofer, E. P., and Luther, W. (2006a). Validated Modeling of Mechanical Systems with SMARTMOBILE: Improvement of Performance by VALENCIA-IVP. In *Proc. of Dagstuhl Seminar 06021: Reliable Implementation of Real Number Algorithms: Theory and Practice*, Lecture Notes in Computer Science. To appear.

Auer, E., Tändl, M., Strobach, D., and Kecskeméthy, A. (2006b). Toward validating a simplified muscle activation model in SMARTMOBILE. In *Proceedings of SCAN 2006*. submitted.

Berz, M. and Makino, K. (1998). Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models. *Reliable Computing*, 4:361–369.

Berz, M. and Makino, K. (2002). COSY INFINITY Version 8.1. User's guide and reference manual. Technical Report MSU HEP 20704, Michigan State University.

de Figueiredo, L. H. and Stolfi, J. (2004). Affine arithmetic: concepts and applications. *Numerical Algorithms*, 37:147158.

Griewank, A. (2000). *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM.

Griewank, A., Juedes, D., and Utke, J. (1996). ADOL–C, a package for the automatic differentiation of algorithms written in C/C++. *ACM Trans. Math. Software*, 22(2):131–167.

Hammer, R., Hocks, M., Kulisch, U., and Ratz, D. (1995). *C++ toolbox for verified computing I - Basic Numerical Problems*. Springer-Verlag, Heidelberg and New York.

Hörsken, C. (2003). *Methoden zur rechnergestützten Toleranzanalyse in Computer Aided Design und Mehrkörpersystemen*. PhD thesis, Univerisity of Duisburg-Essen.

Huckle, T. (2005). Collection of software bugs. `www5.in.tum.de/~huckle/bugse.html`.

Kecskeméthy, A. (1993). *Objektorientierte Modellierung der Dynamik von Mehrkörpersystemen mit Hilfe von Übertragungselementen*. PhD thesis, Gerhard Mercator Universität Duisburg.

Kecskeméthy, A. and Hiller, M. (1994). An object-oriented approach for an effective formulation of multibody dynamics. *CMAME*, 115:287–314.

Klatte, R., Kulisch, U., Wiethoff, A., Lawo, C., and Rauch, M. (1993). *C–XSC: A C++ Class Library for Extended Scientific Computing*. Springer-Verlag, Berlin Heidelberg.

Knüppel, O. (1994). PROFIL/BIAS — a fast interval library. *Computing*, 53:277–287.

Lerch, M., Tischler, G., Wolff von Gudenberg, J., Hofschuster, W., and Krämer, W. (2001). The Interval Library `filib++ 2.0` : Design, Features and Sample Programs. Technical Report 2001/4, Wissenschaftliches Rechnen / Softwaretechnologie, Bergische Universität GH Wuppertal.

Lohner, R. (1988). *Einschließung der Lösung gewönlicher Anfangs- und Randwertaufgaben und Anwendungen*. PhD thesis, Universität Karlsruhe.

Lohner, R. (2001). On the ubiquity of the wrapping effect in the computation of the error bounds. In Kulisch, U., Lohner, R., and Facius, A., editors, *Perspectives on Enclosure Methods*, pages 201–217. Springer Wien New York.

Metropolis, N. and Ulam, S. (1949). The monte carlo method. *Journal of the American Statistic Association*, 44:335–341.

Moore, R. E. (1966). *Interval Analysis*. Prentice-Hall, New York.

Nedialkov, N. S. (1999). *Computing rigorous bounds on the solution of an initial value problem for an ordinary differential equation*. PhD thesis, University of Toronto.

Nedialkov, N. S. (2002). *The design and implementation of an object-oriented validated ODE solver*. Kluwer Academic Publishers.

Neumaier, A. (2002). Taylor forms — use and limits. *Reliable Computing*, 9:43–79.

Stauning, O. and Bendtsen, C. (2005). Fadbad++. Web page `http://www2.imm.dtu.dk/~km/FADBAD/`.

Traczinski, H. (2006). *Integration von Algorithmen und Datentypen zur validierten Mehrkrpersimulation in MOBILE*. PhD thesis, Univerisity of Duisburg-Essen.