

MBDyn - Multibody System Dynamics

LaMSID (EDF R&D - French CNRS Joint Laboratory)
July 24 - August 7, 2008



Pierangelo Masarati <masarati@aero.polimi.it>

Politecnico di Milano
Dipartimento di Ingegneria Aerospaziale

This presentation is split in two parts:

- **Multibody System Dynamics,**

**with specific reference to MBDyn,
the free general-purpose multibody solver
developed at “Dipartimento di Ingegneria Aerospaziale”
of “Politecnico di Milano”, Italy**

- **FEM interaction requirements of MBDyn,**

with the outline of possible solutions for the interaction with Aster

Outline

- **Overview**
- **Multibody dynamics**
- **Software architectures**
- **Problems**
- **Arbitrary motion description**
- **Deformable components**
- **Solving the problem**
- **Extracting useful information**
- **Examples of multibody modeling with MBDyn**
- **Future development**
- **Documentation and support**
- **Acknowledgments**

Multibody dynamics: overview

- **Multibody: a “buzz” word?**
- **Initial idea:**
 - automatically write equations of motion of arbitrary mechanisms
- **Current status:**
 - efficiently and accurately integrate in time
 - exact rigid-body kinematics, plus
 - nonlinear finite elements, plus
 - natural inclination towards multi-physics & system integration
- **Future:**
 - scale to larger and larger problems, and
 - higher performances when solving more complex problems

Multibody dynamics: overview

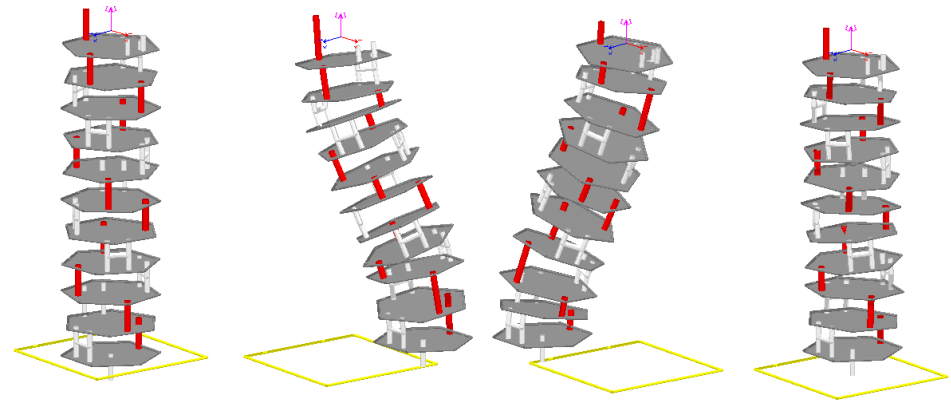
Multibody methods:

- **Usually are general-purpose: can model a wide variety of mechanical systems**
- **Must allow to automatically write motion equations**
- **Should support an arbitrary number of a variety of parts, forces, geometries, constraints, etc.**
- **Most often use numerical methods to compute solutions.**
- **Often integrated in CAD tools, with Graphical User Interfaces (GUI)**

Multibody dynamics: overview

Types of analysis:

- **Kinematics: direct, inverse**
- **Statics (pseudo-time)**
- **Dynamics: direct IVP, BVP, inverse dynamics**
- **Miscellaneous:**
 - modal analysis
 - sensitivity analysis
 - parameter optimization
 - topology optimization and mechanism synthesis









Multibody dynamics: overview

Applications:

- **Robotics**
 - **Automotive**
 - **Crashworthiness**
 - **Aerospace engineering**
 - **Industrial automation**
 - **Applied mechanics**
-
- **Videogames: revenue higher than engineering software!**
 - **Anywhere realistic motion (or at least that looks realistic) is required, multibody simulation is or can (and will) be used.**

Multibody dynamics: overview

Classification of approaches:

- **Based on coordinate set**
 - Minimal coordinate set 
 - Redundant coordinate set 
- **Based on local solution methods**
 - Direct linear solvers 
 - Iterative solvers 
- **Based on problem formulation**
 - Explicit 
 - Implicit 
- **Based on integration schemes**
 - Ordinary Differential Equations (ODE) w/ constraint stabilization
 - Differential-Algebraic Equations (DAE)

Basic equations:

$$\mathbf{M}(\mathbf{x})\ddot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, t)$$

- **mechanics of unconstrained system of bodies**
- **subjected to configuration-dependent loads**

Can be obtained from many (equivalent!) approaches:

- ***Newton-Euler*: linear/angular equilibrium of each body**
- ***d'Alembert-Lagrange*: virtual work of active forces/moments**
- ***Gauss, Hertz, Hamilton, ...*: variational principles**

Note: non-smooth mechanics tend to resort to discrete equations

$$\mathbf{M}(\mathbf{x})\Delta \dot{\mathbf{x}}_{k-1,k} = \int_{k-1}^k \mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, t) dt$$

Constrained system: kinematic constraints

- holonomic (algebraic)

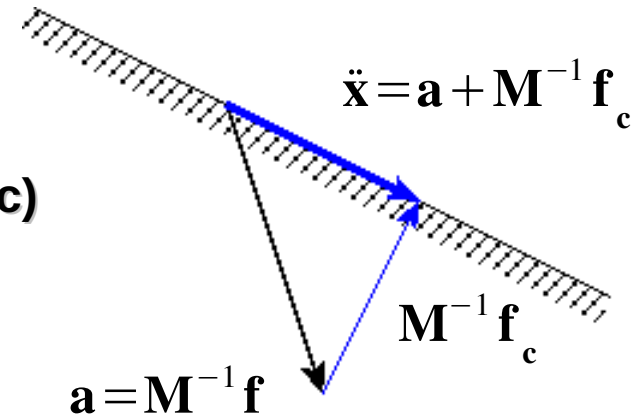
$$\phi(\mathbf{x}, t) = 0$$

- Non-holonomic (differential, not integrable to holonomic)

$$\psi(\mathbf{x}, \dot{\mathbf{x}}, t) = 0$$

usually

$$\mathbf{A}(\mathbf{x}, t) \dot{\mathbf{x}} = \mathbf{b}(\mathbf{x}, t)$$



- algebraic relationship between kinematic variables
- explicitly dependent on time: rheonomic
- scleronomous otherwise

Minimal set:

$$\mathbf{x} = \mathbf{x}(\mathbf{q}, t)$$

usually, this relationship:

- is not known in advance, or
- cannot be easily made explicit with respect to Lagrangian coordinates \mathbf{q}

Coordinate partitioning is required, e.g.:

- direct elimination from derivative of constraint equation
- QR, SVD or similar decomposition

Results in Maggi-Kane equations and similar approaches

Small system is obtained by expensive numerical reduction
(unless topology knowledge can be exploited)


Multibody dynamics

Redundant set:

$$\delta(\lambda \cdot \phi) = \delta \lambda \cdot \phi + \delta \mathbf{x} \cdot \phi_{/x}^T \lambda$$

$$\delta(\mu \cdot \psi) = \delta \mu \cdot \psi + \delta \mathbf{x} \cdot \psi_{/\dot{x}}^T \mu$$

By Lagrange multipliers:

- dynamics of constrained system using physical coordinates
 - constraint reactions applied to equations of motion
 - algebraic constraints explicitly added to the system
- mechanism made of multiple bodies with multiple constraints and few actual DoFs:
 - system size nearly doubles
 - mechanism made of deformable bodies with few constraints (e.g. FEM):
 - system size not significantly altered
 - sparsity is almost preserved 

$$\begin{bmatrix} \mathbf{M} & \phi_{/x}^T \\ \phi_{/x} & 0 \end{bmatrix} \begin{Bmatrix} \ddot{\mathbf{x}} \\ \lambda \end{Bmatrix} = \begin{Bmatrix} \mathbf{f} \\ \mathbf{b}' \end{Bmatrix}$$

Constraint equations written “as is”:

$$\phi(\mathbf{x}, t) = 0$$

problem becomes differential algebraic (DAE); issues:

- needs specific care to be solved: (nearly) L-stable integration, i.e.
 - unconditionally stable (A-stable), and
 - $\Delta \mathbf{x}_{k+1} \rightarrow 0$ for $\Delta t \rightarrow \infty$
- the constraint equation implies the additional constraints

$$\dot{\phi}(\mathbf{x}, t) = 0$$

$$\ddot{\phi}(\mathbf{x}, t) = 0$$

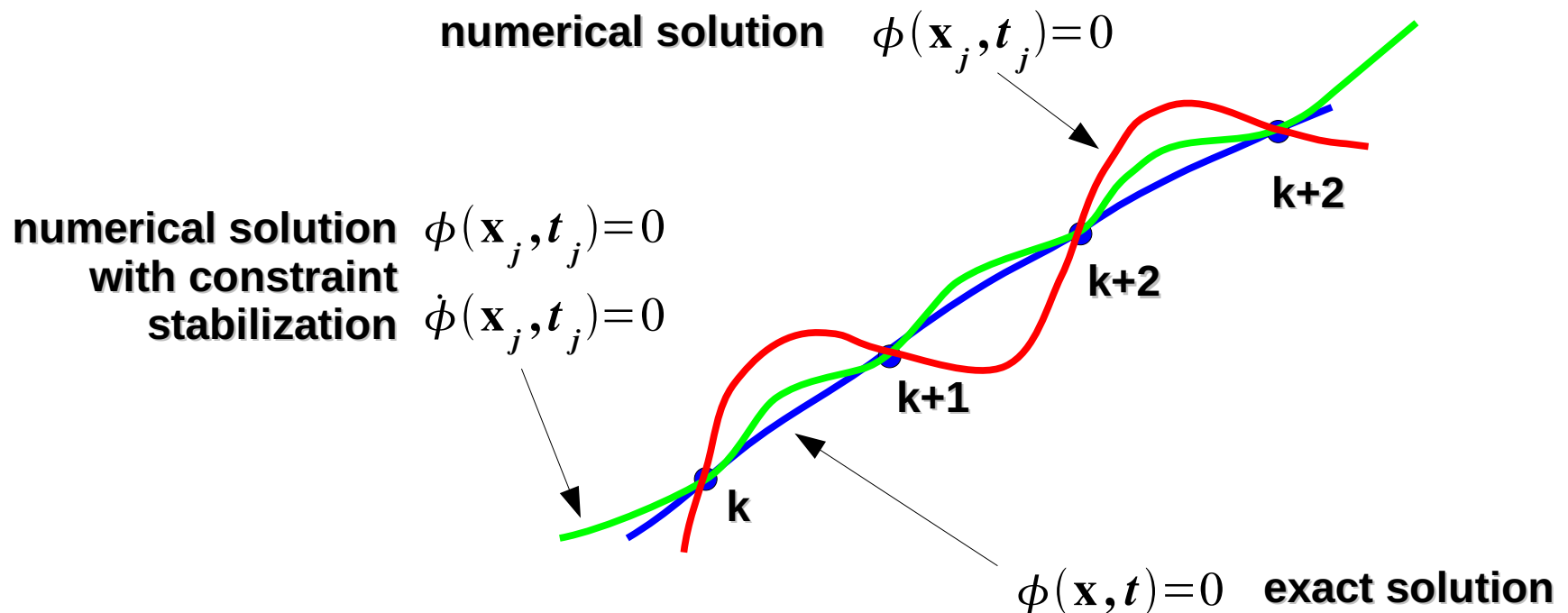
but they are not explicitly enforced:

may need constraint stabilization (Gear et al.)

Multibody dynamics

Constraint equations:

- \mathbf{x} is correct
- derivatives may be inaccurate
- multipliers may be inaccurate



Alternative: constraint equations differentiated to second order:

$$\phi_{/x} \ddot{\mathbf{x}} = \mathbf{b}'$$

problem remains ordinary differential (ODE);

- can be solved by conditionally stable algorithms
- the constraint equation does not imply the original constraints

$$\begin{aligned} \phi(\mathbf{x}, t) &= 0 \\ \dot{\phi}(\mathbf{x}, t) &= 0 \end{aligned}$$

suffers from drift! definitely needs constraint stabilization!

common technique: Baumgarte

$$\phi_{/x} \ddot{\mathbf{x}} = \mathbf{b}' - 2\alpha \dot{\phi} - \beta^2 \phi$$

(violation governed by asymptotically stable linear differential eq.)

Software architectures

- **Monolithic:**
 - user prepares specific model using built-in library elements
 - general-purpose solver swallows model and spits results
- **Library:**
 - user writes specific solver using library elements
 - usually needs programming skills; the solver must be compiled
 - specific solver solves the problem and spits results
- **Symbolic manipulators:**
 - user writes equations
 - symbolic manipulation engine solves equations and spits results
- **Modelica (and Modelica-like):**
 - user prepares model using a modeling language and libs
 - general-purpose interpreter generates specific solver
 - specific solver solves the problem and spits results

Software architectures

Free software examples (surely there are more):

- **Monolithic:**

- MBDyn

non-free counterparts omitted

- **Library:**

- DynaMechs (C++)
- Mbs3d (requires Matlab)
- Open Dynamics Engine (ODE) (C++)

- **Symbolic manipulators:**

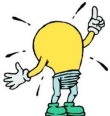
- 3D_MEC
- EasyDyn (MuPad)
- RoboTran (requires Matlab)

frequently architectures overlap

- **Modelica:**



- OpenModelica?

Software architectures

- **MBDyn is monolithic**
- **Input consists in a text file**
- **The input syntax allows some flexibility, e.g.:**
 - **math expressions evaluation**
 - **variables definition**
 - **“rigorous” syntax checking, but free style, indentation, ...**
- **Relevant portions of the code are modular and can be extended by:**
 - **writing run-time loadable modules**
 - **hacking the code (it's free, all in all!)**
- **There is no built-in pre-post processing facility**
- **Help in this area would be warmly appreciated!**
 - **MBDyn output can be translated into EasyAnim**
 - **there is an independent, partial customization based on Blender**
 -  **what about a custom Salome toolbox for multibody?**

Software architectures

MBDyn interacts with other software:

- **Generic streamed input/output**
 - **Input: “drive” (scalar); can be fed to several entities:**
 - imposed load
 - imposed joint motion
 - imposed deformable component prestrain
 - Inputs can be aggregated and manipulated
 - **Output: “measurement” (scalar); can be any exportable entity:**
 - any model state
 - internal parameters of elements and nodes
 - outputs can be aggregated and manipulated
 - **Used to interact with Simulink, Scicos, ... also in hard real-time**
- **Generic motion output/force input**
 - **Used for interaction with CFD solvers**
 -  **Could be (easily?) adapted for interaction with Aster?**
- **Other software needed to pre/post-process data (e.g. FEM)** 

Equations of motion: for each node (purely geometrical entity),

- **Newton-Euler, written as first-order system of equations:**

$$\mathbf{M} \dot{\mathbf{x}} = \mathbf{p} \quad (\text{for dynamic nodes only})$$

$$\dot{\mathbf{p}} = \mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, t) \quad (=0 \text{ for static nodes})$$

- **Momentum and momenta moment are used instead of pseudo-velocities**
- **allows multiple contributions to inertia of a single node (but constraint stabilization is no longer straightforward...)**

Constrained equations are directly written in differential-algebraic form:

$$\mathbf{M} \dot{\mathbf{x}} = \mathbf{p}$$

$$\dot{\mathbf{p}} + \phi_{/x}^T \lambda = \mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, t)$$

$$\phi(\mathbf{x}, t) = 0$$

- **Fundamental problem:**
 - integration of Initial Value Problem (IVP) in time
 - also in Hard Real-Time (by way of RTAI, RT-Net & Scicos) for Hardware-In-the-Loop (HIL) simulations
- **Static analysis as degeneration of IVP dynamic analysis:**
 - momentum and momenta moment definitions omitted
 - only gravity is considered
 - system determination must be provided by kinematic constraints and deformable components
- **Kinematic analysis as degeneration of IVP dynamic analysis:**
 - inertia elements omitted
 - system determination must be provided by kinematic constraints
 - deformable components can act as “regularization”

Problems

$\mathbf{K}' = \mathbf{I}$: Moore-Penrose pseudo-inverse

Experimental inverse dynamics problem

- inverse kinematics (allows under-constrained systems):

same matrix!

$$\begin{array}{l} \left[\begin{array}{cc} \mathbf{K}' & \phi_{/x}^T \\ \phi_{/x} & 0 \end{array} \right] \begin{cases} \Delta \mathbf{x} \\ \Delta \lambda_0 \end{cases} = \begin{cases} 0 \\ -\phi(\mathbf{x}, t) \end{cases} \\ \left[\begin{array}{cc} \mathbf{K}' & \phi_{/x}^T \\ \phi_{/x} & 0 \end{array} \right] \begin{cases} \Delta \dot{\mathbf{x}} \\ \Delta \lambda_1 \end{cases} = \begin{cases} 0 \\ \mathbf{b}(\mathbf{x}, t) \end{cases} \\ \left[\begin{array}{cc} \mathbf{K}' & \phi_{/x}^T \\ \phi_{/x} & 0 \end{array} \right] \begin{cases} \Delta \ddot{\mathbf{x}} \\ \Delta \lambda_2 \end{cases} = \begin{cases} 0 \\ \mathbf{b}'(\mathbf{x}, \dot{\mathbf{x}}, t) \end{cases} \end{array}$$

nonlinear

linear

linear

- the RHS contains the desired motion and its derivatives
- the (regularized) static analysis provides the kinematic inversion

$$\phi_{/x}^T \Delta \lambda = \mathbf{f}'(\mathbf{x}, \dot{\mathbf{x}}, \ddot{\mathbf{x}}, t)$$

linear

- **Direct eigenanalysis (eXperimental)**
 - **issues with constraints formulation (mainly with rotations)**
 - **issues with equations implementation (matrices not available for some elements; e.g. aerodynamics)**
- **Future development: relative coordinate frame dynamics**
 - **imposed frame motion: modifications only to RHS inertia elems**
 - **instrumental for many helicopter rotor/wind turbine dynamics problems**

Arbitrary motion description

- **Mechanical degrees of freedom:**
 - structural node positions in the absolute reference frame
 - structural node orientations with respect to the absolute frame (but in updated form...)
- Kinematics is always written with respect to the absolute frame
- Newton-Euler equations are written in the absolute frame
 - moment equilibrium (Euler) equations are written with respect to the (moving) nodes
- Special elements may introduce further approximations
 - e.g. Component Mode Synthesis (CMS) element

Arbitrary motion description

Orientation handling:

- orientation variables: Cayley-Gibbs-Rodrigues parameters
- orientation matrix:

$$\mathbf{R} = \mathbf{R}(\mathbf{g})$$

- orthonormality:

$$\mathbf{R}^T = \mathbf{R}^{-1}$$

- derivative:

$$\dot{\mathbf{R}} \mathbf{R}^T = \boldsymbol{\omega} \times = (\mathbf{G}(\mathbf{g}) \dot{\mathbf{g}}) \times$$

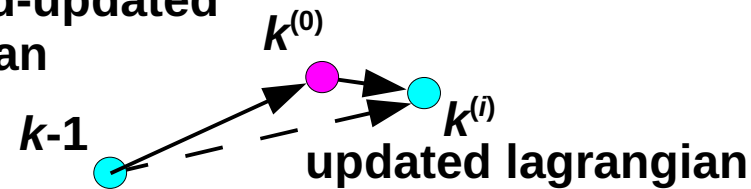
- incremental approach from step k to $k+1$ to eliminate the singularity issue that arises from orientation parameters (increments must be small anyway for accuracy):

$$\mathbf{R}_k = \mathbf{R}(\mathbf{g}_k) \mathbf{R}_{k-1}$$

Arbitrary motion description

- **Orientation handling:**
 - the actual orientation variables are the Cayley-Gibbs-Rodrigues parameters relative to the correction phase of each time step
 - k : time step counter
 - i : correction iteration counter (0: predicted value)

“updated-updated”
lagrangian



- **Orientation matrix:**

$$\mathbf{R}_k^{(i)} = \mathbf{R}(\mathbf{g}_\Delta^{(i)}) \mathbf{R}_k^{(0)}$$

- **Derivative:**

$$\dot{\mathbf{R}}_k^{(i)} (\mathbf{R}_k^{(i)})^T = \omega_k^{(i)} \times = (\mathbf{R}(\mathbf{g}_\Delta^{(i)}) \omega_k^{(0)}) \times + (\mathbf{G}(\mathbf{g}_\Delta^{(i)}) \dot{\mathbf{g}}_\Delta^{(i)}) \times$$

Arbitrary motion description

- **Incremental orientation from previous step:**
 - **Orientation parameters order of magnitude:**

$$\mathbf{g} \sim \mathbf{O}(\|\omega\| \Delta t)$$

- **Incremental orientation from prediction:**
 - **Orientation parameters order of magnitude:**

$$\mathbf{g}_{\Delta} \sim \mathbf{O}(\Delta t^{n+1})$$

where n is the min between the order of the predictor and that of the integration method (MBDyn: 3 and 2, respectively)

- **As a consequence:** $\mathbf{R}(\mathbf{g}_{\Delta}) \approx \mathbf{I}$

$$\mathbf{G}(\mathbf{g}_{\Delta}) \approx \mathbf{I}$$

(only in Jacobian
matrix, of course!)

$$\dot{\mathbf{G}}(\mathbf{g}_{\Delta}) \approx 0$$

Deformable components

- **Lumped deformable components**
 - rod (1D)
 - linear, angular components (3D)
 - linear & angular component (6D)

- **Intrinsic, composite-ready Finite-Volume beam element**
 - arbitrary constitutive law
 - piezoelectric constitutive law
 - aerodynamic (strip-theory with inflow model) beam element

- **Component Mode Synthesis (CMS)**
 - attached to a floating frame (a multibody node) for rigid body motion
 - linear state-space representation of unsteady aerodynamics

$$\dot{\mathbf{x}} = \mathbf{A} \mathbf{x} + \mathbf{B} \mathbf{q}_s$$

$$\mathbf{f}_a = \mathbf{q} (\mathbf{C} \mathbf{x} + \mathbf{D}_0 \mathbf{q}_s + (b/V) \mathbf{D}_1 \dot{\mathbf{q}}_s + (b/V)^2 \mathbf{D}_2 \ddot{\mathbf{q}}_s)$$

Deformable components

Lumped deformable components (3D, 6D):

$$\theta = \mathbf{ax}(\exp^{-1}(\mathbf{R}_a^T \mathbf{R}_b))$$

$$\mathbf{m} = \mathbf{R}(\xi \theta) \tilde{\mathbf{m}}(\theta)$$

- **Attached form:**
 - **constitutive properties referred to either of the connected nodes**
$$\xi = 0, \xi = 1$$
- **Intrinsic form (invariant: $\xi = 1/2$):**
 - **constitutive properties referred to a floating reference frame**
 - **intrinsically handles geometrical nonlinearity related to rotations**
 - **correctly captures bending-torsion buckling behavior**
 - **essential for anisotropic deformable components**

The intrinsic/invariant form is unique to MBDyn.

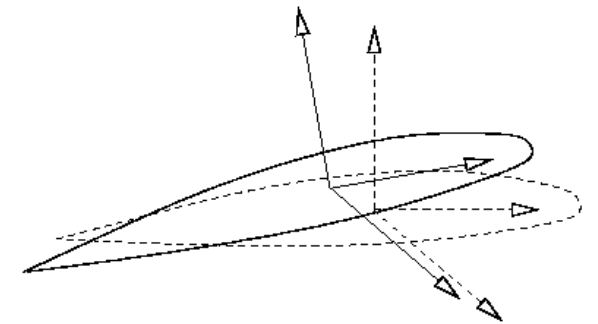
It resulted from a work for Hutchinson CdR

Deformable components

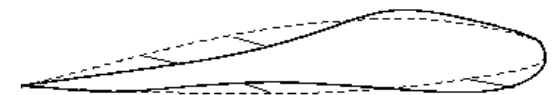
Intrinsic, composite-ready beam

- Topology:
 - 1D reference line \mathbf{p} , 1D reference structure \mathbf{R}
 - 2D section characterization

reference motion: \mathbf{p}, \mathbf{R}

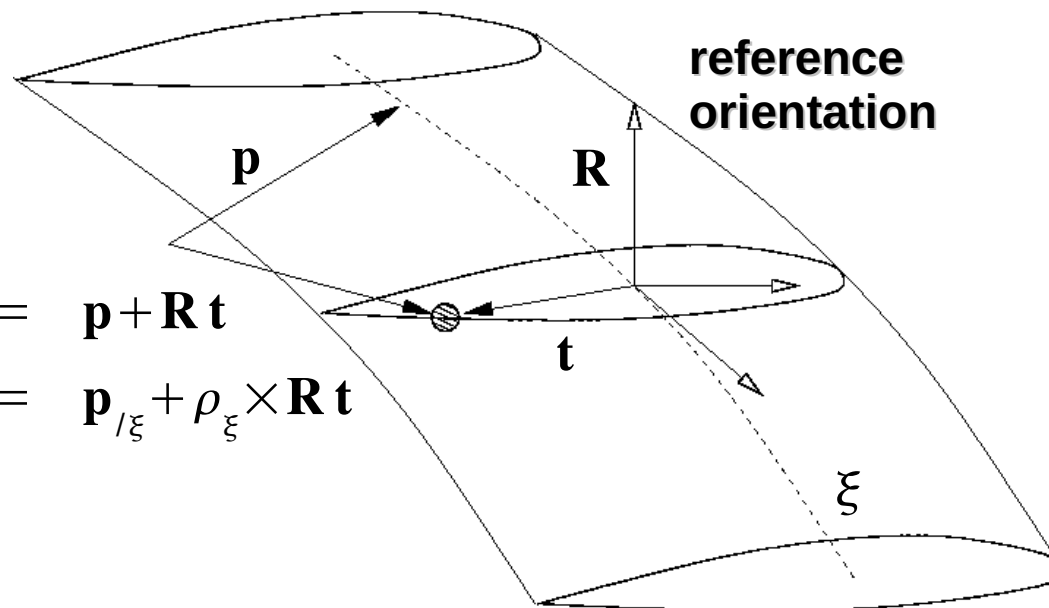


+



warping

reference line



$$\mathbf{x} = \mathbf{p} + \mathbf{R} \mathbf{t}$$

$$\mathbf{x}_{/\xi} = \mathbf{p}_{/\xi} + \rho_{\xi} \times \mathbf{R} \mathbf{t}$$

Deformable components

Intrinsic, composite-ready beams

- strain measure:

$$\nu = \mathbf{R}^T \mathbf{p}_{/\xi} - \mathbf{R}_0^T \mathbf{p}_{0/\xi}$$

$$\kappa = \mathbf{R}^T \rho_{\xi} - \mathbf{R}_0^T \rho_{\xi 0}$$

- equilibrium (from VWP):

$$\mathbf{f}_{/\xi} = \tau$$

$$\mathbf{m}_{/\xi} + \mathbf{p}_{/\xi} \times \mathbf{f} = \mu$$

- constitutive properties:

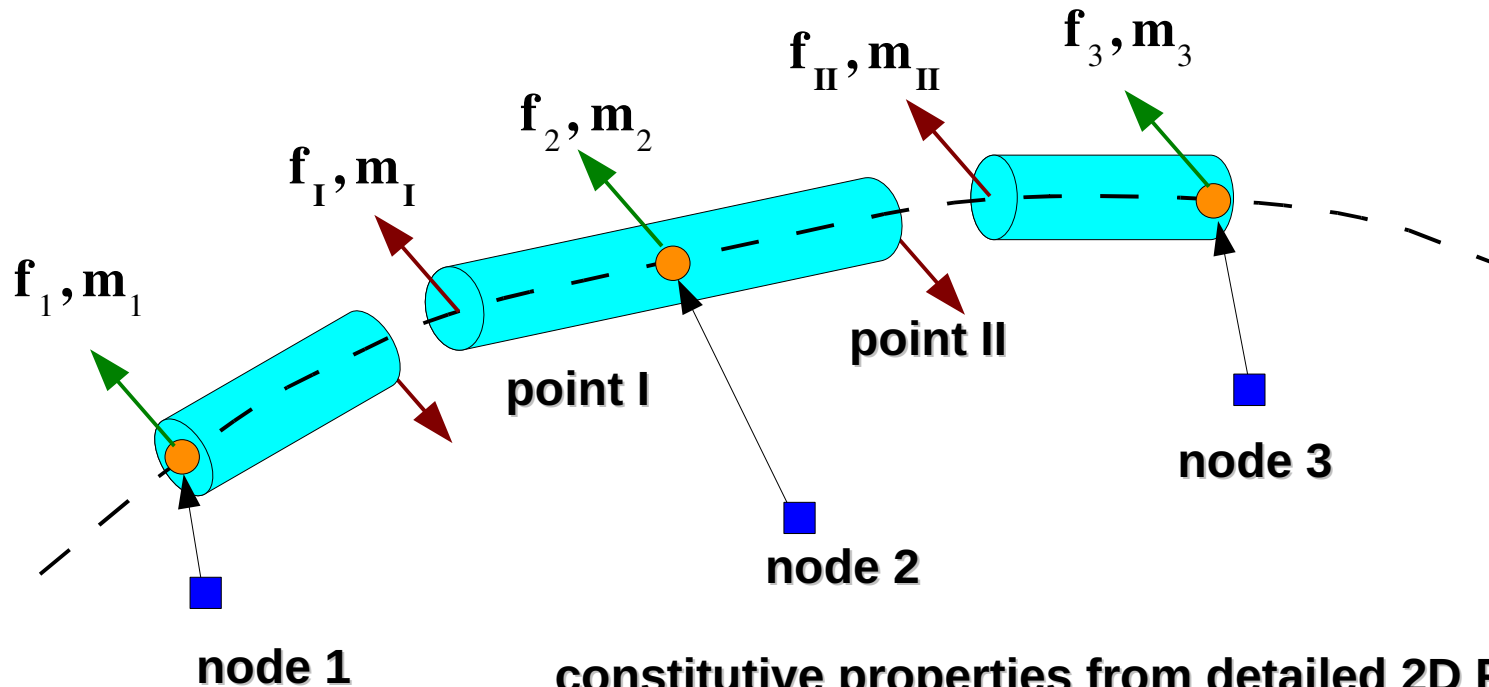
$$\mathbf{f} = \mathbf{f}(\nu, \kappa)$$

$$\mathbf{m} = \mathbf{m}(\nu, \kappa)$$

Deformable components

Intrinsic, composite-ready beams: 3-node discretization

- Finite Volume approach: equilibrium of finite portions of beam
- internal forces function of node kinematics thru constitutive laws
- warping goes into constitutive properties computation



Solving the problem

Numerical integration

- **implicit, (quasi-)L stable 2 step algorithm**

$$\mathbf{y}_k = \mathbf{a}_1 \mathbf{y}_{k-1} + \mathbf{a}_2 \mathbf{y}_{k-2} + \Delta t (\mathbf{b}_0 \dot{\mathbf{y}}_k + \mathbf{b}_1 \dot{\mathbf{y}}_{k-1} + \mathbf{b}_2 \dot{\mathbf{y}}_{k-2})$$

- **tunable algorithmic dissipation: asymptotic spectral radius $1 \rightarrow 0$**
 - **asymptotic spectral radius = 0: 2nd order BDF**
 - **“optimal” dissipation: spectral radius ~ 0.6**
- **second-order accurate, with third-order accurate predictor**
- **variable time step**
- **not ideal for non-smooth problems (multi-step)**
- **different integrators can be used; new ones can be implemented**

Solving the problem

- Prediction:**

$$\dot{\mathbf{y}}_k^{(0)} = (\mathbf{m}_1 \mathbf{y}_{k-1} + \mathbf{m}_2 \mathbf{y}_{k-2}) / \Delta t + \mathbf{n}_1 \dot{\mathbf{y}}_{k-1} + \mathbf{n}_2 \dot{\mathbf{y}}_{k-2}$$

$$\mathbf{y}_k^{(0)} = \mathbf{a}_1 \mathbf{y}_{k-1} + \mathbf{a}_2 \mathbf{y}_{k-2} + \Delta t (\mathbf{b}_0 \dot{\mathbf{y}}_k^{(0)} + \mathbf{b}_1 \dot{\mathbf{y}}_{k-1} + \mathbf{b}_2 \dot{\mathbf{y}}_{k-2})$$

- Correction iteration:**

$$\mathbf{f}_{/\dot{\mathbf{y}}} \Delta \dot{\mathbf{y}}^{(i)} + \mathbf{f}_{/y} \Delta \mathbf{y}^{(i)} = -\mathbf{f}(\dot{\mathbf{y}}_k^{(i-1)}, \mathbf{y}_k^{(i-1)}, \mathbf{t}_k)$$

but

$$\Delta \mathbf{y}^{(i)} = \Delta t \mathbf{b}_0 \Delta \dot{\mathbf{y}}^{(i)}$$

the problem becomes algebraic

$$(\mathbf{f}_{/\dot{\mathbf{y}}} + \Delta t \mathbf{b}_0 \mathbf{f}_{/y}) \Delta \dot{\mathbf{y}}^{(i)} = -\mathbf{f}(\dot{\mathbf{y}}_k^{(i-1)}, \mathbf{y}_k^{(i-1)}, \mathbf{t}_k)$$

$$\dot{\mathbf{y}}_k^{(i)} = \dot{\mathbf{y}}_k^{(i-1)} + \Delta \dot{\mathbf{y}}^{(i)}$$

$$\mathbf{y}_k^{(i)} = \mathbf{y}_k^{(i-1)} + \Delta t \mathbf{b}_0 \Delta \dot{\mathbf{y}}^{(i)}$$

Solving the problem

Model assembly

- model could be input incorrectly
- initial values of the state (position, velocity, reactions) are needed
- this task might not be trivial
- initial state values must comply with constraints:

$$\phi(\mathbf{x}_0, t_0) = 0$$

$$\dot{\phi}(\mathbf{x}_0, t_0) = 0$$

- a dummy static nonlinear problem is solved (regularization):

$$\mathbf{K}'(\mathbf{x} - \mathbf{x}_0) + \phi_{/x}^T \lambda' = \mathbf{f}'$$

$$\mathbf{C}'(\dot{\mathbf{x}} - \dot{\mathbf{x}}_0) + \phi_{/x}^T \mu' = \dot{\mathbf{f}}'$$

$$\phi(\mathbf{x}, t_0) = 0$$

$$\dot{\phi}(\mathbf{x}, t_0) = 0$$

Solving the problem

Solution initialization (so-called “derivatives”)

- **explicit problem:**

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, t)$$

- **implicit problem:**

$$0 = \mathbf{f}(\mathbf{y}, \dot{\mathbf{y}}, t)$$

- **modified correction phase to initialize solution:**

$$(\mathbf{f}_{/\dot{\mathbf{y}}} + c \mathbf{f}_{/\mathbf{y}}) \Delta \dot{\mathbf{y}}^{(i)} = -\mathbf{f}(\dot{\mathbf{y}}_0^{(i-1)}, \mathbf{y}_0, t_0)$$

$$\dot{\mathbf{y}}_0^{(i)} = \dot{\mathbf{y}}_0^{(i-1)} + \Delta \dot{\mathbf{y}}^{(i)}$$

$$\mathbf{y}_0 = \mathbf{y}_0$$

- **convergence no longer quadratic, but saves lots of code duplication**
- **Setting $c=0$ might not work (problem can be structurally singular)**

Extracting useful information

- Detailed analysis requires detailed models, but...
- excessive details endanger the chance to extract useful information
- Proper Orthogonal Decomposition allows to extract information from redundant measures

- Consider a set of N measurements \mathbf{X} for n time steps; their SVD:

$$\mathbf{X}^T (\in \mathbb{R}^{n \times N}) = \mathbf{U} \Sigma \mathbf{V}^T$$

- The singular values allow to determine the m most relevant signals

$$\mathbf{X}_{(1:m,n)}^T = \mathbf{U}_{n,1:m} \Sigma_{1:m,1:m} \mathbf{V}_{N,1:m}^T$$

- Note that

$$\begin{aligned} \mathbf{X}^T \mathbf{X} &= \mathbf{U} \Sigma^2 \mathbf{U}^T & \mathbf{X} \mathbf{X}^T &= \mathbf{V} \Sigma^2 \mathbf{V}^T \\ \mathbf{U}^T \mathbf{X}^T &= \Sigma \mathbf{V}^T & \mathbf{X}^T \mathbf{V} &= \mathbf{U} \Sigma \end{aligned}$$

- This allows to efficiently compute the singular values and the POMs

Extracting useful information

- **The POMs can be used to identify a transition matrix**

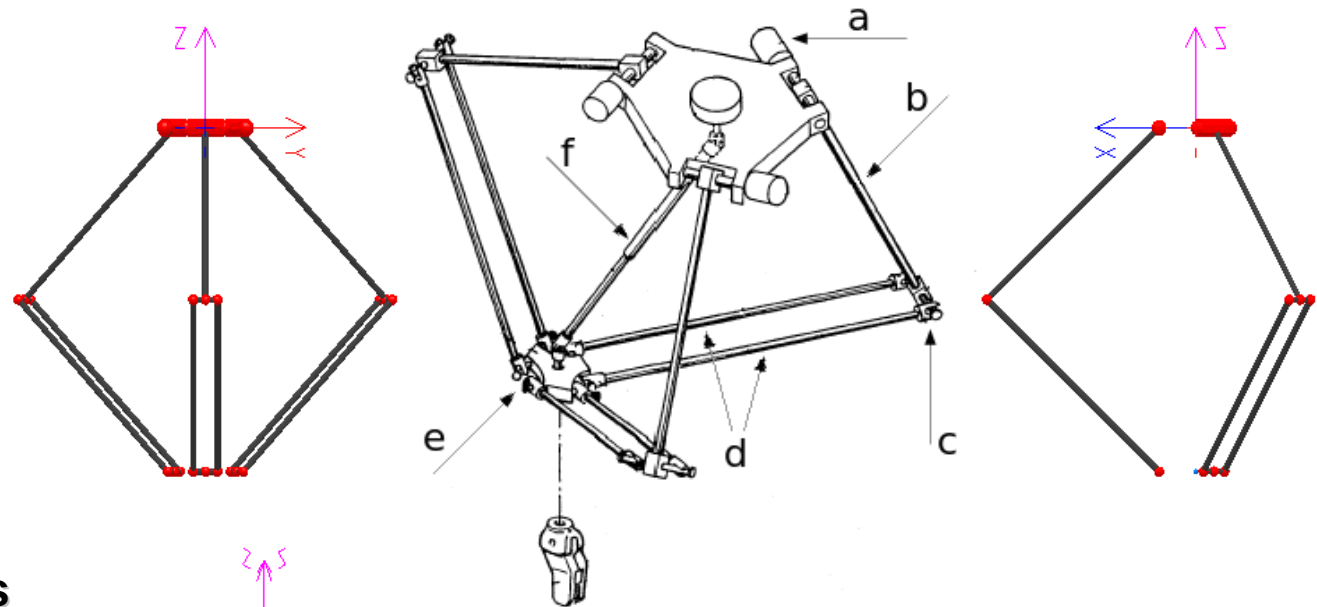
$$\mathbf{X}^{(k+1)} = \Phi \mathbf{X}^{(k)}$$

- **If \mathbf{X} contains the free response of the system, the transition matrix allows to estimate the relevant eigenvalues (AR model)**
- **More sophisticated system identification techniques can be used**
- **A technique based on covariance estimates from time histories has been recently proposed; works for:**
 - **free response**
 - **forced response**
 - **unmeasured forced response**

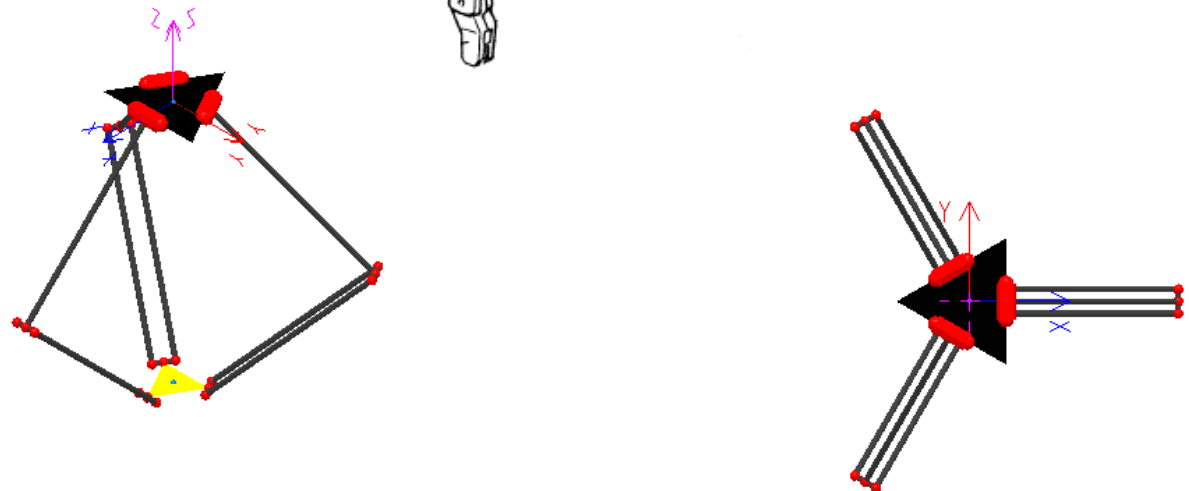
- [1] G. Quaranta, P. Masarati, and P. Mantegazza: “Continuous-Time Covariance Approaches for Modal Analysis”, Journal of Sound and Vibration, Vol.310/1-2 pp. 287-312, 5 February 2008.
- [2] G. Quaranta, P. Masarati, and P. Mantegazza: “Assessing the Local Stability of Periodic Motions for Large Multibody Nonlinear Systems Using POD”, Journal of Sound and Vibration, Vol 271/3-5, pp. 1015-1038, 2004.

Examples of multibody modeling with MBDyn

Robotics: delta robot

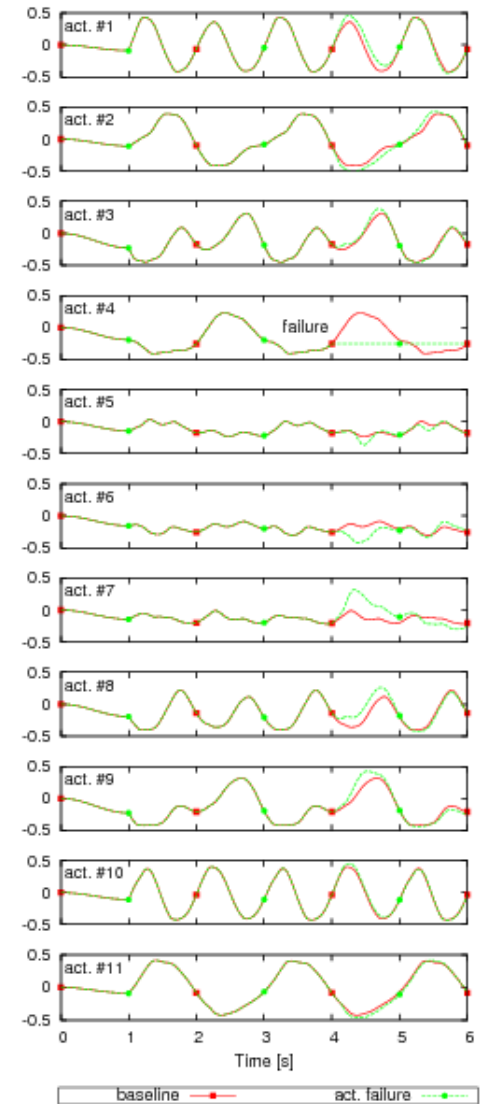
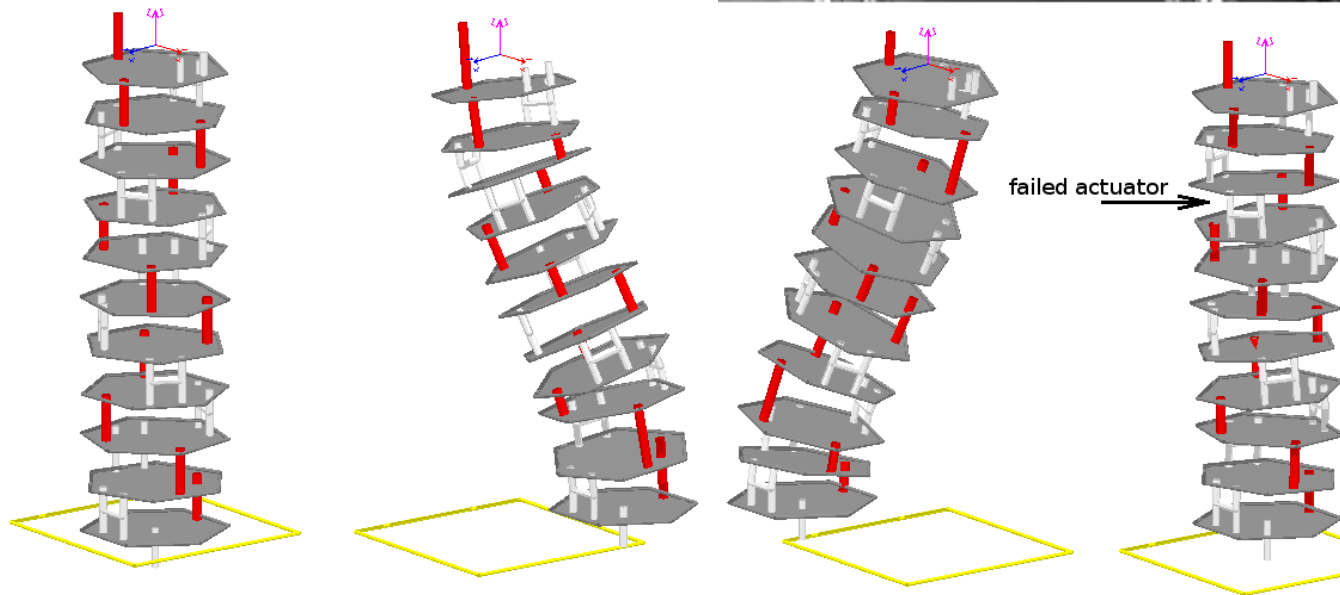
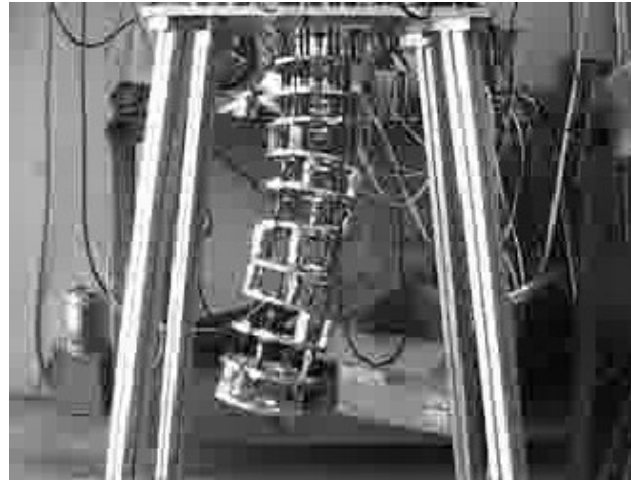


inverse dynamics for computed torque control



Examples of multibody modeling with MBDyn

Robotics:
biomimetic robot
real-time motion planning
by inverse kinematics
with fault detection

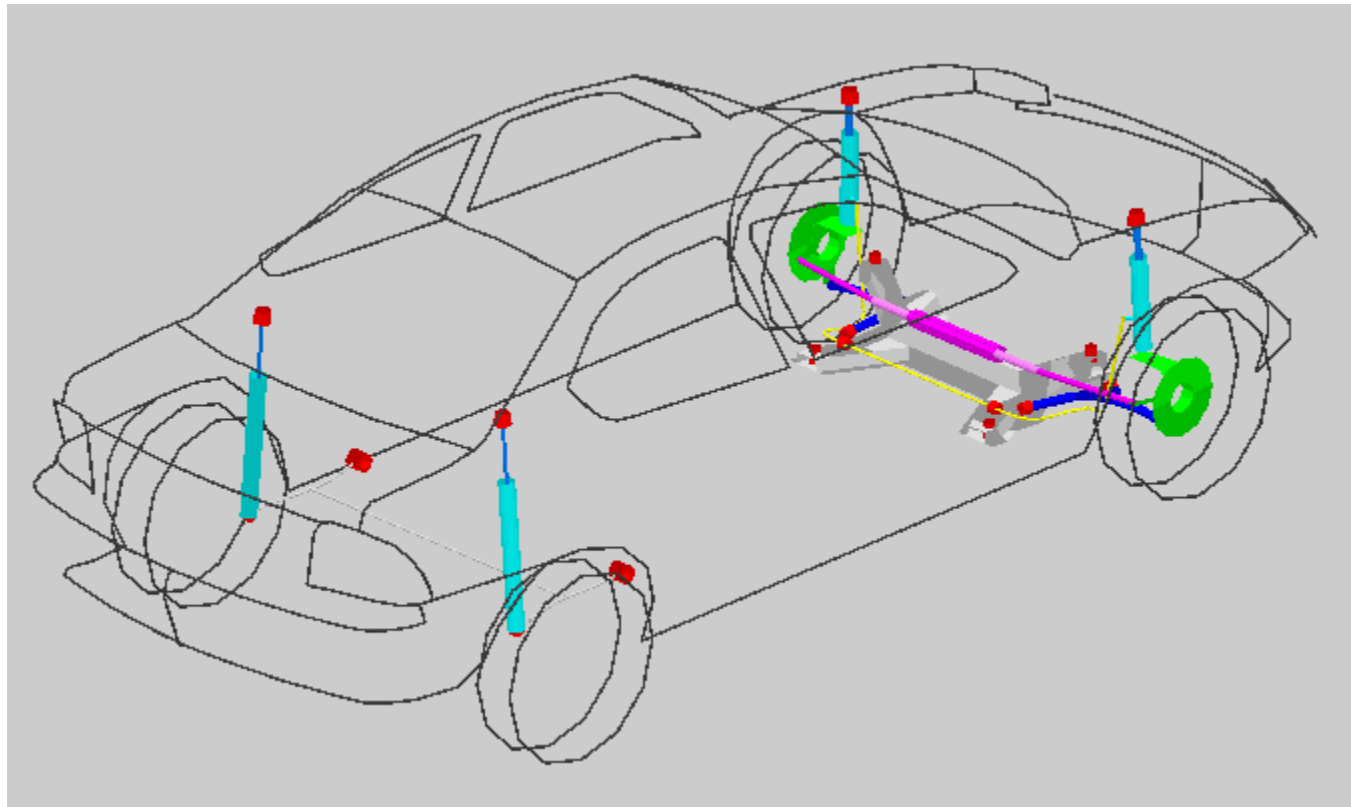


Industrial processes:

- **simulation of automotive components assembly (car brake pipe) to:**
 - **check stresses introduced during assembly**
 - **check loads on supports introduced during assembly**
 - **check interference with other parts during assembly**
 - **check interference with other parts during operation**
- **the model has been developed by a rubber manufacturer (Hutchinson CdR)**
- **it is used for product design and certification**
- **it required the development of specific features for solution partitioning, which are now part of the standard MBDyn (the “hints”)**

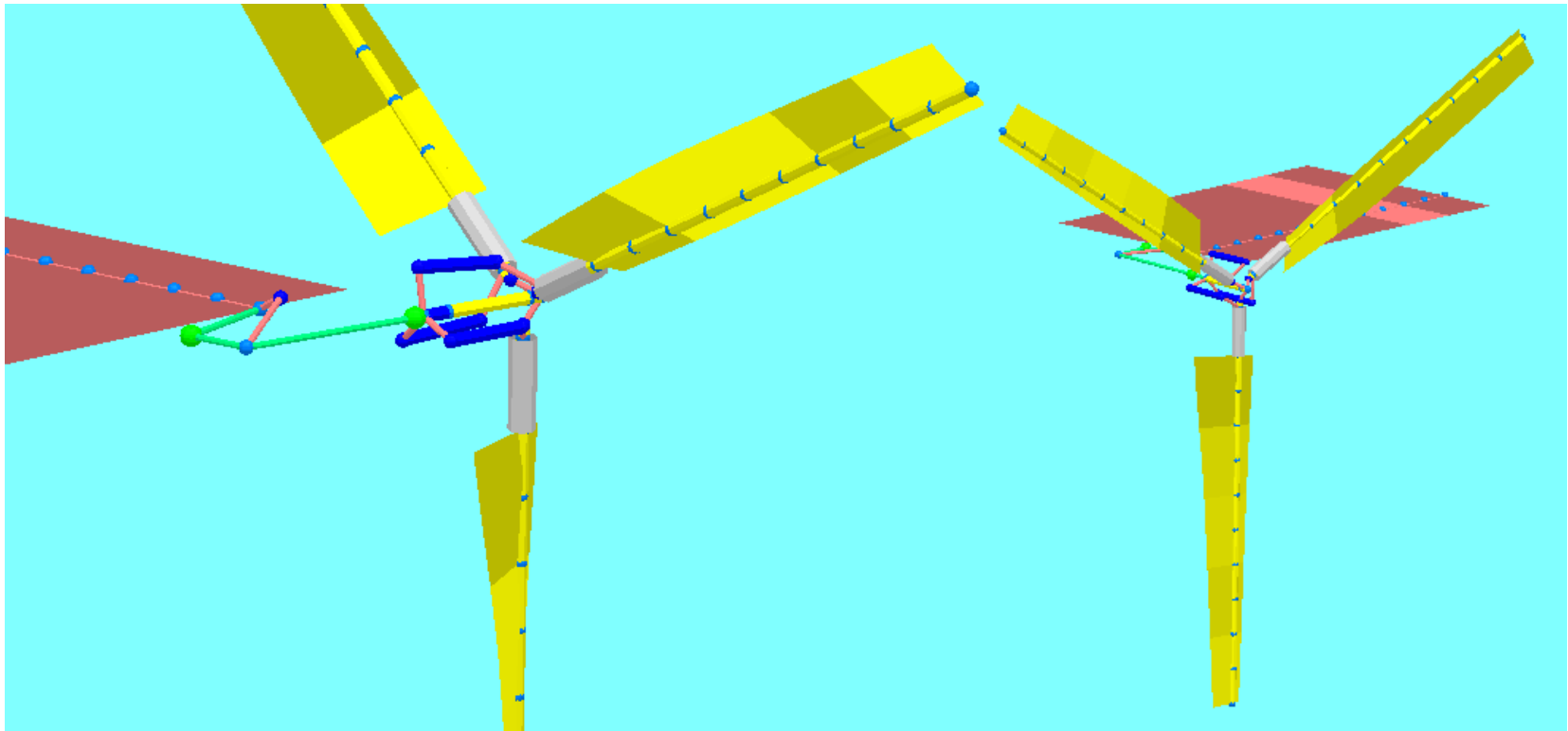
Automotive: mechanical modeling of suspensions

purpose: determine loads in rubber bushings and other components



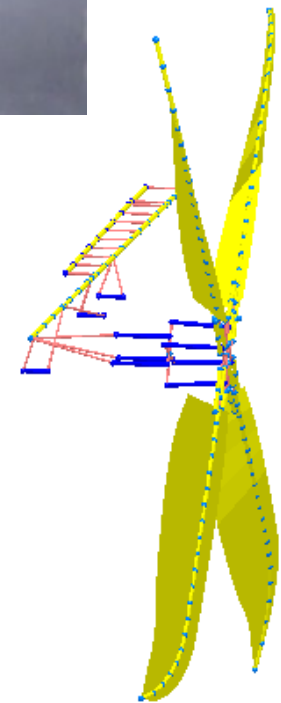
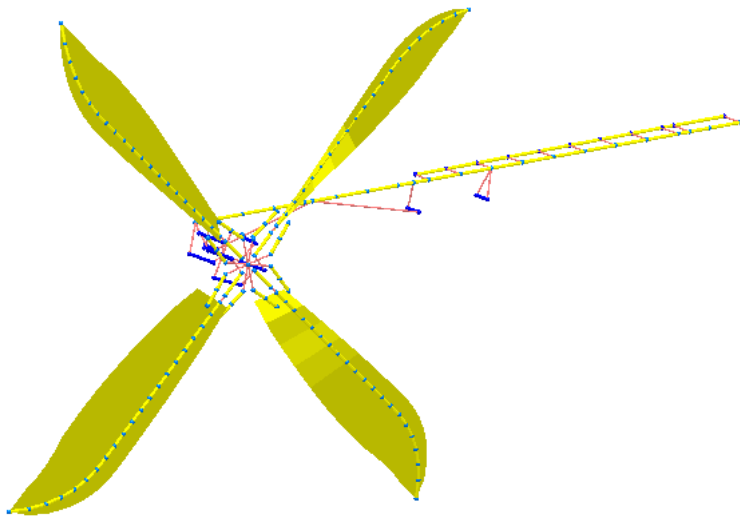
Rotorcraft dynamics and aeroservoelasticity:

- WRATS (NASA/Army) tiltrotor aeromechanics



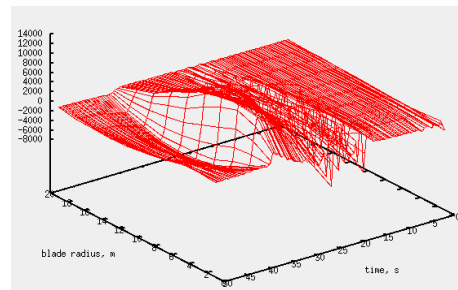
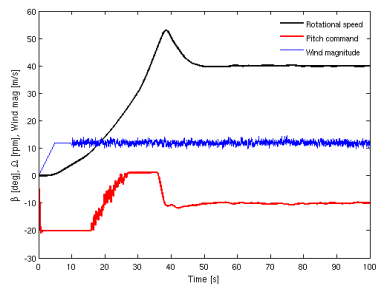
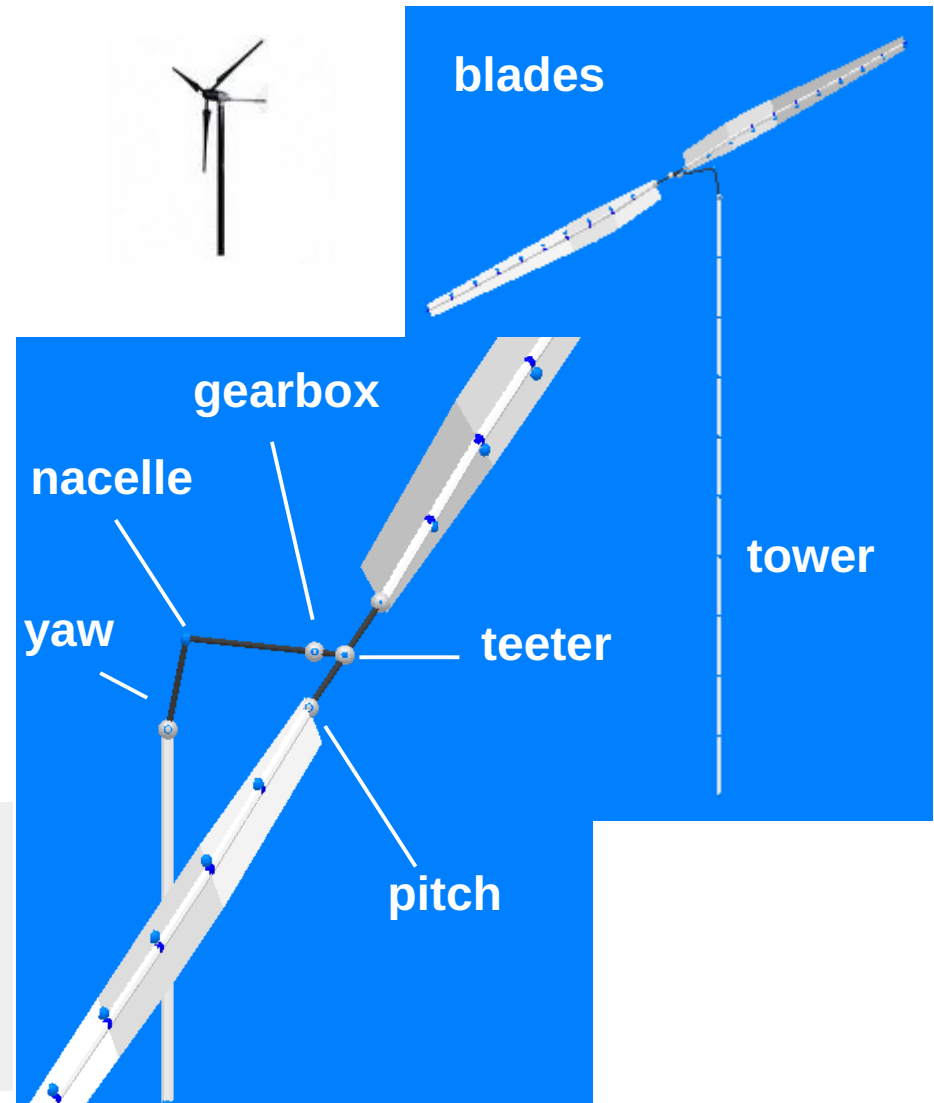
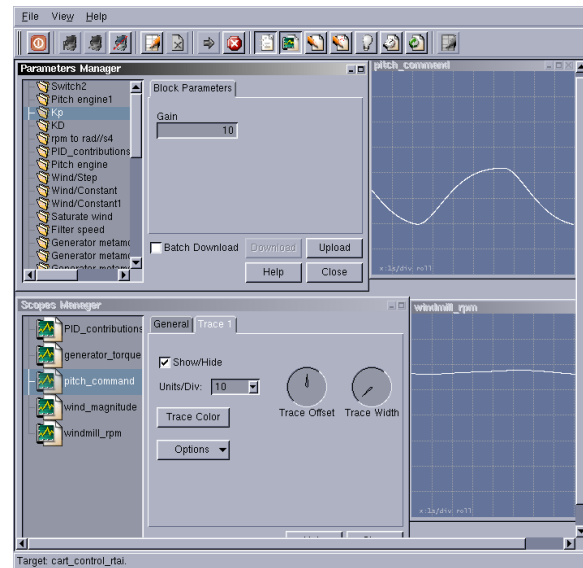
Rotorcraft dynamics and aeroservoelasticity:

- ERICA (AgustaWestland) tiltrotor aeromechanics (ADYN, NICETRIP EU 6FP, in cooperation with Eurocopter, DLR, ONERA and more...)



Examples of multibody modeling with MBDyn

Wind-turbine dynamics simulation, also in Real-Time



Future development

- **Multiscale handling of submodels with different dynamics; for example, in the flying helicopter case:**
 - **aircraft flight mechanics (~0 to 5 Hz: very slow)**
 - **airframe & main rotor dynamics (~5 to 40 Hz: intermediate)**
 - **tail rotor dynamics (~25 to >100 Hz: fast)**
- **Interfacing with different domains**
 - **fluid-structure (Lagrangian/Eulerian modeling of work flows)**
 - **structure-structure (dynamic loads to detailed static analysis)**
 - **active control of large deformable systems**
- **Better abstraction/modularization of components/solution phases**
 - **more freedom in model customization**
 - **tight integration into nonlinear structural analysis (structural damage growth under dynamic loads?)**
- **More...**

Future development

- **Development has always been problem (and customer!) driven**
- **Development model:**
 - **research (possibly new) solutions in (possibly new) areas**
 - **procure research grants and contracts in those fields**
 - **never ask funds just for software development**
 - **sell expertise, not just software; as a consequence...**
 - **...software is not the result of a contract: it's a pre-existing tool**
 - **software remains free even when customers are narrow-minded**
 - **when possible, work with open-minded customers :-)**
- **Drawback: cannot stick with tight development plans :-)**

**We are a University: we should not sacrifice our research freedom
to industry schedules**

Documentation and support

- **Freedom and features are not enough: software needs to be usable**
- **Users need:**
 - **documentation**
 - **forums to discuss issues**
 - **issue tracking provisions**
 - **reasonable guarantee of maintenance**
 - **reasonable guarantee of stability across releases**
- **The main guarantee is the freedom of the software:**
 - **in the worst case, one can always fork and go on its own; but...**
 - **... first, better talk to the developers: cooperation saves efforts**

Documentation and support

- **Theory manual:**
 - Incomplete; needs lots of work
- **User manual**
 - available and complete, but probably hard to read; needs improvements
- **Tutorials**
 - available, but reportedly too simple; need work
- **Applications manual**
 - just started
- **Installation manual**
 - available, but incomplete and outdated
- **Mailing lists**
 - available: announce, users, devel
 - right now, the “users” list also serves as issue tracking provision

Documentation and support

- **Another important item that is missing is an automated test suite, that**
 - **can be run automatically after building the software**
 - **allows to check build errors**
 - **allows to check regressions in new releases**
 - **serves as example of modeling functionalities**
- **The rest is underway (always a work in progress)**

Given the nature of the project, contributions are always welcome!

Acknowledgments

- **Paolo Mantegazza (“grandfather” of MBDyn)**
- **Marco Morandini (lots of optimizations, friction, ...)**
- **Giuseppe Quaranta (schur decomposition, FSI, ...)**
- **Alessandro Fumagalli (inverse dynamics, ...)**

- **many Graduate and PhD students**

- **few contributions from outside as well (mostly bug fixes)**

Questions?