MULTIBODY DYNAMICS 2005, ECCOMAS Thematic Conference
J.M. Goicolea, J.Cuadrado, J.C.García Orden (eds.)
Madrid, Spain, 21–24 June 2005

MODULAR FORWARD DYNAMIC SIMULATION OF CONSTRAINED MECHANICAL SYSTEMS

Roland Kasper[*], Dmitry Vlasenko[†]

[*] Otto-von-Guericke-University Magdeburg
Institute of Mechatronics and Drives (IMAT)
Universitätsplatz 2, D-39106 Magdeburg, Germany
e-mail: Roland.Kasper@mb.Uni-Magdeburg.de

[†] Otto-von-Guericke-University Magdeburg
Institute of Mechatronics and Drives (IMAT)
Universitätsplatz 2, D-39106 Magdeburg, Germany
e-mail: Dmitri.Vlasenko@Masch-Bau.Uni-Magdeburg.DE

Keywords: Modular modelling, Dynamics Simulation, Multibodies.

Abstract. *This paper presents a recursive algorithm for calculating the forward dynamics of general rigid-body systems using a subsystem approach. It is an exact, non-iterative algorithm that is applicable to mechanisms with any joint type and any topology, including branches and kinematic loops. As stabilization is done also in a modular way, the method is well suited for distributed processing. The calculation of accelerations has O(log(n)) time complexity on O(n) processors, that is comparable with the fastest available parallel algorithms. The method was implemented in an object-oriented simulation tool, which can translate models from Autodesk Inventor.*

1 INTRODUCTION

The development of a tool for simulation of mechanical systems is a sophisticated problem. Simulating software should satisfy the wide set of conditions [1]: numerical efficiency, stability, distributivity, flexibility, interaction with other tools, distributed development, etc.

Trying to satisfy all demands, modern simulation tools use the object-oriented method [2]. But though the object-oriented approach has a huge number of advantages, this type of modularization in most cases is given up during simulation, especially for mechanical systems because common modelling formulations use access to the complete system e.g. to calculate all accelerations needed. On the other hand, there are big advantages of a simulation on the basis of subsystems:

- Subsystems can be modelled, tested and compiled. Then they can be used in a way similar to software components that encapsulate their internal structure and can be connected via interfaces.
- Critical effects like coulomb friction, backslash etc. can be encapsulated inside a subsystem.
- Subsystems are ideal candidates for the partitioning of large systems on multiple processors.

This paper describes an object-oriented method for the distributed simulation of multibodies with variable number of degrees of freedom. Unlike of a huge number of other methods, the described method keeps the block-module concept during simulation. The simulation of a mechanical subsystem has *O(log(n))* time complexity on *O(n)* processors, that is comparable with the fastest available parallel algorithms [3, 4].

The method was implemented in the simulation software [5] that can be used for the simulation mechanical parts of mechatronic systems.

Also is developed an integration of the software with Autodesk Inventor. Design engineers can specify geometric and material data of simulation models inside Inventor and then translate it into the simulation tool. This approach minimise the model's development cost and training of the design engineers.

2 DESCRIPTION OF METHOD

The base idea of the method is to perform the simulation of mechanical systems using the hierarchy of submodels that builds up the complete system. The submodels of the first level in general consist of connected bodies. The submodels of next levels (called children) consist, without loss of generality, of connected submodels (called parents). Since the main number of calculations proceeds inside of submodels, it follows that the simulation can be distributed easily on several processors. During the simulation on each time step the following tasks, shown in Fig. 1, have to be performed:

1. Distributed calculation of absolute accelerations $\dot{\mathbf{V}}(t_n)$.
2. Calculation of the absolute coordinates and velocities on the next time step. Using a favourite ODE integration scheme (e.g. Runge-Kutta or some multistep method), the value of absolute coordinates $\tilde{\mathbf{X}}(t_{n+1})$ and velocities $\tilde{\mathbf{V}}(t_{n+1})$ on the new time step can be obtained.
3. Distributed stabilization of coordinates $\mathbf{X}(t_{n+1})$ and velocities $\mathbf{V}(t_{n+1})$.

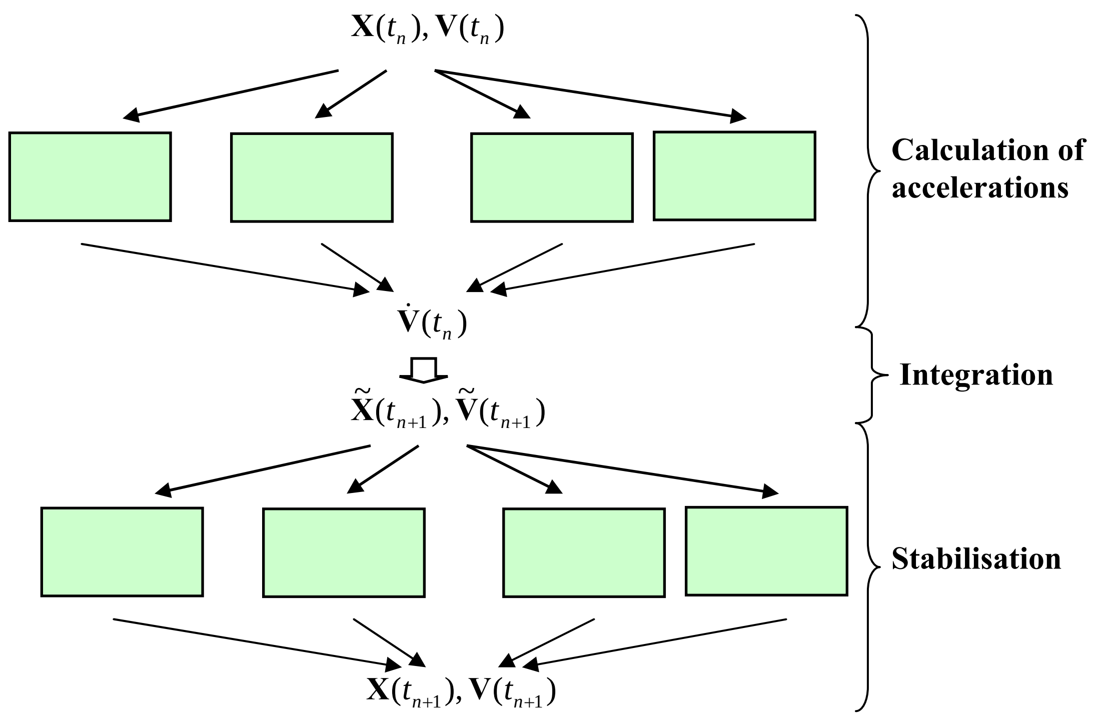

$\mathbf{X}(t_n), \mathbf{V}(t_n)$
Calculation of accelerations
$\dot{\mathbf{V}}(t_n)$
Integration
$\tilde{\mathbf{X}}(t_{n+1}), \tilde{\mathbf{V}}(t_{n+1})$
Stabilisation
$\mathbf{X}(t_{n+1}), \mathbf{V}(t_{n+1})$

Figure 1. Simulation steps

3 DISTRIBUTED CALCULATION OF ACCELERATION

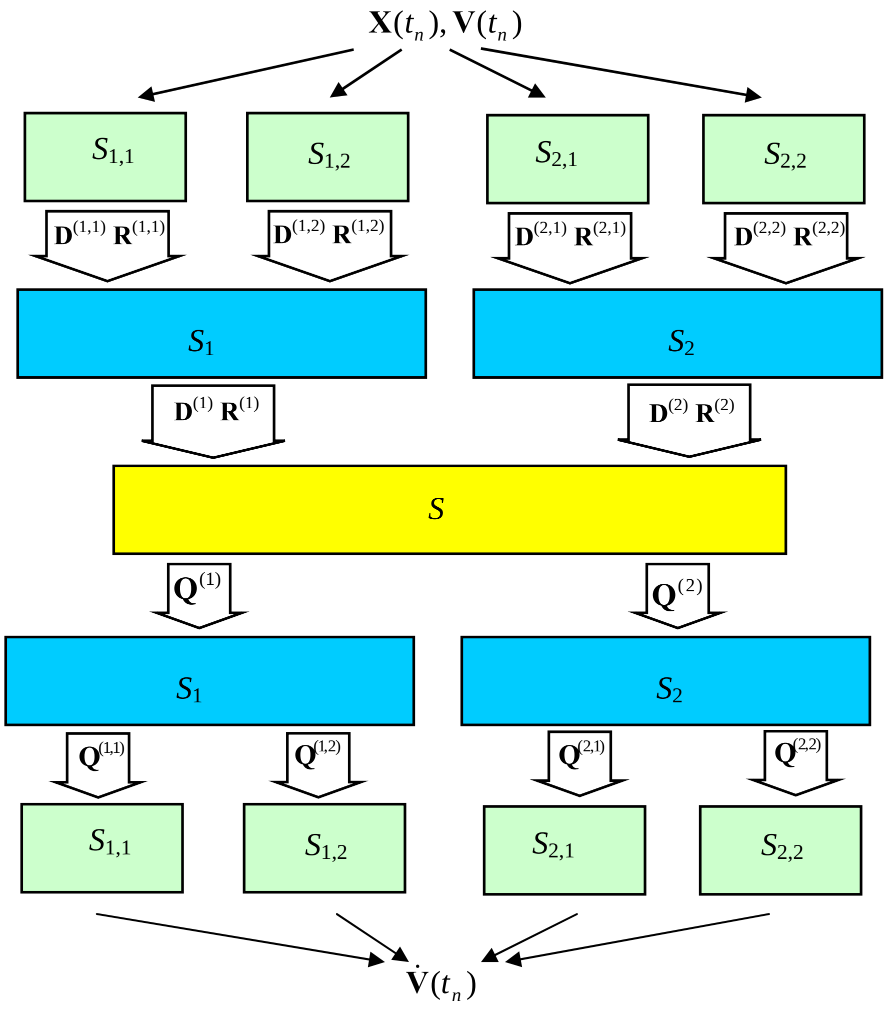
The Newton-Euler equation of motion describing the dynamics of constrained multibody system can be written in the form:

$$\begin{aligned} &\dot{\mathbf{X}} = \mathbf{V} \\ &\mathbf{M}(\mathbf{X})\dot{\mathbf{V}} = \mathbf{f}(\mathbf{X},\mathbf{V}) - \mathbf{G}^T(\mathbf{X})\boldsymbol{\lambda} \\ &\mathbf{g}(\mathbf{X}) = \mathbf{0} \end{aligned} \tag{1}$$

where

X is the vector of absolute coordinates of bodies
V is the vector of absolute velocities
M(**X**) is the mass matrix
f(X,V) is the vector of external forces (other than constrain forces)
g(X) is the vector of (holonomic) constraints
$\mathbf{G}(\mathbf{X}) = \dfrac{\partial \mathbf{g}}{\partial \mathbf{X}}$ is the constraint Jacobian matrix
$\boldsymbol{\lambda}$ is the vector of Lagrange multipliers.

Trying to find acceleration from (1) in a non-distributed way, the inverse of the matrix $\mathbf{GM}^{-1}\mathbf{G}^T$ will be needed that is a numerical-expensive procedure, proportional to the cube of the number of simulating bodies. That is why the distributed calculation of acceleration shown in Fig. 2 is preferred. For big good-partitioned models this method costs *O(n)* numerical operations, where *n* denotes the total number of bodies in the simulation model.



$\mathbf{X}(t_n), \mathbf{V}(t_n)$
$S_{1,1}$
$S_{1,2}$
$S_{2,1}$
$S_{2,2}$
$\mathbf{D}^{(1,1)}\ \mathbf{R}^{(1,1)}$
$\mathbf{D}^{(1,2)}\ \mathbf{R}^{(1,2)}$
$\mathbf{D}^{(2,1)}\ \mathbf{R}^{(2,1)}$
$\mathbf{D}^{(2,2)}\ \mathbf{R}^{(2,2)}$
S_1
S_2
$\mathbf{D}^{(1)}\ \mathbf{R}^{(1)}$
$\mathbf{D}^{(2)}\ \mathbf{R}^{(2)}$
S
$\mathbf{Q}^{(1)}$
$\mathbf{Q}^{(2)}$
S_1
S_2
$\mathbf{Q}^{(1,1)}$
$\mathbf{Q}^{(1,2)}$
$\mathbf{Q}^{(2,1)}$
$\mathbf{Q}^{(2,2)}$
$S_{1,1}$
$S_{1,2}$
$S_{2,1}$
$S_{2,2}$
$\dot{\mathbf{V}}(t_n)$

Figure 2. Calculation of acceleration steps

3.1 Hierarchical generations of equations of motion

Each subsystem gets from its parents their dependency matrices $\mathbf{D}^{(i)}$ and $\mathbf{R}^{(i)}$:

$$\dot{\mathbf{v}}_e^{(i)} = \mathbf{D}^{(i)}\mathbf{Q}^{(i)} + \mathbf{R}^{(i)}$$

where

$\dot{\mathbf{v}}_e^{(i)}$ is the vector of accelerations of the *i*-th parent's bordering bodies

$\mathbf{Q}^{(i)}$ is the vector of forces acting in the i-th parent's external links.

For example, in Fig. 2 the subsystem S_1 gets matrices $\mathbf{D}^{(1,1)}$, $\mathbf{R}^{(1,1)}$, $\mathbf{D}^{(2,1)}$, $\mathbf{R}^{(2,1)}$ from its parents $S_{1,1}$, $S_{1,2}$ etc. Here a body is called *bordering* to a subsystem, if it has constraints in the subsystem and is connected with external joints. A body is called *internal* to a subsystem, if it has constraints in the subsystem and is not connected to any external joint.

The subsystem generates matrices $\mathbf{D}$ and $\mathbf{R}$ using the equation of constraints connecting the parents. Here $\mathbf{D}$ and $\mathbf{R}$ are the dependency matrices:

$$\dot{\mathbf{v}}_{\mathbf{e}} = \mathbf{DQ} + \mathbf{R}$$

where

$\dot{\mathbf{v}}_{\mathbf{e}}$ is the vector of accelerations of subsystem's bordering bodies

$\mathbf{Q}$ is the vector of forces in subsystem's external links.

Then the subsystem transmits $\mathbf{D}$ and $\mathbf{R}$ to its child.

3.2 Calculation of absolute accelerations on the top of hierarchy

The subsystem of the highest level calculates $\mathbf{Q}^{(i)}$ for each parent *i* and transmits it to the parent (e.g. in Fig. 2 the subsystem *S* transmits $\mathbf{Q}^{(1)}$, $\mathbf{Q}^{(2)}$ to its parents S_1, S_2 correspondingly).

3.3 Backward hierarchical calculation of absolute accelerations

Subsequently, each subsystem gets the current value of $\mathbf{Q}$ from its child (e.g. in Fig. 2 the subsystem $S_{1,2}$ gets $\mathbf{Q}^{(2)}$ from its child S_1). Using $\mathbf{Q}$ the subsystem calculates $\dot{\mathbf{v}}_{\mathbf{e}}$. Then for each parent *i* the subsystem calculates $\mathbf{Q}^{(i)}$ and transmits it to the parent. Finally the absolute accelerations of all bodies are obtained.

4 ALGORITHM OF DISTRIBUTED POST-STABILIZATION

Using Runge-Kutta method of the fourth order, the value of $\tilde{\mathbf{X}}(t_{n+1}), \tilde{\mathbf{V}}(t_{n+1})$ can be obtained on the new time step. To avoid drift errors, it is necessary to minimize the drift of the constraints on the new time step. This can be achieved by the post-stabilization of coordinates, based on Chin method [6]:

$$\mathbf{X}_{\mathbf{n+1}} = \hat{\mathbf{X}}_{\mathbf{n+1}} - \mathbf{Z}(\hat{\mathbf{X}}_{\mathbf{n+1}}) \tag{2}$$

Here Z is the displacement stabilizing constraints in the simulation system:

$$\mathbf{Z} = \mathbf{G}^{+}\mathbf{g} \tag{3}$$

where

$\mathbf{g} = \mathbf{g}(\hat{\mathbf{X}}_{\mathbf{n+1}})$ is the vector of constraints

$\mathbf{G} = \mathbf{G}(\hat{\mathbf{X}}_{\mathbf{n+1}}) = \dfrac{\partial \mathbf{g}}{\partial \hat{\mathbf{X}}}$ is the constraint's Jacobian matrix

$\mathbf{G}^{+}$ is the Moore-Penrose inverse of matrix $\mathbf{G}$.

Obviously, the explicit calculation of a pseudo-inverse is a numerical very expensive procedure. Furthermore directly building up the matrix **G** and its inversion is a global process that conflicts with the idea of distributed simulation. One way to overcome this is to distribute the post-stabilization of constraints. In the method proposed here, the distributed and undistributed stabilization calculates the same vector **Z,** but the distributed way is more effective. For big good-partitioned models the distributed stabilization costs *O(n)* operations, where *n* denote the total number of bodies in the simulating model. That is much less than the complexity of non-distributed stabilization of absolute coordinates, which needs *O(n³)*.

It should be also noted that for good-partitioned models the distributed stabilization has *O(log(n))* time complexity on *O(n)* processors, that gives the total *O(log(n))* time complexity of the simulation method.

Now the details of this procedure will be given more precisely.

4.1 Definitions

Consider an arbitrary body from the simulation model of a mechanical system. Let J denote the array of joints that are connected to the body. Let joints from J be included in subsystems S_1, S_2, … , S_m, that are situated on different hierarchy levels. Let $\mathbf{x}=(x_1 \;\; x \ldots x_7)$ denote the 7-length vector of body's coordinates, where x_1, x_2, x_3 are coordinates of the body's centre of mass and x_4, x_5, x_6, x_7 are Euler parameters of the body. Let g_0 denote the drift of normalization condition:

$$g_0 = x_4 + x_5 + x_6 + x_7 - 1$$

Then the stabilization of $\mathbf{x}$ can be written down as:

$$\mathbf{x}_{\mathbf{n+1}} = \tilde{\mathbf{x}}_{\mathbf{n+1}} - \mathbf{z} = \tilde{\mathbf{x}} - (\mathbf{z}^{(0)} + \mathbf{z}^{(1)} + \mathbf{z}^{(2)} + \ldots + \mathbf{z}^{(m)})$$

where

$\mathbf{z}^{(0)}$ is the displacement stabilizing normalization condition.

$\mathbf{z}^{(i)}$ is the displacement stabilizing constraints in subsystem S_i (i>0).

The vector $\mathbf{z}^{(0)}$ is called *internal body's displacement* and $\tilde{\mathbf{z}} = \mathbf{z}^{(1)} + \mathbf{z}^{(2)} + \ldots + \mathbf{z}^{(m)}$ *external body's displacement.* The distributed calculation of stabilization displacements $\mathbf{Z}^{(i)}$ is shown in Fig. 3:

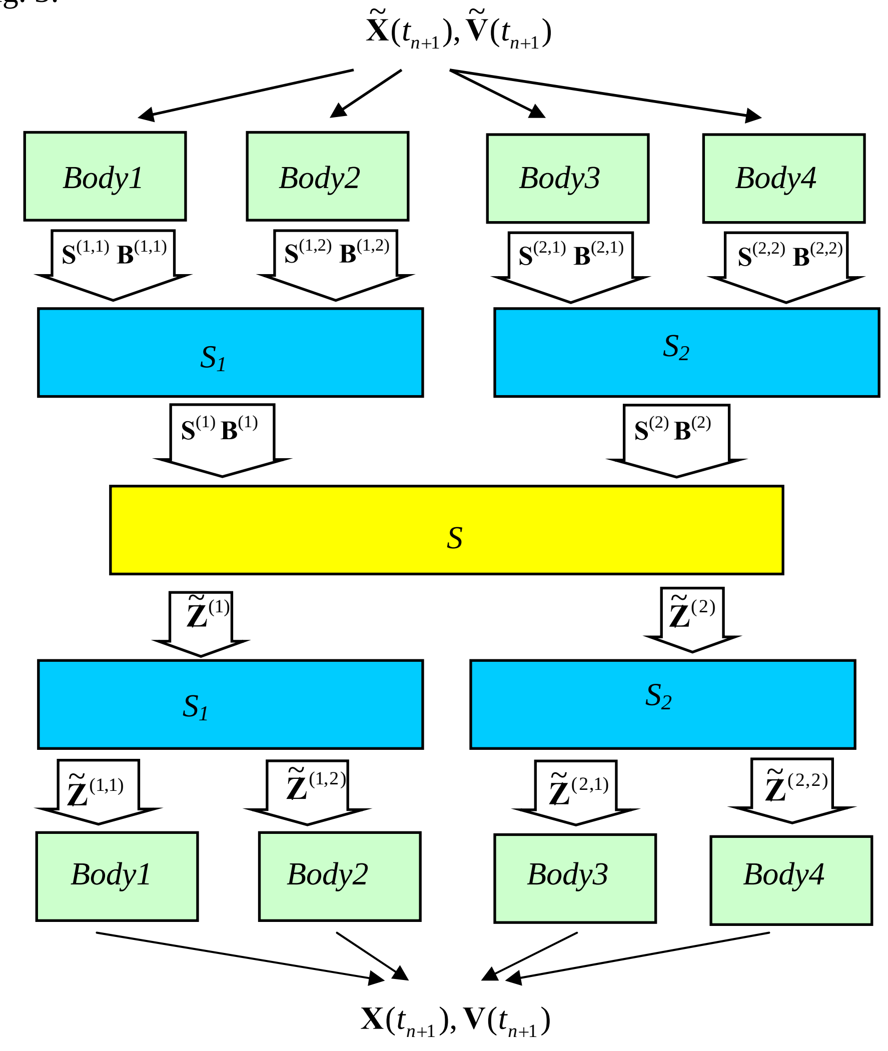

$\tilde{\mathbf{X}}(t_{n+1}), \tilde{\mathbf{V}}(t_{n+1})$
Body1
Body2
Body3
Body4
$\mathbf{S}^{(1,1)}\,\mathbf{B}^{(1,1)}$
$\mathbf{S}^{(1,2)}\,\mathbf{B}^{(1,2)}$
$\mathbf{S}^{(2,1)}\,\mathbf{B}^{(2,1)}$
$\mathbf{S}^{(2,2)}\,\mathbf{B}^{(2,2)}$
S_1
S_2
$\mathbf{S}^{(1)}\,\mathbf{B}^{(1)}$
$\mathbf{S}^{(2)}\,\mathbf{B}^{(2)}$
S
$\tilde{\mathbf{Z}}^{(1)}$
$\tilde{\mathbf{Z}}^{(2)}$
S_1
S_2
$\tilde{\mathbf{Z}}^{(1,1)}$
$\tilde{\mathbf{Z}}^{(1,2)}$
$\tilde{\mathbf{Z}}^{(2,1)}$
$\tilde{\mathbf{Z}}^{(2,2)}$
Body1
Body2
Body3
Body4
$\mathbf{X}(t_{n+1}), \mathbf{V}(t_{n+1})$

Figure 3. Stabilization steps

4.2 Generations of equations of bodies' normalized conditions

A body calculates dependency matrices **S** and **B**:

$$\mathbf{z}^{(\mathbf{0})} = \mathbf{S}\tilde{\mathbf{z}} + \mathbf{B}$$

where

$\mathbf{z}^{(0)}$ is *internal body's displacement*
$\tilde{\mathbf{z}}$ is *external body's displacement.*

Then the body transmits **S** and **B** to its submodel of the first level of hierarchy (e.g. in Fig. 3 body 1 transmits $\mathbf{S}^{(1,1)}$, $\mathbf{B}^{(1,1)}$ matrices to its parent – the subsystem S_1).

4.3 Forward hierarchical generations of stabilization's equations

Consider a stabilization of constraints in an arbitrary subsystem. Let the subsystem include joints, stabilizing k bodies: *body*$_1$, *body*$_2$, … , *body*$_k$. It is necessary to calculate the dependency of vector $\mathbf{Z} = \begin{pmatrix}\mathbf{z}_1^T & \mathbf{z}_2^T & \ldots & \mathbf{z}_k^T\end{pmatrix}^T$, stabilizing constraints in the subsystem, from the vector of external displacements $\tilde{\mathbf{Z}}$, stabilizing constraints in subsystems of higher levels. The subsystem gets from its parents their dependency matrices $\mathbf{S}^{(\mathbf{t})}$ and $\mathbf{B}^{(\mathbf{t})}$:

$$\mathbf{Z}_i^{(t)} = \mathbf{S}^{(t)}\tilde{\mathbf{Z}}^{(t)} + \mathbf{B}^{(t)}$$

where

$\mathbf{Z}_i^{(t)}$ is the vector of displacement stabilizing constraints in t-th parent.
$\tilde{\mathbf{Z}}^{(t)}$ is the vector of external displacements acting on bodies bordering to t-th parent.

Then the subsystem generates dependency matrices **S** and **B** using the equation of constraints:

$$\mathbf{Z} = \mathbf{S}\tilde{\mathbf{Z}} + \mathbf{B}$$

where

$\mathbf{Z}$ is the vector of displacement stabilizing subsystem's constraints
$\tilde{\mathbf{Z}}$ is the vector of external displacements acting on bodies bordering to the subsystem.

Then the subsystem transmits **S** and **B** to its child (e.g. in Fig. 3 the subsystem S_1 transmits $\mathbf{S}^{(1)}$, $\mathbf{B}^{(1)}$ matrices to its parent – the subsystem S).

4.4 Backward hierarchical calculation of displacements and the stabilization of constraints

A subsystem gets the current value of $\tilde{\mathbf{Z}}$ from its child. Using $\tilde{\mathbf{Z}}$ the subsystem calculates $\mathbf{Z} = \begin{pmatrix}\mathbf{z}_1^T & \mathbf{z}_2^T & \ldots & \mathbf{z}_k^T\end{pmatrix}^T$. Then the subsystem changes the coordinates of subsystem's bodies:

$$Body_i.\mathbf{x} := Body_i.\mathbf{x} - \mathbf{z}_k$$

Then for each $t=1$ ... k the subsystem calculates $\tilde{\mathbf{Z}}^{(t)}$ and transmits it to the t-th parent.

4.5 Stabilization of bodies' normalization conditions

A body gets the current value of $\tilde{\mathbf{z}}$ from its subsystem of first level of hierarchy. Using $\tilde{\mathbf{z}}$ the body calculates $\mathbf{z}^{(0)}$ and changes coordinates:

$$Body.\mathbf{x} := Body.\mathbf{x} - \mathbf{z}^{(0)}$$

4.6 Distributed stabilization of constraints on velocity level

Because of the discretization of the model there is a need to stabilize also the constraints on the velocity level:

$$0 = \dot{g} = \mathbf{GV}$$

The stabilization of velocities is performed in similar way, except the stabilization of internal bodies' constraint and is omitted in this paper.

5 EQUATIONS OF DISTRIBUTED POST-STABILIZATION

After the description of the processing flow, in the following a detailed mathematical basement of the algorithm of the distributed post-stabilization will be given.

5.1 Equations of bodies' stabilization

The stabilization vector of the complete system can be calculated using the formula

$$\mathbf{Z} = \mathbf{G}^{\mathbf{T}}(\mathbf{GG}^{\mathbf{T}})^{-1}\mathbf{g}$$

where

$\mathbf{g} = \mathbf{g}(\hat{\mathbf{X}}_{\mathbf{n+1}})$ is the global vector of simulation system's constraints

$\mathbf{G} = \mathbf{G}(\hat{\mathbf{X}}_{\mathbf{n+1}}) = \dfrac{\partial \mathbf{g}}{\partial \hat{\mathbf{X}}}$ is the constraint Jacobian matrix.

Let **M** denote the vector:

$$\mathbf{M} = (\mathbf{GG}^{\mathbf{T}})^{-1}\mathbf{g} \tag{4}$$

Then **Z** is equal to:

$$\mathbf{Z} = \mathbf{G}^{\mathbf{T}}\mathbf{M} \tag{5}$$

Equation (4) can be rewritten as:

$$\mathbf{g} = (\mathbf{GG}^{\mathbf{T}})\mathbf{M} \tag{6}$$

To simplify the proof it is helpful to reorder the vector **g** and the rows of the correspondent matrix **G**. From the object-oriented point of view, bodies can be considered as subsystems of zero level of hierarchy. Then normalized conditions of bodies can be rewritten as constraints of subsystems of zero level. Let the simulated system consist of N_0 bodies, N_1 submodels of the first level, N_2 derived submodels of the second level etc. Let C be the sequence of subsystems $C = \{ S_1^{(0)}, S_2^{(0)}, ..., S_{N_1}^{(0)}, \ldots, S_1^{(L)}, ..., S_{N_L}^{(L)} \}$, where $S_t^{(k)}$ denotes the t-th subsystem of k-th level and L denotes the number of levels of submodels. The vector of constraints **g** is reordered corresponding to C: $\mathbf{g} = \begin{pmatrix} g_1^{(0)} & ... & g_{N_1}^{(0)} & \mathbf{g}_{N_1}^{(L)T} & ... & \mathbf{g}_{N_L}^{(L)T} \end{pmatrix}^T$, where $\mathbf{g}_t^{(k)}$ denotes the vector of constraints of t-th subsystem of k-th level.

Let all bodies be connected with joints. Then matrices **G, g** can be rewritten as:

$$\mathbf{g} = \begin{pmatrix} g_1^{(0)} \\ g_2^{(0)} \\ \dots \\ g_{N_0}^{(0)} \\ \tilde{\mathbf{g}} \end{pmatrix} \qquad \mathbf{G} = \begin{pmatrix} \mathbf{G}_1 & 0 & \dots & 0 \\ 0 & \mathbf{G}_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \mathbf{G}_{N_0} \\ \tilde{\mathbf{G}}_1 & \tilde{\mathbf{G}}_2 & \dots & \tilde{\mathbf{G}}_{N_0} \end{pmatrix}$$

where

$\mathbf{G}_t = \dfrac{\partial \mathbf{g}_t^{(0)}}{\partial x_t}$ is the Jacobian matrix of *t*-th body.

$\tilde{\mathbf{g}} = \left(\mathbf{g}_1^{(1)T} \quad \dots \quad \mathbf{g}_{N_L}^{(L)T}\right)^T$ is the vector of constraints of higher levels of hierarchy.

Matrix **M** can be divided in an analogue way into subvectors:

$$\mathbf{M} = \left(M_1 \quad \dots \quad M_{N_0} \quad \tilde{\mathbf{M}}^T\right)^T$$

From equation (5) it can be obtained:

$$\mathbf{Z} = \begin{pmatrix} M_1\mathbf{G}_1^T + \tilde{\mathbf{G}}_1^T\tilde{\mathbf{M}} \\ \vdots \\ M_{N_0}\mathbf{G}_{N_0}^T + \tilde{\mathbf{G}}_{N_0}^T\tilde{\mathbf{M}} \end{pmatrix} = \begin{pmatrix} \mathbf{z_1} + \tilde{\mathbf{z}}_\mathbf{1} \\ \vdots \\ \mathbf{z}_{\mathbf{N_0}} + \tilde{\mathbf{z}}_{\mathbf{N_0}} \end{pmatrix}$$

The meaning of $\mathbf{z}_t = M_t\mathbf{G}_t^T$ is the displacement of body *t* stabilizing its normalized conditions, i.e. internal body's displacement. The meaning of $\tilde{\mathbf{z}}_t = \tilde{\mathbf{G}}_t^T\tilde{\mathbf{M}}$ is the displacement of body *t* stabilizing constraints of higher levels, i.e. external body's displacement.

Then (6) can be rewritten as:

$$\begin{pmatrix} g_1^{(0)} \\ \vdots \\ g_{N_0}^{(0)} \\ \tilde{\mathbf{g}} \end{pmatrix} = \begin{pmatrix} \mathbf{G}_1\mathbf{G}_1^T M_1 + \mathbf{G}_1\tilde{\mathbf{G}}_1^T\tilde{\mathbf{M}} \\ \vdots \\ \mathbf{G}_{N_0}\mathbf{G}_{N_0}^T M_{N_0} + \mathbf{G}_{N_0}\tilde{\mathbf{G}}_{N_0}^T\tilde{\mathbf{M}} \\ \tilde{\mathbf{G}}_1\mathbf{G}_1^T M_1 + \dots + \tilde{\mathbf{G}}_{N_0}\mathbf{G}_{N_0}^T M_1 + \tilde{\mathbf{G}}\tilde{\mathbf{G}}^T\tilde{\mathbf{M}} \end{pmatrix} \tag{7}$$

where $\tilde{\mathbf{G}} = \left(\tilde{\mathbf{G}}_1 \quad \tilde{\mathbf{G}}_2 \quad \dots \quad \tilde{\mathbf{G}}_{N_0}\right)$ is the Jacobian matrix of constraints of higher levels.

Using definitions of $\mathbf{z}_t$ and $\tilde{\mathbf{z}}_t$, the new system of equations is obtained:

$$\begin{aligned} g_t^{(0)} &= \mathbf{G}_t\mathbf{z}_t + \mathbf{G}_t\tilde{\mathbf{z}}_t \qquad t = 1\dots N_0 \\ \tilde{\mathbf{g}} &= \tilde{\mathbf{G}}_1\mathbf{z}_1 + \dots + \tilde{\mathbf{G}}_{N_0}\mathbf{z}_{N_0} + \tilde{\mathbf{G}}\tilde{\mathbf{Z}} \end{aligned} \tag{8}$$

where $\mathbf{Z} = \left(\tilde{\mathbf{z}}_\mathbf{1}^T \quad \dots \quad \tilde{\mathbf{z}}_{\mathbf{N_0}}^T\right)^T$ is the displacement of all bodies stabilizing constraints of higher levels.

Therefore, for each *t* the dependency of the *t*-th body's internal displacement $\mathbf{z}_t$ on the external body's displacement $\tilde{\mathbf{z}}_t$ can be expressed as:

$$\mathbf{z}_t = \mathbf{S}_t \tilde{\mathbf{z}}_\mathbf{t} + \mathbf{B}_t \tag{9}$$

where

$$\mathbf{S} = -\mathbf{G}_t^+ \mathbf{G}_t$$
$$\mathbf{B} = g_t^{(0)} \mathbf{G}_t^+$$

Here $\mathbf{G}_t^+$ is the Moore-Penrose inverse of matrix $\mathbf{G}_t$.

The new equation can be obtained as the result of substitution (9) in the lower part of (8):

$$\tilde{\mathbf{g}} = \tilde{\mathbf{G}} \begin{pmatrix} \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_{N_0} \end{pmatrix} + \tilde{\mathbf{G}}\tilde{\mathbf{Z}} = \tilde{\mathbf{G}} \begin{pmatrix} \mathbf{S_1} + \mathbf{E} & ... & 0 \\ ... & ... & ... \\ 0 & .. & \mathbf{S_{N_0}} + \mathbf{E} \end{pmatrix} \tilde{\mathbf{Z}} + \tilde{\mathbf{G}} \begin{pmatrix} \mathbf{B}_1 \\ \vdots \\ \mathbf{B}_{N_0} \end{pmatrix}$$

That can be rewritten as:

$$\tilde{\mathbf{g}} = \tilde{\mathbf{G}}\tilde{\mathbf{D}}\tilde{\mathbf{Z}} + \tilde{\mathbf{G}}\tilde{\mathbf{B}} \tag{10}$$

where $\tilde{\mathbf{D}}$ is a symmetric block-diagonal matrix.

5.2 Equations of a subsystem stabilization

Consider a subsystem $S_1^{(1)}$. Let g_i denote the vector of constraints in the subsystem. Then vector $\tilde{\mathbf{g}}$ can be rewritten as

$$\tilde{\mathbf{g}} = \left(\mathbf{g}_i^T \quad \hat{\mathbf{g}}^T \right)^T$$

Let $\mathbf{X}_e$ denote the vector of coordinates of subsystem's bordering bodies, $\mathbf{X}_i$ denote the vector of coordinates of subsystem's internal bodies. Let $\hat{\mathbf{X}}_U$ denote the vector of coordinates of bodies that do not have constraints in the subsystem.

Reordering coordinates allows rewriting (10) in the form:

$$\begin{pmatrix} \mathbf{g_i} \\ \hat{\mathbf{g}} \end{pmatrix} = \begin{pmatrix} \mathbf{G}_{ii} & \mathbf{G}_{ie} & 0 \\ 0 & \hat{\mathbf{G}}_{ee} & \hat{\mathbf{G}}_U \end{pmatrix} \begin{pmatrix} \mathbf{D}_{ii} & \mathbf{D}_{ie} & 0 \\ \mathbf{D}_{ie}^T & \mathbf{D}_{ee} & 0 \\ 0 & 0 & \hat{\mathbf{D}}_U \end{pmatrix} \begin{pmatrix} \mathbf{Z}_i \\ \mathbf{Z}_e + \tilde{\mathbf{Z}} \\ \hat{\mathbf{Z}}_U \end{pmatrix} + \begin{pmatrix} \mathbf{G}_{ii} & \mathbf{G}_{ie} & 0 \\ 0 & \hat{\mathbf{G}}_{ee} & \hat{\mathbf{G}}_U \end{pmatrix} \begin{pmatrix} \mathbf{B}_i \\ \mathbf{B}_e \\ \hat{\mathbf{B}}_U \end{pmatrix} \tag{11}$$

where

$\mathbf{G}_i = \left(\mathbf{G}_{ii} \quad \mathbf{G}_{ie} \right) = \begin{pmatrix} \dfrac{\partial \mathbf{g}_i}{\partial \mathbf{X}_i} & \dfrac{\partial \mathbf{g}_i}{\partial \mathbf{X}_e} \end{pmatrix}$ is the Jacobian matrix

$\mathbf{Z}_A = \left(\mathbf{Z}_i^{\ T} \quad \mathbf{Z}_e^{\ T} \right)^T$ is the vector of displacement stabilizing subsystem's constraints

$\tilde{\mathbf{Z}}$ is the vector of displacement of bordering bodies stabilizing external constraints

$\hat{\mathbf{Z}}_U$ is the vector of displacement of external bodies stabilizing external constraints.

The goal is to express $\mathbf{Z}_A$ as the function of $\hat{\mathbf{Z}}_U$. Then $\mathbf{Z}_A$ can be substituted in the lower part of (11) and the equation that has the same form as (10), can be obtained, but with less number of displacements.

The matrix equation (11) can be transformed to the system of equations:

$$\begin{aligned}\mathbf{g_i} &= \mathbf{G_i}\mathbf{D}_A\mathbf{Z}_A + \mathbf{G_i}\mathbf{D}_e^{\mathbf{T}}\tilde{\mathbf{Z}} + \mathbf{G_i}\mathbf{B}_A \\ \hat{\mathbf{g}} &= \hat{\mathbf{G}}_{ee}\mathbf{D}_e\mathbf{Z}_A + \hat{\mathbf{G}}_{ee}\mathbf{D}_{ee}\tilde{\mathbf{Z}} + \hat{\mathbf{G}}_{\mathbf{U}}\hat{\mathbf{D}}_{\mathbf{U}}\hat{\mathbf{Z}}_{\mathbf{U}} + \hat{\mathbf{G}}_{ee}\mathbf{B}_e + \tilde{\mathbf{G}}_{\mathbf{U}}\hat{\mathbf{B}}_{\mathbf{U}}\end{aligned} \tag{12}$$

where

$$\mathbf{D}_A = \begin{pmatrix} \mathbf{D}_{ii} & \mathbf{D}_{ie} \\ \mathbf{D}_{ie}^T & \mathbf{D}_{ee} \end{pmatrix}$$

$$\mathbf{D}_e = \begin{pmatrix} \mathbf{D}_{ie}^T & \mathbf{D}_{ee} \end{pmatrix}$$

$$\mathbf{B}_A = \begin{pmatrix} \mathbf{B}_i^T & \mathbf{B}_e^T \end{pmatrix}^T$$

Finally, the desired dependency is obtained from the first equation of (12):

$$\mathbf{Z}_A = \mathbf{S}\tilde{\mathbf{Z}} + \mathbf{B} \tag{13}$$

where

$$\begin{aligned}\mathbf{S} &= -(\mathbf{G_i}\mathbf{D_i})^{+}\mathbf{G_i}\mathbf{D}_e^{\mathbf{T}} \\ \mathbf{B} &= -(\mathbf{G_i}\mathbf{D_i})^{+}(\mathbf{g_i} - \mathbf{G_i}\mathbf{B}_A)\end{aligned} \tag{14}$$

Substitution $\mathbf{Z}_A$ in second equation of (12) leads to the equation:

$$\hat{\mathbf{g}} = \hat{\mathbf{G}}_{ee}(\mathbf{D}_e\mathbf{S} + \mathbf{D}_{ee})\tilde{\mathbf{Z}} + \hat{\mathbf{G}}_{ee}(\mathbf{D}_e\mathbf{B} + \mathbf{B}_x) + \hat{\mathbf{G}}_{\mathbf{U}}\hat{\mathbf{D}}_{\mathbf{U}}\hat{\mathbf{Z}}_{\mathbf{U}} + \hat{\mathbf{G}}_{\mathbf{U}}\hat{\mathbf{B}}_{\mathbf{U}}$$

Therefore,

$$\hat{\mathbf{g}} = \hat{\mathbf{G}}\hat{\mathbf{D}}\hat{\mathbf{Z}} + \tilde{\mathbf{G}}\hat{\mathbf{B}}$$

where

$$\hat{\mathbf{G}} = \begin{pmatrix} \tilde{\mathbf{G}}_{ee} & \hat{\mathbf{G}}_U \end{pmatrix}$$

$$\hat{\mathbf{D}} = \begin{pmatrix} \mathbf{D}_e\mathbf{S} + \mathbf{D}_{ee} & 0 \\ 0 & \hat{\mathbf{D}}_u \end{pmatrix}$$

$$\hat{\mathbf{B}} = \begin{pmatrix} \mathbf{D}_e\mathbf{B} + \mathbf{B}_x \\ \hat{\mathbf{B}}_{\mathbf{U}} \end{pmatrix} \qquad \hat{\mathbf{Z}} = \begin{pmatrix} \tilde{\mathbf{Z}} \\ \hat{\mathbf{Z}}_U \end{pmatrix}$$

So, an equation of the same form as (10) can be obtained, but now it does not include subsystem's constraints and subsystem's internal bodies. Obviously, the same procedure can be consequently implemented for all subsystems in *C*. From (14) follows that matrices **S** and **B** can be calculated using only the information about subsystem's constraints ($\mathbf{g}_\mathrm{i}$, $\mathbf{G}_\mathrm{i}$ matrices) and information about sybsystem's parents ($\mathbf{S}^{(t)}, \mathbf{B}^{(t)}$ matrices and some additional matrices, that are needed for the calculation of $\mathbf{B}_\mathrm{A}$**,** $\mathbf{D}_i$**,** $\mathbf{D}_e$).

5.3 Equations of stabilization on the highest level of hierarchy

Consequently repeating the procedure described in the previous chapter, gives the equations of displacements in the subsystem of the highest level of hierarchy:

$$\hat{\mathbf{g}} = \hat{\mathbf{G}}\hat{\mathbf{D}}\hat{\mathbf{Z}} + \tilde{\mathbf{G}}\hat{\mathbf{B}}$$

where

$\hat{\mathbf{g}}$ is the vector of subsystem's constraints

$\hat{\mathbf{G}} = \frac{\partial \hat{\mathbf{g}}}{\partial \mathbf{X}}$ is the Jacobian matrix.

X is the vector of subsystem's bodies

$\hat{\mathbf{Z}}$ is the vector of displacement stabilizing subsystem's constraints

Therefore, the value of $\hat{\mathbf{Z}}$ is obtained:

$$\hat{\mathbf{Z}} = (\hat{\mathbf{G}}\hat{\mathbf{D}})^{+}(\hat{\mathbf{g}} - \tilde{\mathbf{G}}\hat{\mathbf{B}})$$

By *k* denote the number of subsystem's parents. Let $\tilde{\mathbf{Z}}^{(t)}$ be the vector of external displacements acting on bodies bordering to the *t*-th parent (*t*=1…*k*). Obviously, $\tilde{\mathbf{Z}}^{(t)}$ is a corresponding subvector of $\hat{\mathbf{Z}}$. The subsystem calculates $\tilde{\mathbf{Z}}^{(t)}$ and transmits it to the *t*-th parent.
Now (13) can be used for the backward hierarchical calculation of displacements. A subsystem gets $\tilde{\mathbf{Z}}$ from its child and calculates $\mathbf{Z}_A$. Then for each parent the subsystem generates the correspondent subvector of $\mathbf{Z}_A$ and transmits it to the parent. At the end of the process, the global vector of displacement **Z** is generated.

6 EXAMPLE OF SIMULATION

It was performed a simulation of the 6-bodies pendulum shown in Fig. 4. This example illustrates all advantages of the method: the object-oriented simulation of multibodies, the stabilization of a closed-loop system and the numerical efficiency of the distributed simulation.

The pendulum was modelled in Autodesk Inventor and converted to a simulation model. The simulation time interval was choosen to be [0s, 20s]. The simulation was performed with Runge-Kutta method of the fourth order with the fixed time step equal to 0.005*s*.

The pendulum consists of two identical subsystems (marked by light grey and by dark grey) connected with each other and with the ground (marked by black) by revolute joints. Each subsystem consists of three bodies connected by revolute joints. Obviously, subsystems are stiff and the complete system has only two degrees of freedom.

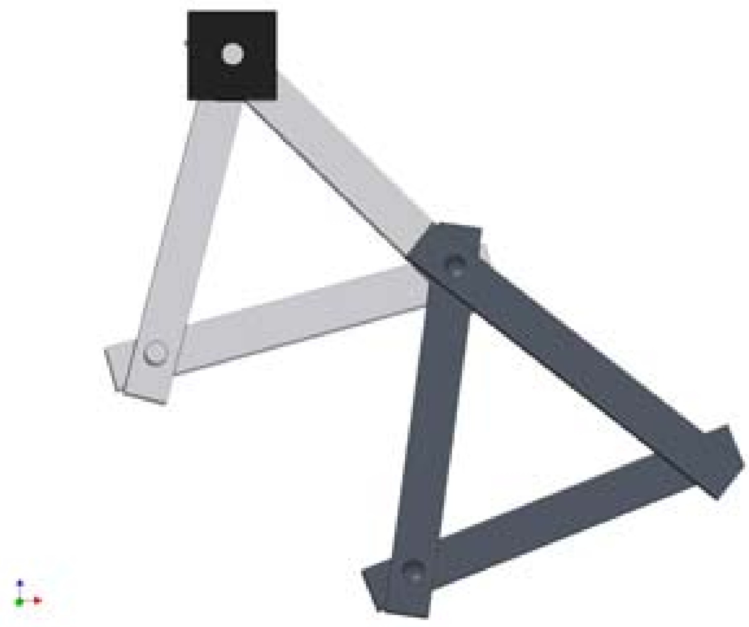


Figure 4. 6-Bodies pendulum

Fig. 5 shows the deviation of degrees of freedom from start values.

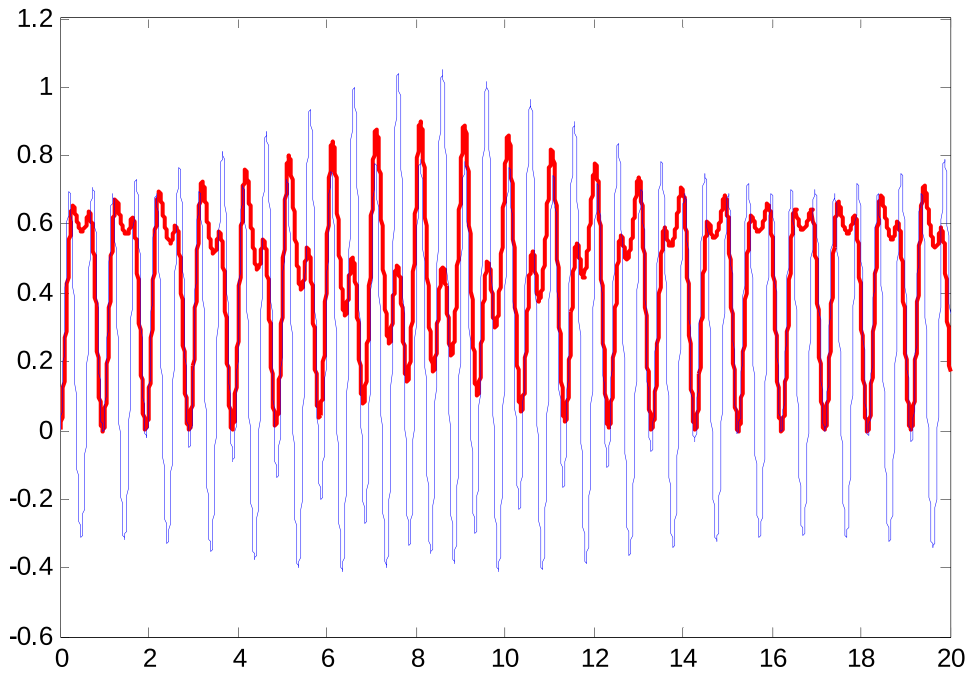


Figure 5. Oscillation of the pendulum

In Fig. 6 the drift of the distributed stabilization is presented. Experimental data show that the algorithm is stable and the drift of the model is about the accuracy of model’s definition.

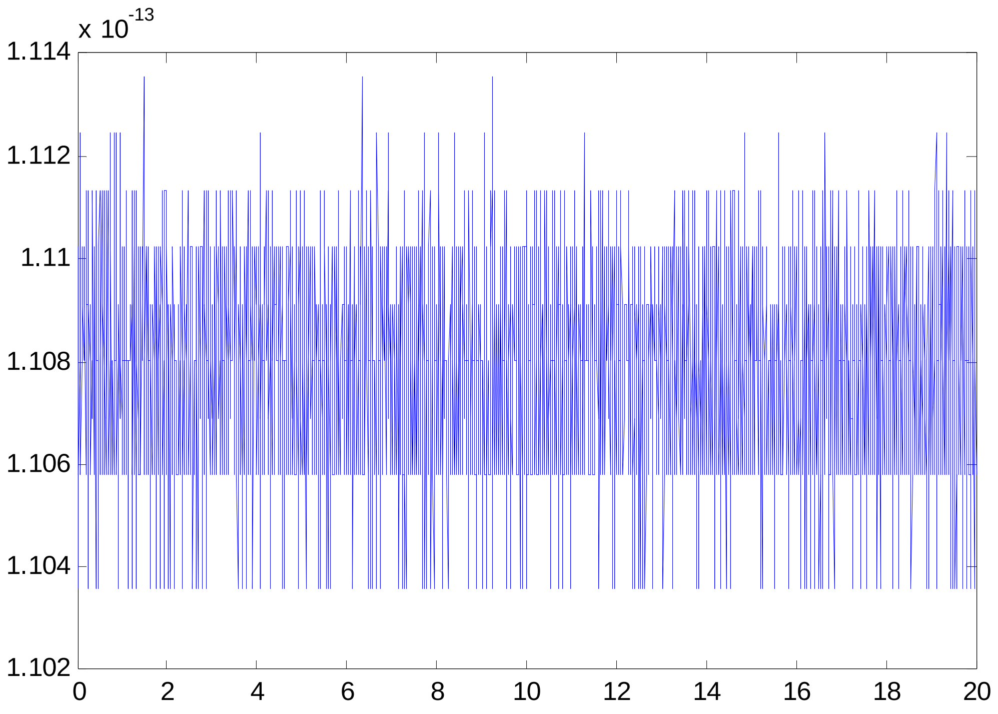


Figure 6: Drift of the model

While the distributed stabilization four 18x18 matrices and two 12x12 matrices have to be inverted, which needs about $5 \cdot 10^6$ arithmetic operations. That is much more numerical efficient than the undistributed stabilization of absolute coordinates (then the pseudo-inverse of a 54x42 matrix and a 54x36 matrix has to be calculated, which needs about $3 \cdot 10^7$ arithmetic operations).

7 CONCLUSION

This paper presents a new recursive object-oriented algorithm for calculating the forward dynamics of general rigid-body system using a subsystem approach that is well suited for distributed processing. It is an exact, non-iterative algorithm, and is applicable to mechanisms with any joint type and any topology, including branches and kinematic loops. The simulation of a mechanical subsystem has *O(log(n))* time complexity on *O(n)* processors, that is comparable with the fastest available parallel algorithms.

It was performed the implementation of the method and developed an object-oriented tool for simulation of multibody systems. Also is developed an integration of the software with Autodesk Inventor. Design engineers can specify geometric and material data of simulation model inside Inventor and then translate it into the simulation tool. This approach minimise the model’s development cost and training of the design engineers.

Experimental data show the stability of the method. The drift of closed-loop structures is limited for a long period of time. Thus, the experimental proof that the tool can be implemented for the simulation of large constrained multibody systems is obtained.

REFERENCES

[1] Cellier, F.E. (1996), *Object-Oriented Modeling*: *Means for Dealing With System Complexity*, Proc. 15th Benelux Meeting on Systems and Control, Mierlo, The Netherlands, pp.53-64., 1996

[2] Cellier, F.E., H. Elmqvist, and M. Otter (1995), *Modeling from Physical Principles*, The Control Handbook (W.S. Levine, ed.), CRC Press, Boca Raton, FL.

[3] Anderson, K. and Duan, S., 2000, *Highly parallelizable low-order dynamics simulation algorithm for multi-rigid-body systems*, AIAA Journal on Guidance, Control and Dynamics 23, no. 2, pp. 355-364.

[4] Featherstone, R., 1999, *A divide-and-conquer articulated-body algorithm for parallel O(log(n)) calculation of rigid-body dynamics, part 2: trees, loops and accuracy*, Int. Journal of Robotics Research 18, no. 9, pp. 876-892.

[5] R. Kasper, D. Vlasenko. *Method for distributed forward dynamic simulation of constrained mechanical systems*. Proceedings of the Eurosim Conference, Paris, 2004.

[6] U. Ascher, H. Chin, L. Petzold and S. Reich. *Stabilization of constrained mechanical systems with DAEs and invariant manifolds*. The Journal of Mechanics of Structures and Machines, 23(2), 135-157(1995).