Contents lists available at ScienceDirect

Comptes Rendus Mecanique

www.sciencedirect.com

# Interactive virtual prototyping of a mechanical system considering the environment effect. Part 2: Simulation quality

Zheng Wang

*Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China*

A R T I C L E   I N F O

*Article history:*
Received 7 October 2010
Accepted after revision 10 June 2011
Available online 18 July 2011

*Keywords:*
Dynamical systems
Multi-body
Stability
Accuracy
Interactivity

A B S T R A C T

Perhaps the three most important issues in numerical simulation of mechanical systems are: (1) how well do the multi-body systems represent a physical system of interest; (2) how efficient is the simulation; and (3) how accurate is the simulation. With the advances in computational mechanics, the efficiency of multi-body simulations is steadily improving. Indeed, analysts are increasingly envisioning real-time simulation. With these improvements in computation and efficiency, the modeling of physical systems is also improving through the ability to have more comprehensive modeling. The issue of accuracy, however, remains. Generally, the complexity of multi-body system dynamics leaves the analyst without firm assurance about the numerical accuracy short of experimental verification or intuitive reasonableness of the results. In the second part of the paper we present some methods and experiments to clarify simulation quality of the basic model described in the first article.

## 1. Introduction

At present modeling represents the basic scientific tool applied for pure-theoretical and practical purposes. With simulation, one can study a problem that is not often subject to direct experiment. It allows researchers to build system models, based on real world data collected from many studies, and then test a model under conditions when time and materials are not available. It is a powerful tool for designing and analyzing complex and dynamic systems, for predicting their behavior under different conditions on time scale.

In the earlier article [1], we have built a set of dynamic models for virtual prototyping of mechanical system considering environment effects. This elementary simulating tool will be used to design, identify and control robots and it is a powerful technique to improve quality and productivity of researchers work. Using the software environment, one can visually design and model systems by means of simulating separate parts of these systems and investigating its behavior under conditions that are close to real ones. However, how can we verify validity of the simulation tool? Defining a criterion of simulation quality as a combination of stability and accuracy, real-time performance and reconfigurability analysis, we will give a description in detail as follows.

In Section 2, an overview of related works is given. The stability and accuracy analysis in Section 3, the real-time performance analysis in Section 4 will be seen in order. Then we developed a numerical experiment for reconfigurability and a real robot in Sections 5 and 6. Finally, the conclusion is given in Section 7.

*E-mail address:* flair@126.com.

1631-0721/$ – see front matter © 2011 Académie des sciences. Published by Elsevier Masson SAS. All rights reserved.
doi:10.1016/j.crme.2011.06.003

## 2. Related works

### 2.1. Stability and accuracy

Multi-body systems are frequently modeled as constrained systems, and the arising governing equations incorporate the closing constraint equations at the acceleration level. One consequence of accumulation of integration truncation errors is the phenomenon of violation of the lower-order constraint equations by the numerical solutions to the governing equations. The constraint drift usually tends to increase in time and may spoil reliable accuracy and stability of the simulation results.

Several reliable and numerically efficient stabilization and projection methods have been proposed to compute a numerical solution for the equations of motion of constrained multi-body systems, formulated either as DAEs or resolved to ODEs, which remains on or close to the constraint manifold. Baumgarte's Constraint Violation Stabilization (CVS) method proposed in [2] and then developed in numerous contributions [3–6], is probably the most popular technique of this type that stabilizes the constraint drift to some bounds. Other useful approaches are based on simultaneous consideration of more than one of the constraint equations [7–10]. Then, in the Constraint Violation Elimination (CVE) schemes, originally introduced in [11] and then developed in [12–14], the current constraint violations transformed into appropriate state corrections that assure constraint satisfaction. A variety of other/similar methods for constraint violation suppression in the numerical simulation of constrained mechanical system have also been developed, and a comprehensive review of them is provided in the survey papers [15,16]. Braun and Goldfarb [17] proposed recently yet another constraint violation stabilization technique, in which violations are used as correction terms in the modified constraint equations at velocity and acceleration levels.

### 2.2. Real-time performance and interactivity

The challenge in including dynamic simulation in applications is clearly that computations should be (at least) as fast as real time, especially when dealing with low-level control, graphics render and HCI (human computer interaction). The development of efficient dynamic simulation algorithms for tree-structured kinematic chains and the fast growth of computational power in general purpose machines has helped reaching the goal of simulating rather complex structures in real time [18–21], such as open chains of up to some tens of links. However impressive these advancements have been, they are not yet comparable to what has been achieved in time span by graphic software and hardware accelerators. Many complex dynamic systems remain beyond real-time simulation capabilities of present-day software and hardware.

Generally speaking, an important factor influencing real-time performance lies in the integration method. Real-time simulator or hardware-in-the-loop test facilities have special demands on the integration methods utilized [22,23]. The classical explicit Euler method suffers from numerical instability in the application to stiff systems, such as vehicle models, with stiff suspension elements or strongly damped components. To avoid the iterative solution of nonlinear equations that is typical of implicit time integration methods for stiff systems, the linear-implicit Euler method that may be interpreted as an implementation of the implicit Euler method with an a priori fixed number of Newton steps in the iterative solution of the arising systems of nonlinear equations [24].

Another method improving performance should be HPC (high performance computing) or parallel computing. Tasora presented a large-scale parallel multi-body dynamics algorithm for graphical processing units [25]. Their algorithm is based on a cone complementarity problem for the simulation of frictional contact dynamics. Their approach is suited for more than one million rigid bodies. Two open source examples are the famous Open Dynamics Engine (ODE) [26] and the OpenTissue library [27]. However, currently none of them offers massively MPI parallel rigid body dynamics. Fleissner focus on parallel simulation of particle-based discretization methods [28], such as the discrete element method or smoothed particle hydrodynamics, instead of rigid body dynamics. Although their approach offers the option to run large-scale particle simulations, they are only considering point masses instead of fully resolved rigid bodies.

## 3. Stability and accuracy verification

In the absence of roundoff, index reduction and integration of the resulting ODE would be perfectly adequate. Unfortunately, numerical drift of the solution from the invariant manifold $\Phi(q,t)$ requires alternative schemes. One such scheme is based on coordinate partitioning. Here the coordinates are split into independent and dependent sets. Finding a robust detection method for changing the parameterization is the most challenging aspect of the scheme. This scheme is not suitable for real-time HIL simulations, an important consideration for our solver as it supports code generation for real-time deployment. Other schemes are based on stabilization and coordinate projection. Stabilization involves the addition of extra terms to the equation of motion. These terms vanish on the manifold $\Phi(q,t)=0$, but have the effect of making the solution asymptotically attractive to the manifold.

If the solution does drift of the manifold, it is ultimately attracted back onto it, although there are no predefined bounds on the extent of the drift. The most popular stabilization technique is Baumgarte stabilization. Its simplicity has made it a popular choice in engineering applications. But Baumgarte stabilization requires parameterization, and there is no known generic procedure for choosing these parameters to make the stabilization robust. Indeed the choice of suitable parameters depends on the discretization scheme used to integrate the equations of motion, a serious practical limitation.

Coordinate projection involves the numerical discretization of the ODE. At the end of the discretization step, the solution is projected onto the invariant manifold. For example, the effect of coordinate projection on step size selection, event location, and order of convergence are all considered. Our simulation codes can all be adapted to allow for coordinate projection without compromising accuracy or efficiency. A fundamental change to our package has been the addition of a projection method to the ODE suite that can be called once the discretized solution has been updated following the acceptance of a successful step based on the error estimates. The scheme works for the one-step Runge–Kutta formulas, as well as for variable-step variable-order codes. Finally, the standard theory for convergence in BDF codes like ode15s is still applicable when projection is carried out in this way.

An important property of the solver is its ability to localize and detect events using discontinuity locking (to ensure that the integrators see a continuous vector field) and its ability to output solutions at any time in the interval of integration using continuous extension formulae. Both of these features are affected by projection. The solver allows users to refine the output from the solvers using highly efficient interpolation schemes, and these interpolated outputs typically satisfy the invariants to accuracy comparable with the accuracy of the numerical solution. The interpolated values are further projected to ensure that the invariants are satisfied.

To ensure that events are located correctly, the outputs of the interpolants are projected before sampling the switching functions. This makes event location more expensive, but is necessary for robust event detection. For example, the projection approach is appropriate for a one-step method used to compute an approximate solution at time $t_{n+1}$ from a solution at $t_n$. The step size is $h$, and $t_{n+1} = t_n + h$. The solution takes the form

$$\begin{bmatrix} q^*_{n+1} \\ \dot{q}^*_{n+1} \end{bmatrix} = h\Phi(t_n, q_n, \dot{q}_n) \tag{1}$$

If we consider the nonlinear position constraint $\Phi(q,t) = 0$, the predicted position variables $q^*_{n+1}$ are projected onto the closest point on the manifold, denoted by $q_{n+1}$. Linearizing the constraints about the predicted value $q^*_{n+1}$ gives

$$0 = \Phi(t_{n+1}, q_{n+1}) = \Phi\big(t_{n+1}, q^*_{n+1}\big) + G\big(t_{n+1}, q^*_{n+1}\big)\big(q_{n+1}, q^*_{n+1}\big) + o\big(\|q_{n+1} - q^*_{n+1}\|^2\big) \tag{2}$$

To leading order, the projected solution $q_{n+1}$ is obtained from solving

$$G\delta = \Phi \tag{3}$$

$$q_{n+1} = q^*_{n+1} - \delta \tag{4}$$

If the Euclidean norm is used,

$$\delta = G^T\big(GG^T\big)^{-1}\Phi \tag{5}$$

The projection process is repeated, and so is referred to as sequential projection, until a suitable level of convergence is attained. This is much more efficient than forming the full set of Karush–Kuhn–Tucker (KKT) equations to determine the exact minimization on the manifold and is easily motivated by the fact that error control has made the predicted solution $q^*_{n+1}$ close to the manifold already. In practical applications, it is advantageous to perform the projection in a weighted norm. The projection weights reflect the weights used in the error control of the ODE solver. For an example, we give a car engine represented with a slider crank mechanism as in Fig. 1(a), and a comparison result of different methods as in Fig. 1(b).

## 4. Real-time performance verification

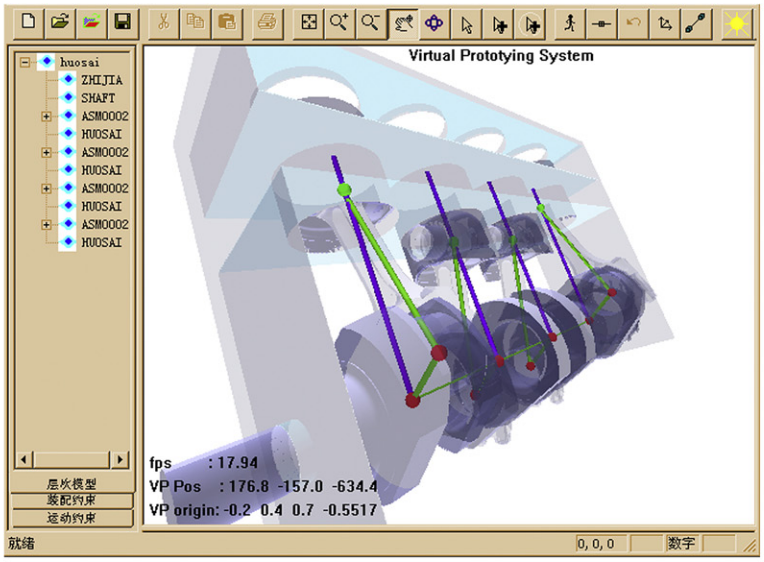
### 4.1. Index reduction strategy

In implicit time integration methods, the consideration of constraints is straightforward. Furthermore, there are implicit methods with excellent stability properties for stiff equations. However, for real-time applications implicit methods are not suitable since systems of nonlinear equations have to be solved iteratively in each time step. Similar stability properties may be achieved by linear-implicit methods without any iterative algorithms. For constrained systems, we cannot expect that a non-iterative method solves nonlinear constraints exactly. Now, we extend the linear-implicit Euler method to a non-iterative method for constrained systems and prove that the error in the constraints $\Phi(q) = 0$ is bounded by $\Phi(q_n) \leqslant kh^r$ with $r \geqslant 2$ and a constant $k$ that is independent of the length of the time interval.

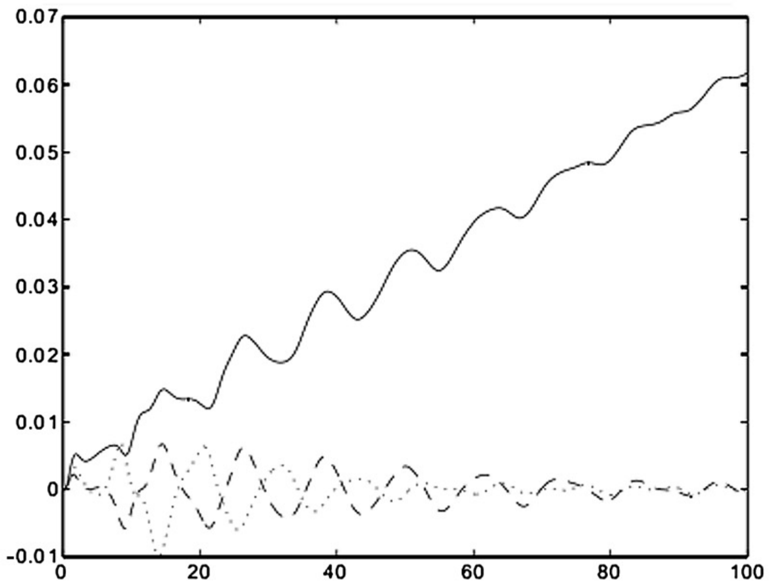
In the earlier article [1], the equations of motion (1) are given by a differential-algebraic system of index-3. To avoid numerical problems which make the direct time integration of index-3 systems difficult, the system is transformed to a new system with lower index by differentiation of the constraints [29].

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = F_e + \Phi_q^T F_c \quad \Leftrightarrow \quad M(q)\ddot{q} = f(q,\dot{q}) + \lambda\Phi(q)$$

and



(a) Mechanism skeletal topology of slider and crank, and slider is a cut-off joint.



(b) Error in the position of the slider: without stabilization (solid line), with coordinate projected stabilization (dashed line) and with Baumgarte's stabilization (dotted line).

**Fig. 1.** Stabilization comparison for a slider crank mechanism.

$$M(q)\ddot{q} = f(q,\dot{q}) + \lambda\Phi(q) \quad \Leftrightarrow \quad \begin{cases} \dot{q} = u \\ \dot{u} = \tilde{f}(q,u) = M^{-1}(q)f(q,u) \end{cases}$$

we will have

$$\begin{aligned} 0 &= \Phi(q) \\ 0 &= \frac{\mathrm{d}\Phi(q)}{\mathrm{d}t} = \Phi_q(q)\dot{q} = G(q)u \\ 0 &= \frac{\mathrm{d}^2\Phi(q)}{\mathrm{d}t^2} = \Phi_{qq}(q)(u,u) + G(q)\dot{u} \end{aligned} \tag{6}$$

If we substitute the original constraints on position level by their second derivative on acceleration level we get the so-called index-1 formulation. Then we can calculate the Lagrange multipliers $\lambda$ by solving the linear system of equations

$$\lambda = \left(G(q)M^{-1}G^T(q)\right)^{-1}\left(\Phi_{qq}(q)(u,u) + G(q)M^{-1}f(q,u)\right) \tag{7}$$

Accordingly we get the index-2 formulation by using the first derivative (6) on velocity level of the constraints. Because of the special structure of the equations of motion we can insert the formula for the integration step of $u_{n+1}$, in (6) and choose $\lambda_n$ so that $G(q_{n+1})u_{n+1} = 0$ holds:

$$q_{n+1} = q_n + hu_n \tag{8}$$

$$\begin{pmatrix} M - hJ_u & G^T(q_n) \\ G(q_{n+1}) & 0 \end{pmatrix}\begin{pmatrix} u_{n+1} - u_n \\ \lambda_n \end{pmatrix} = h\begin{pmatrix} f + hJ_q u_n \\ -G(q_{n+1})u_n \end{pmatrix} \tag{9}$$

Here, $J_q$ and $J_u$ denote the lower blocks in the $2 \times 2$ block Jacobin matrix $J$. The main advantages of this linear-implicit Euler method are that the velocity constraints (6) are solved exactly and there is no need for the second derivative of $\Phi(q)$. It may be considered as a drawback of this approach that the matrix of (9) is no longer symmetric and the order of the calculation of $q_{n+1}$ and $u_{n+1}$ is fixed.

### *4.2. HPC strategy*

Currently, high-end GPUs offer floating-point parallel computing power close to one Teraflop, thus exceeding those of multi-core CPUs. This computational resource, usually devoted to the execution of pixel shading fragments for the rendering of 3D visualization, can be also exploited for scientific computation.

We implemented our code on graphics board of the 9800 GX2 family, from NVIDIA. Each board features two GPU processors, for a total of 256 streaming processors and capable of running 24,576 live threads. The processed data resides in the 2 GB of DDR3 device memory. The basic idea is that, at each simulation step, the CPU uploads data into the GPU memory, launches a kernel to be performed simultaneously on many parallel GPU threads, and gathers the results of the computations by downloading select portions of the GPU memory back into the host RAM. Out of the entire computational time, the time slice spent on the CPU should be as small as possible to exploit the scalable nature of the GPU parallelism.

For the problem at hand, not all of the multi-body simulation has been ported on the GPU. In particular, this is the case of the collision detection engine, which is still executed on the CPU and becomes the bottleneck of the entire simulation. Nonetheless, the proposed algorithm fits well into the GPU multithreaded model because the computation can be split in multiple threads each acting on a single contact or kinematic constraint.

We built the data structures on the GPU as large arrays (*buffers*) to match the execution model associated with NVIDIA's CUDA. Specifically, threads are grouped in rectangular thread blocks, and thread blocks are arranged in rectangular grids. Four main buffers are used: the contacts buffer, the constraints buffer, the reduction buffer, and the bodies' buffer.

When designing the data structures of these buffers, special care should be paid to minimize the memory overhead caused by repeated transfers of large data structures. Moreover, data structures should be organized to exploit fast GPU coalesced memory access to fetch data for all parallel threads in a warp, which is a set of 32 threads all running simultaneously in parallel. Provided that bytes are contiguous and that the $k$th thread accesses the $k$th element in the data structure, up to 512 bytes can be fetched in one operation by a warp of threads. Failing to perform coalesced memory access may slow the kernel significantly.

Numerical experiments show that for high memory throughput, it is better to pad the data into a four-float width structure even at the cost of wasting memory space when several entries end up not being used. Also, the variables in the data structures are organized in a way that minimizes the number of fetch and store operations. This approach maximizes the arithmetic intensity of the kernel code, as recommended by the CUDA development guidelines.

The components of mechanical system are distributed according to their location in the simulation domain in order to keep the communication between fluid solver and rigid body dynamics solver minimal. In other words, for all objects that are contained in the local sub-domain of one process, the solvers can communicate with each other by procedure calls. Only bodies on process interfaces need to be synchronized with the directly neighboring processes. Due to the local communication and the strictly local collision treatment, this approach supports large scale simulations of several million bodies immersed in the fluid. But still, compared to methods with point masses, the parallelization is not trivial because of the geometrical extent of the bodies. Each body is managed by the process where the center of mass belongs to. Bodies which are contained in the sub-domain of more than one process need to be synchronized between the processes.

In contrast to the nonparallel algorithm, the parallel version contains a total of four communication steps to handle the distributed computation. The rest of the algorithm remains unchanged in comparison to the nonparallel formulation. Instead of immediately starting with the first position and velocity half-step for every rigid body, the external forces are synchronized. Before the generation of all contacts and the resulting constraints, remote rigid bodies are updated and bodies entering the local domain for the first time are notified. When all constraints have been set up, these are exchanged between the processes. At the end, remote rigid bodies are updated again and bodies entering the local domain for the first time are notified. We test GPU and CPU performance with a chain increasing bodies from 10 to 250 as in Fig. 2(a), and the result can be seen in Fig. 2(b).

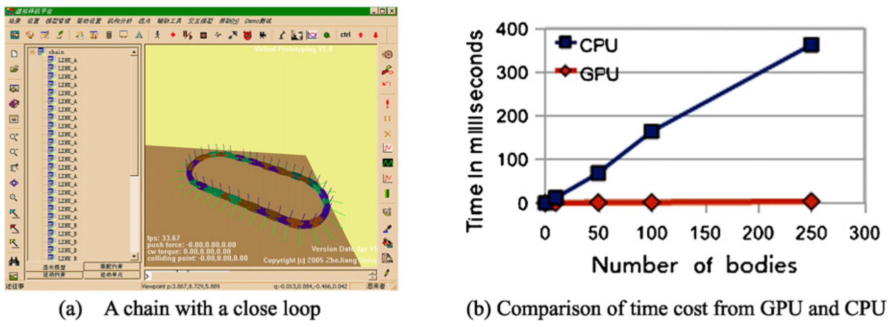


**Fig. 2.** Test GPU and CPU performance with a chain.

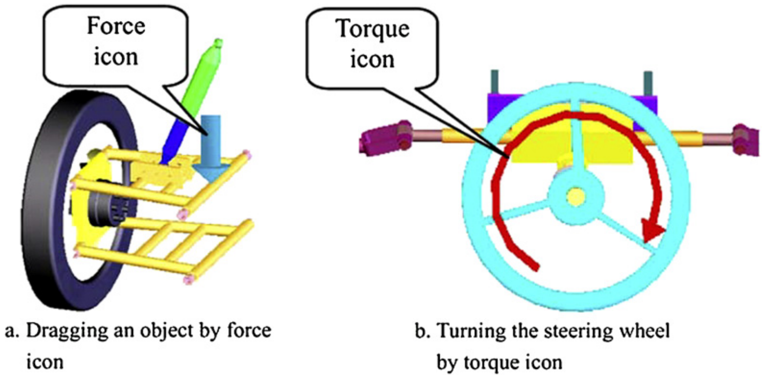

Force icon
Torque icon
a. Dragging an object by force icon
b. Turning the steering wheel by torque icon

**Fig. 3.** Force and torque icon.

## 5. Reconfigurability verification with an interactive numerical experiment

### 5.1. Reconfigurability of the package

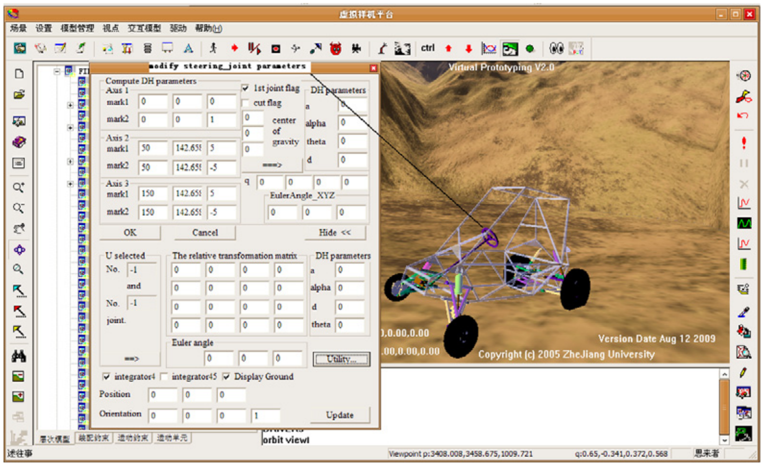
Reconfigurability and openness are features already recognized by many as essential in the development of advanced control algorithms. Not only is it important to have easy access to the system at all levels (e.g. from high-level supervisory control all the way down to fast servo loops at the lowest level), but it is a necessity to have open control architectures where software modules can be modified and exteroceptive sensors like force/torque sensors and vision systems can be easily integrated. Reconfigurability should also be reflected when more fundamental changes to the controller architecture are required, in the necessity of quickly being able to make modifications in the original design and verify the effect of these modifications on the system. In other words, the user should be able to quickly modify the structure of the control without having to alter the simulation system itself.

However, it is difficult for current commercial programs such as ADAMS to allow users' reconfiguration of some interested parameters in any time step of simulating process. To overcome this problem, we embed the control interface into simulation loop.

In our simulator, a user can select any component of the virtual mechanism to implement user-defined online control regardless of run-time state. Four types of simulation manipulator are provided to control the simulation of mechanical system:

(1) *Adding force/torque/motion manipulator*. Force, torque and motion are presented by corresponding 3D icons. Users can interactively apply force and torque to simulation model by altering the position, orientation, length and parameters of the icons. The force icon is a 3D arrow entity, whose head or tail is mapped to the contact points of push or drag actions. Push force is along the direction of force arrow (see Fig. 3(a)), and the opposite direction is drag force. Users can interactively modify the length of arrow so as to change the value of the force. The torque icon is a 3D arc arrow (see Fig. 3(b)). The center axis of the arc is the center axis of the applied torque, and the radius of the arc expresses the value of the torque. The motion icon is a 3D sphere entity near the applied joint.

(2) *Dragging manipulator*. Users can directly pick and drag the components in the simulation system. By using 3D mouse, the operator can pick the component and drive it directly. In this way, the mechanical system can be seen as a set of geometrical objects with dynamical constrained properties. When the user modifies the position of one of them, all the others have to resolve the appropriate position and orientation according to the constraints. This task must be done in real time in order to give the user an acceptable visual feedback.


虚拟样机平台
场景 设置 模型管理 视点 交互模型 驱动 帮助(H)
Virtual Prototyping V2.0
Version Date Aug 12 2009
Copyright (c) 2005 ZheJiang University
orbit view!
Viewpoint p:3408.008,3458.675,1009.721
q:0.65,-0.341,0.372,0.568

modify steering_joint parameters

Compute DH parameters

| | | | |
|---|---|---|---|
| Axis 1 | | | |
| mark1 | 0 | 0 | 0 |
| mark2 | 0 | 0 | 1 |
| Axis 2 | | | |
| mark1 | 50 | 142.65 | 5 |
| mark2 | 50 | 142.65 | -5 |
| Axis 3 | | | |
| mark1 | 150 | 142.65 | 5 |
| mark2 | 150 | 142.65 | -5 |

✓ 1st joint flag
cut flag
center of gravity: 0 0 0

DH parameters: a 0, alpha 0, theta 0, d 0

q: 0 0 0 0

EulerAngle_XYZ: 0 0 0

OK Cancel Hide <<

U selected: No. -1 and No. -1 joint.

The relative transformation matrix

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

DH parameters: a 0, alpha 0, d 0, theta 0

Euler angle: 0 0 0

Utility...

✓ integrator4 integrator45 ✓ Display Ground

Position: 0 0 0

Orientation: 0 0 0 1

Update

**Fig. 4.** Constraint modification manipulator.

(3) *Parameter modification manipulator*. Users can pick the force/torque/motion icons in the simulator and activate their corresponding parameter modification dialogs. Users can modify the parameters in these dialogs. By this method, it is easy for a designer to change how the simulation functions. For an instance, if a control should feel stiffer, the designer can change a spring rate or damping coefficient by adjusting parameters.

(4) *Constraint modification manipulator*. Users can interactively pick joints between components of dynamic model in the simulation system and modify the DOFs or parameters of interested joints (Fig. 4). For example, a cylindrical joint, which has one rotational degree and one translational degree, can be modified to be a revolute joint by reducing the translational degree. Users can also change properties of the joint, such as motion limit or friction of the joint.

Thus with the help of the simulation manipulator, the user can study if a specific product is well adapted to the task, and adjust the product and fluid environment parameters to optimize the task. We have found that the use of a user interface to simulate mechanical system in a parameterized fluid environment drastically improves the “feeling” of the simulation and presentation. In particular, the interface allows us to understand the behavior of a virtual prototyping in a special environment, and to find why a new designed product does not work as expected.

### 5.2. *Simulation with user-defined online control*

An example of the developed *MiniBaja* is shown in Fig. 5; we input a movement stimulus on a front wheel of the vehicle, and then the vehicle will be driven, at the same time the linear velocity, angular velocity, linear acceleration, angular acceleration, total force and torque about the interested object will be plotted at the bottom of simulation window.

The interaction with *MiniBaja* is achieved by mouse clicking on any component of the mechanism to drive, then adjusting input quantity on control panel or adjusting the generalized force/torque/motion icon based on tasks. As shown in Fig. 6, *MiniBaja* is driven by torque icon. Data analysis tools are also developed that will allow for a variety of data presentation and analysis options, which includes both the tracking of timings as well as the values of specific variables or combinations of variables. For example, the total force applied on a component may be constantly tracked.

To verify and validate our simulator, we have conducted a numerical experiment to compare the simulation result of ADAMS to ours. The same virtual vehicles were built and simulated in ADAMS and our package. The two simulations were started under the same initial condition. At the 10th second, an additional torque is interactively applied to the simulation model by torque icon in our simulator, and the same torque is applied by user-written subroutines in ADAMS. Fig. 7 shows the comparison of the change curves of driving torque during simulation. The red curve shows the simulation result in ADAMS and the blue one is in our simulator. Comparing the two curves, we can see that the simulation result in ours has good correlation with ADAMS and the tolerance is acceptable.

## 6. Practical experiments

To clarify the validity of our simulation package, some developers of our group created a methodology that guarantees close relations with real world systems, and continuous reality checks. The idea is rather simple. Whenever a new component, is added to the package, one or more experiments should be designed in order to assess the accuracy of the simulation. These validation experiments will be performed twice, once in the real system, and once in a simulation setting resembling as much as possible the real system used. After the two experiments, results should be quantitatively compared. The methodology was lifted at a higher level, and more complex units were validated: for example human–robot interfaces.

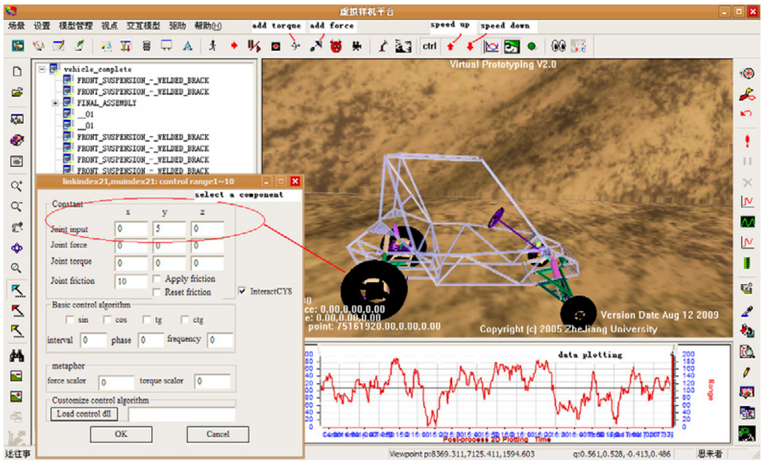
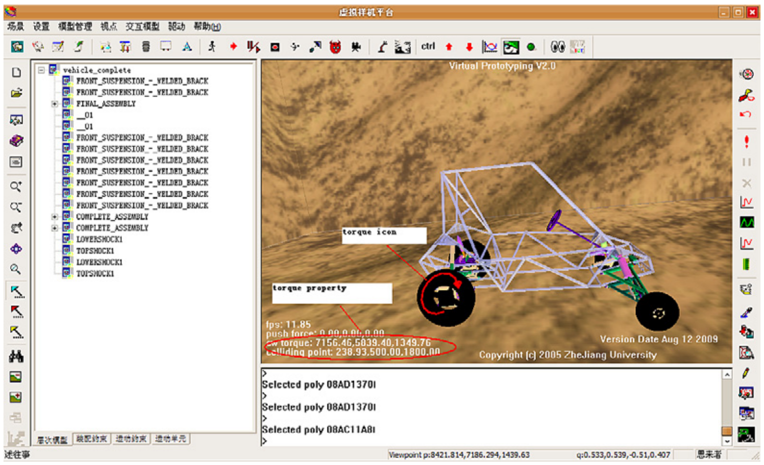


**Fig. 5.** Apply a movement stimulus on a front wheel.



fps: 11.85
push force: 0.00,0.00,0.00
w torque: 7156.46,5039.40,1349.76
colliding point: 238.93,500.00,1800.00

Selected poly 08AD1370I
Selected poly 08AD1370I
Selected poly 08AC11A8I

**Fig. 6.** Steering virtual vehicle by the torque icon.

In order to exemplify, the procedure followed to validate the most recently added sensor, i.e. the GPS is presented (to design a realistic GPS simulation module within our package).

In order to validate the newly added sensor, a robot equipped with a GPS received was driven around the CASIA, as in Fig. 8(a). The robot was purposely driven in open areas far from buildings as well as in close proximity to them. During these runs, the number of satellites visible by the GPS receiver was logged, as well as the path returned by the reader. Next, the same experiment was performed in simulation. A model of the relevant part of the institute was developed, including appropriately scaled buildings. The simulated robot was then driven through the same path at the same simulated time of day (to experience the same positions of the NAVSTAR satellites), and the same information was logged and compared, as in Fig. 8(b). Figs. 9 and 10 show the results obtained.

In Fig. 9 there is a comparison between results obtained in simulation (dark paths) and with the real GPS sensor (bright paths). Paths provided by the GPS were overlaid to the appropriate map retrieved from Google Earth.

In Fig. 10(a) there is a comparison between the numbers of satellites tracked by the simulated GPS sensor (dark series) and by the real GPS sensor (bright series). Even though the simulated sensor almost always tracks one more satellite than the real one and exhibits a less jagged profile, the trends are clearly the same. In Fig. 10(b), we calculated the position of centers of mass corresponding to GPS signal excluding coordinate-$y$ for simplicity, and it is concluded that the simulated follows the real.

## 7. Conclusion

In view of the dynamics model presented in the first part of the paper, fatherly we clarify its simulation quality in some important respects including stability and accuracy, real-time performance and reconfigurability with interaction. These clarifications resorted to some theoretical, technical and numerical methods, as well as practical experiment. As for a most significant difference with ADAMS, some types of simulation manipulator are provided in our package to control the sim-

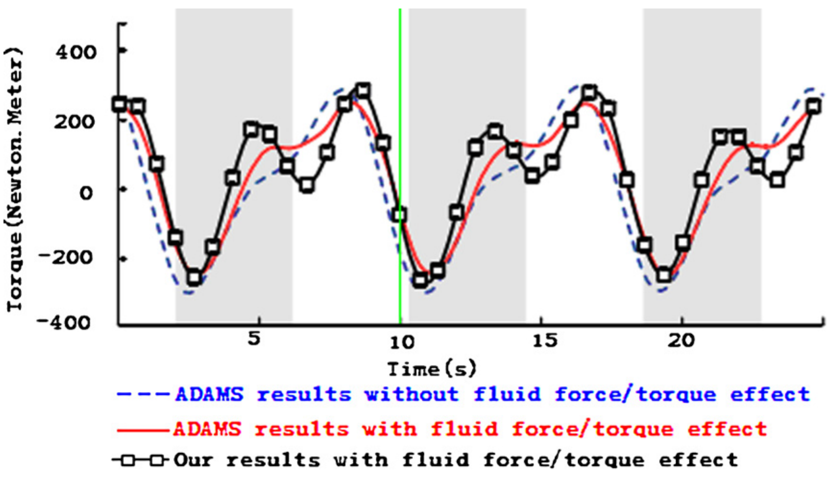


**Fig. 7.** Simulated driving torque in ADAMS and our simulator. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

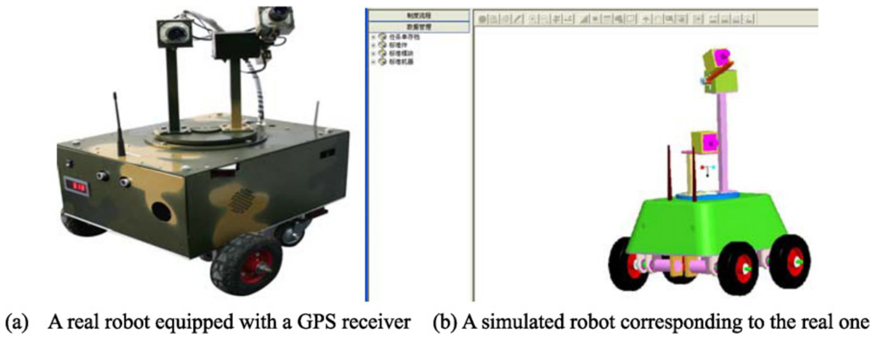

(a) A real robot equipped with a GPS receiver
(b) A simulated robot corresponding to the real one

**Fig. 8.** Real and simulated robot.

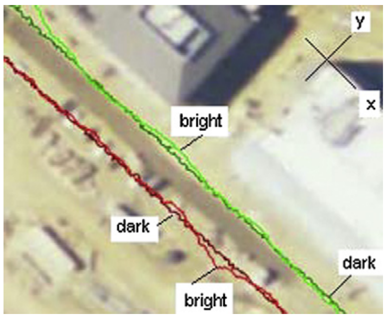


**Fig. 9.** Simulated and real path.

ulation of mechanical system interactively. With the real-time interaction, solution and visualization of simulation model, our package affords better support for designers to participate in the simulation. On the one hand, a numerical example is implemented in the dynamic simulation of a *MiniBaja* and the simulation result is compared with ADAMS. On the other hand, we design a real robot and a simulated robot respectively to verify the fidelity of our package to real system through a trend comparison.

With the above experiments and results, validation efforts convinced us otherwise skeptical about simulation that our package is indeed a useful tool to develop code to be eventually run on real robot systems, and that’s indeed a target we shall work on in a foreseeable future.

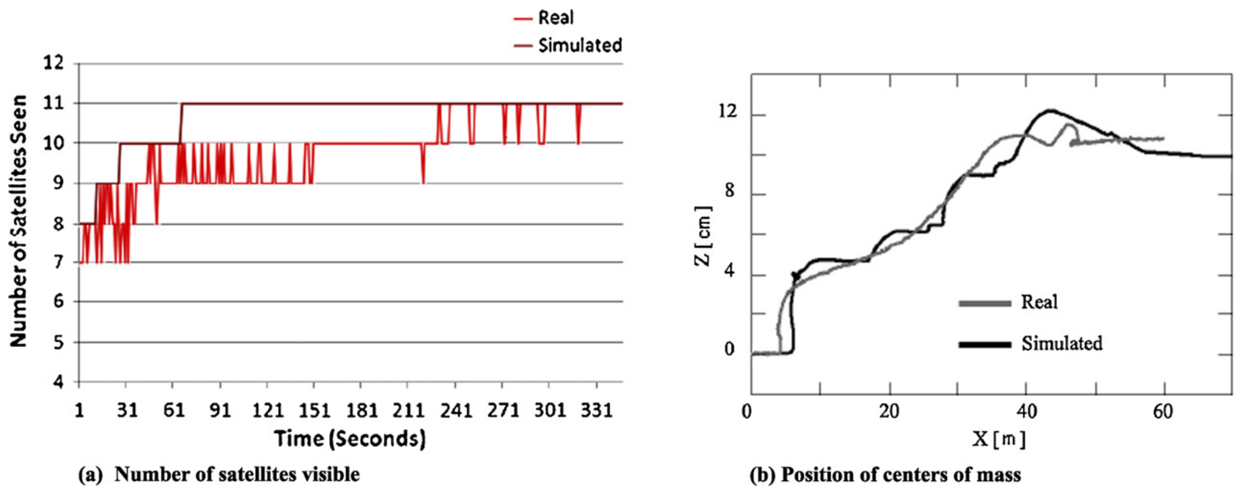


**Fig. 10.** Simulated and real results comparison.

## Acknowledgement

The author will be grateful to the State Key Lab of CAD&CG in Zhejiang University for the original basis into this paper. We should also acknowledge the reviewers to the paper for a series of improvement suggestions.

## References

[1] Z. Wang, Interactive virtual prototyping of a mechanical system considering the environment effect. Part 1: Modeling dynamics, C. R. Mecanique 399 (2011), doi:10.1016/j.crme.2011.06.001.
[2] J. Baumgarte, Stabilization of constraints and integrals of motion in dynamical systems, Comput. Methods Appl. Mech. Engrg. 1 (1972) 1–16.
[3] C.O. Chang, P.E. Nikravesh, An adaptive constraint violation stabilization method for dynamic analysis of mechanical systems, J. Mech. Transmissions Autom. Design 107 (1985) 488–492.
[4] G.P.O. Stermeyer, On Baumgarte stabilization for differential algebraic equations, in: J. Haug, R.C. Deyo (Eds.), Real-Time Integration Methods for Mechanical System Simulations, in: NATO ASI Series, vol. F69, Springer, Berlin, 1990, pp. 193–207.
[5] S. Yoon, R.M. Howe, D.T. Greenwood, Stability and accuracy analysis of Baumgarte's constrained violation stabilization method, J. Mech. Design 117 (1995) 446–453.
[6] S.T. Lin, M.C. Hong, Stabilization method for numerical integration of multi-body mechanical systems, J. Mech. Design 120 (1998) 565–572.
[7] B. Simeon, MBSPACK. Numerical integration software for constrained mechanical motion, Surv. Math. Ind. 5 (1995) 169–202.
[8] C.W. Gear, B. Leimkuhler, G.K. Gupta, Automatic integration of Euler–Lagrange equations with constraints, J. Comput. Appl. Math. 12–13 (1985) 77–90.
[9] C. Führer, B. Leimkuhler, Numerical solution of differential-algebraic equations for constrained mechanical motion, Numer. Math. 59 (1991) 55–69.
[10] W.M. Seiler, Numerical integration of constrained Hamiltonian systems using Dirac brackets, Math. Comp. 68 (1999) 661–681.
[11] S. Yoon, R.M. Howe, D.T. Greenwood, Geometric elimination of constraint violations in numerical simulation of Lagrangian equations, J. Mech. Design 116 (1994) 1058–1064.
[12] W. Blajer, A geometric unification of constrained system dynamics, Multibody Syst. Dyn. 1 (1997) 3–21.
[13] Z. Terze, D. Lefeber, O. Muftic, Null space integration method for constrained multi-body systems with no constraint violation, Multibody Syst. Dyn. 6 (2001) 229–243.
[14] W. Blajer, Elimination of constraint violation and accuracy aspects in numerical simulation of multi-body system, Multibody Syst. Dyn. 7 (2002) 265–284.
[15] A. Laulusa, O.A. Bauchau, Review of classical approaches for constraint enforcement in multi-body systems, J. Comput. Nonlinear Dynam. 3 (2008) 11004.
[16] O.A. Bauchau, A. Laulusa, Review of contemporary approaches for constraint enforcement in multi-body systems, J. Comput. Nonlinear Dynam. 3 (2008) 011005.
[17] D.J. Braun, M. Goldfarb, Eliminating constraint drift in the numerical simulation of constrained dynamical systems, Comput. Methods Appl. Mech. Engrg. 198 (2009) 3151–3160.
[18] D. Baraff, Linear-time simulation using Lagrange-multipliers, in: Proc. SIGGRAPH, 1996, pp. 137–146.
[19] U.M. Ascher, D.K. Pai, P.G. Kry, Forward dynamics algorithms for multi-body chains and contact, in: Proc. IEEE Int. Conf. Robotics and Automation, 2000, pp. 857–862.
[20] Garcia de Jalon, E. Bayo, Kinematic and Dynamic, Simulation of Multi-Body Stems: The Real Time Challenge, Springer-Verlag, 1994.
[21] K.M. Lilly, Efficient Dynamic Simulation of Mechanisms, Kluwer, 1993.
[22] A. Eichberger, W. Rulka, Process save reduction by macro joint approach: The key to real time and efficient vehicle simulation, Vehicle System Dynamics 41 (2004) 401–413.
[23] W. Rulka, E. Pankiewicz, MBS approach to generate equations of motions for HiL-simulations in vehicle dynamics, Multibody Syst. Dyn. 14 (2005) 367–386.
[24] E. Hairer, G. Wanner, Solving Ordinary Differential Equations. II. Stiff and Differential-Algebraic Problems, 2nd edition, Springer-Verlag, Berlin/Heidelberg/New York, 1996.
[25] A. Tasora, D. Negrut, M. Anitescu, Large-scale parallel multi-body dynamics with frictional contact on the graphical processing unit, Proc. Inst. Mech. Eng. K J. Multi-body Dyn. 222 (4) (2008) 315–326.
[26] Homepage of the Open Dynamics Engine (ODE), http://www.ode.org/.

[27] Homepage of the OpenTissue simulation framework, http://www.opentissue.org.
[28] F. Fleissner, P. Eberhard, Parallel load-balanced simulation for short-range interaction particle methods with hierarchical particle grouping based on orthogonal recursive bisection, Internat. J. Numer. Methods Engrg. 74 (2007) 531–553.
[29] E. Hairer, G. Wanner, Solving Ordinary Differential Equations. II. Stiff and Differential-Algebraic Problems, 2nd edition, Springer–Verlag, Berlin/ Heidelberg/New York, 1996.