Modelling and Simulation of Rigid and Flexible Multibody Systems in Modelica

Tutorial at the Modelica'2011 Conference

Dresden, March 20th, 2011

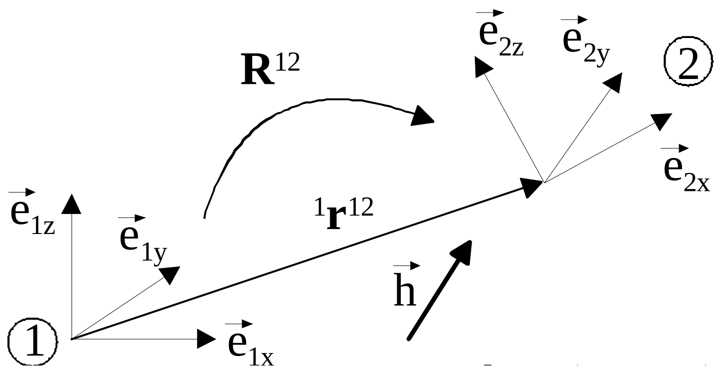
Dr.-Ing. Andreas Heckmann, German Aerospace Center (DLR)

Institute of Robotics and Mechatronics

Contents

Modelica Multibody Basics: Orientation

- Coordinate systems and their orientation

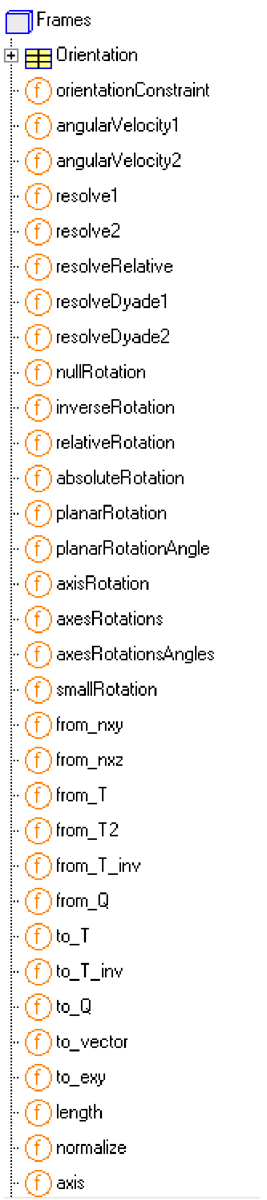



```
import MultiBody.Frames;
 Frames.Orientation R12;
 Real h1[3] "h resolved in frame 1";
 Real h2[3] "h resolved in frame 2";
equation
  h2 = Frames.resolve2(R12, h1); //or
  h1 = Frames.resolve1(R12, h2);
```

- Orientation object R^{12}
  - describes orientation of coordinate system 2 wrt.1
  - holds

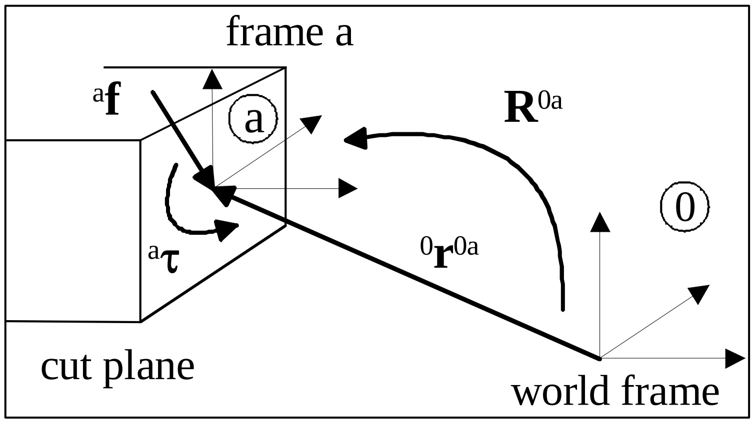
```
Real T[3, 3] "Transformation matrix from world frame to local frame";
SI.AngularVelocity w[3]
  "Absolute angular velocity of local frame, resolved in local frame";
```

  - may be computed using rotation angles or quaternions
- Multibody Lib. contains over 30 functions to operate on orientation objects



Modelica Multibody Basics: Connectors I

- Connectors: the interface to connect components
  - Position is resolved in world frame
  - Forces and torques are resolved in local frame



non-flow !

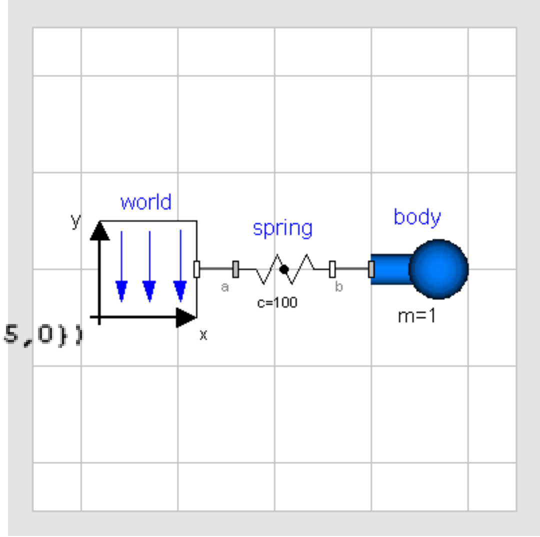
flow !

```
connector Frame
  "Coordinate system fixed to the component with one cut-force and cut-torque (no icon)"
  import SI = Modelica.SIunits;
  SI.Position r_0[3]
    "Position vector from world frame to the connector frame origin, resolved in world frame";
  Frames.Orientation R
    "Orientation object to rotate the world frame into the connector frame";
  flow SI.Force f[3] "Cut-force resolved in connector frame" a;
  flow SI.Torque t[3] "Cut-torque resolved in connector frame";
  a;
end Frame;
```

Modelica Multibody Basics: Connectors II

- Connectors: how they work

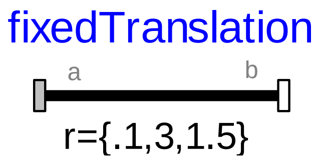
```
model SpringMass
  inner Modelica.Mechanics.MultiBody.World world
    a;
  Modelica.Mechanics.MultiBody.Forces.Spring spring(c=100)
    a;
  a;
  Modelica.Mechanics.MultiBody.Parts.Body body(r_0_start={0,.5,0})
    a;
equation
  connect(world.frame_b, spring.frame_a) a;
  connect(spring.frame_b, body.frame_a) a;
end SpringMass;
```



- Modelica‘s general connections rules
  - non-flow variables are set to be equal, i.e. frames coincide
    - since they represent „some kind of potential“
  - flow variables sum to zero (Kirchhoff‘s current law)
    - since they represent time derivatives of preserved quantities
    - are consequently set to zero if connector is not connected to anything
- see Modelica.UsersGuide.Connectors for a comparison of connectors in various domains

Modelica Multibody Basics: Components I

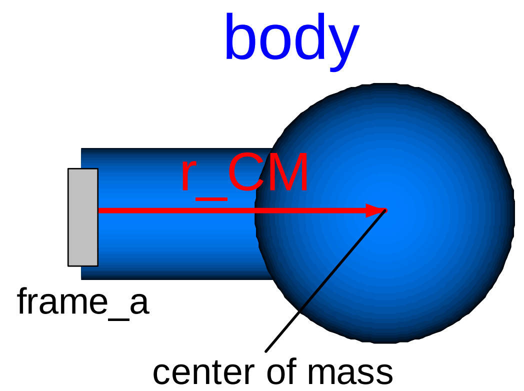
- Kinematics:
  - Component equations provide relations between connector variables on position level
  - MultiBody.Parts.FixedTranslation
    i.e. fixed translation of frame_b with respect to frame_a
  - Tool (e.g. Dymola) differentiates these equations twice for dynamics




```
frame_b.r_0 = frame_a.r_0 + Frames.resolve1(frame_a.R, r);
frame_b.R = frame_a.R;

/* Force and torque balance */
zeros(3) = frame_a.f + frame_b.f;
zeros(3) = frame_a.t + frame_b.t + cross(r, frame_b.f);
```

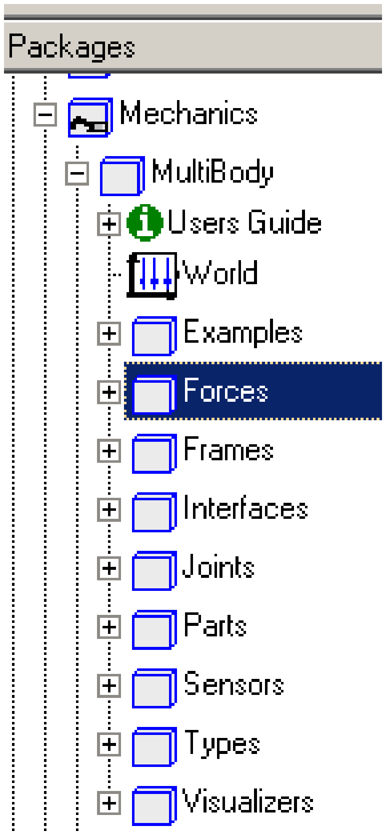
Modelica Multibody Basics: Components II

- Dynamics
  - Newton-Euler equations
  - MultiBody.Parts.Body

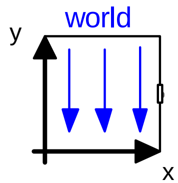



```
// import Modelica.Mechanics.MultiBody.Frames;
// translational kinematic differential equations resolved in local frame_a
v_a = Frames.resolve2(frame_a.R, der(frame_a.r_0));
a_a = der(v_a);

// rotational kinematic differential equations
w_a = Modelica.Mechanics.MultiBody.Frames.angularVelocity2(frame_a.R);
z_a = der(w_a);

// Newton/Euler equations with respect to center of mass
          a_CM = a_a + cross(z_a, r_CM) + cross(w_a, cross(w_a, r_CM));
          f_CM = m*a_CM;
          t_CM = I*z_a + cross(w_a, I*w_a);
     frame_a.f = f_CM;
     frame_a.t = t_CM + cross(r_CM, f_CM);
```

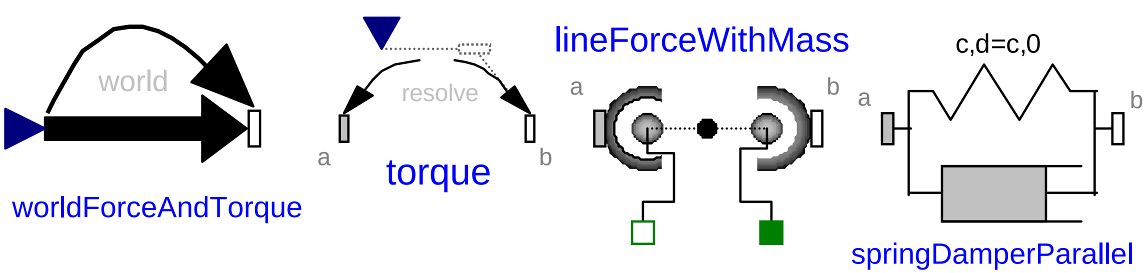
Modelica Multibody Basics: Elementary Components I



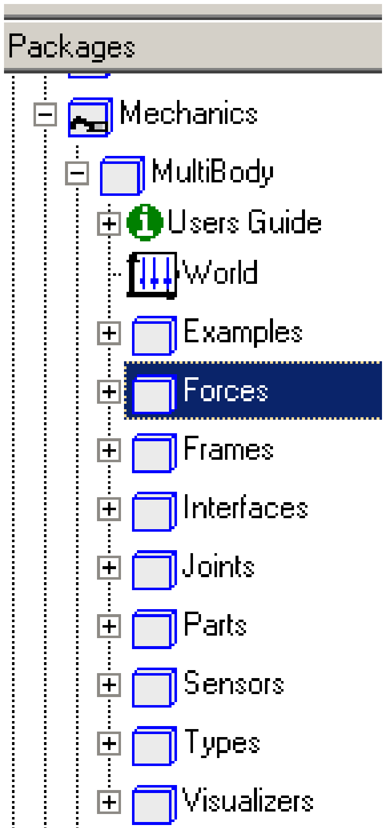
- Modelica.Mechanics.MultiBody.World
  - defines inertial frame, gravity, animation defaults



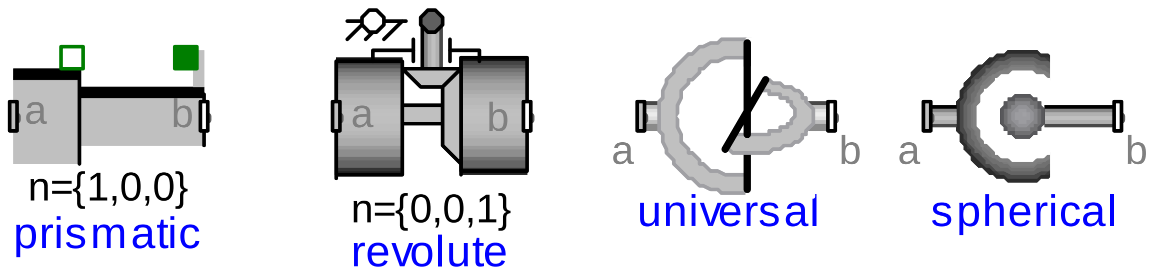
- Modelica.Mechanics.MultiBody.Forces
  - different resolution properties
  - interface to Real input functions and 1D mechanics
  - several spring/damper configurations



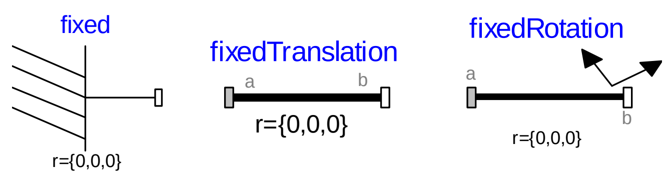
Modelica Multibody Basics: Elementary Components II



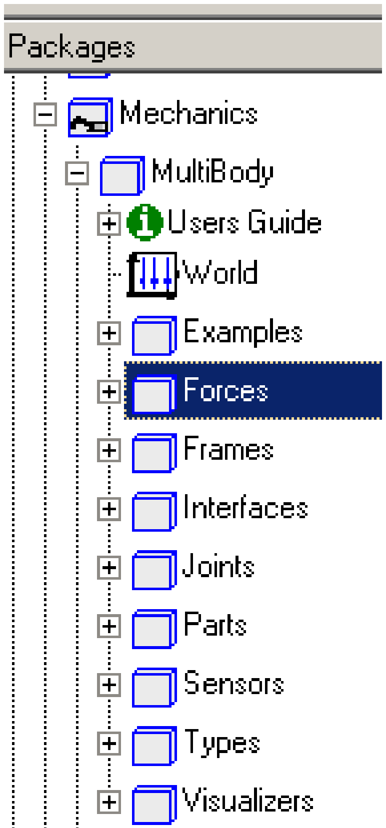
- Modelica.Mechanics.MultiBody.Joints
  - define specific degree of freedom
  - capability to set-up initial configuration
  - interface to/for 1D mechanics and rheonom motion
  - e.g.:



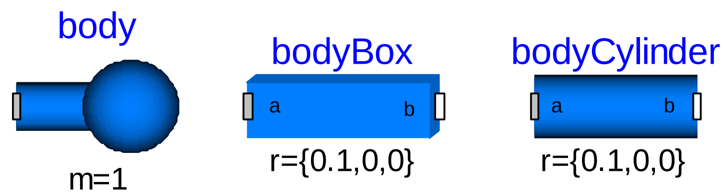
- Modelica.Mechanics.MultiBody.Parts
  - Fixed, FixedTranslation and FixedRotation



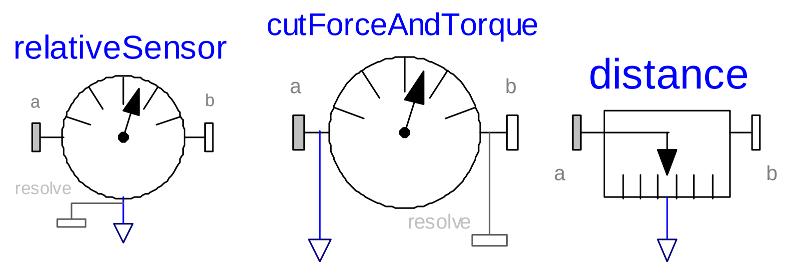
Modelica Multibody Basics: Elementary Components III



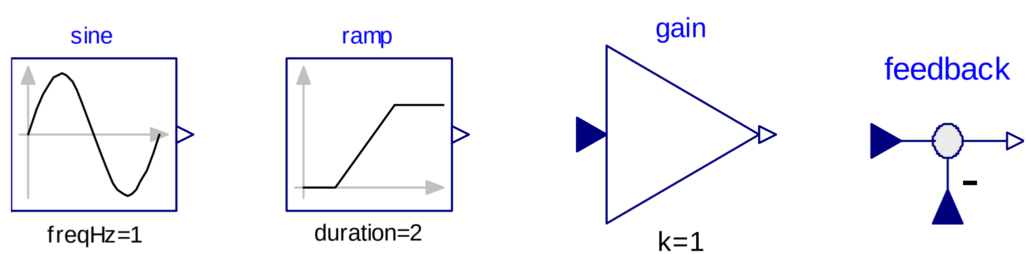
- Modelica.Mechanics.MultiBody.Parts
  - Rigid bodies with predefined geometric shapes



  - Modelica.Mechanics.Multibody.Sensors
    - for control and validation purposes

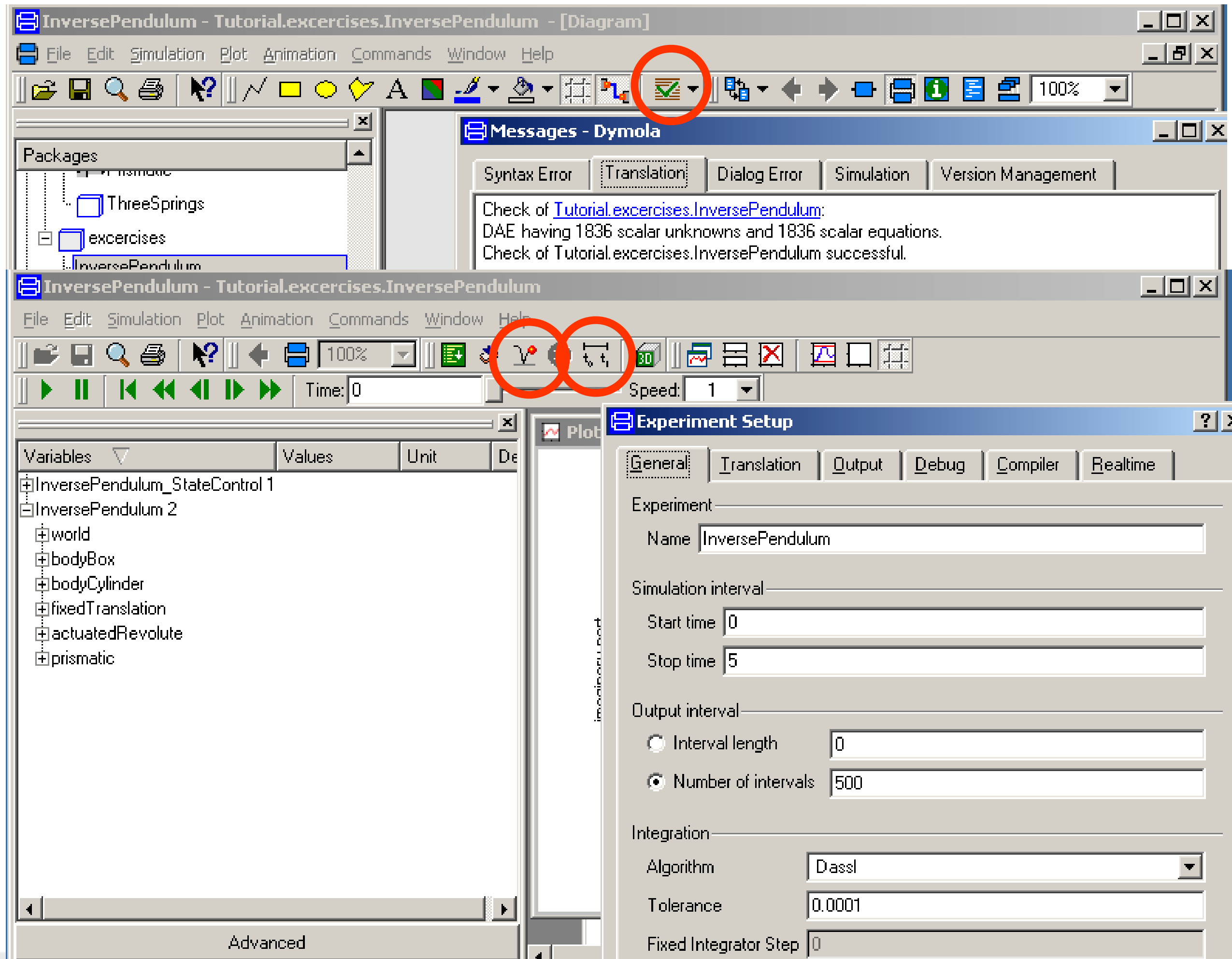


- Modelica.Blocks.Sources + Modelica.Blocks.Math



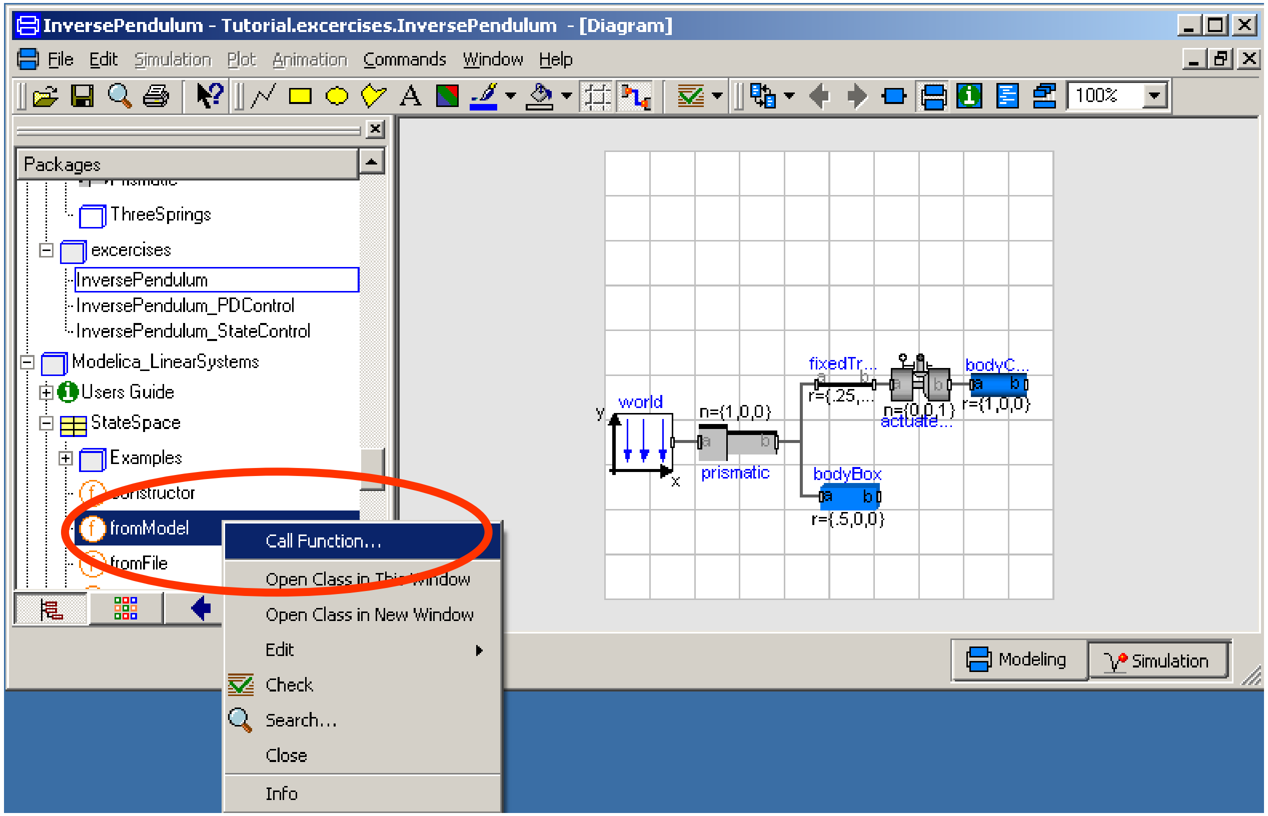
Modelica Multibody Basics: Analysis Methods

- Model check
- Experiment setup, translation and time simulation



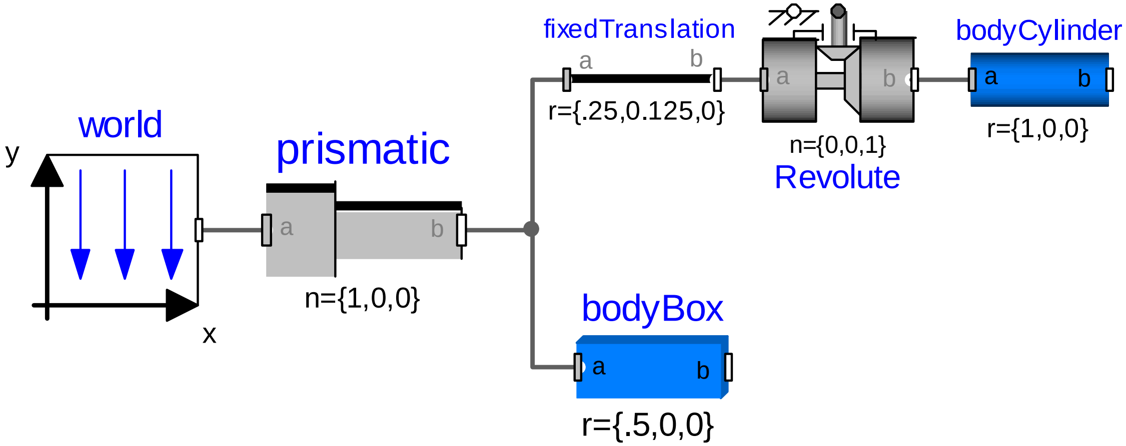
Modelica Multibody Basics: Analysis Methods

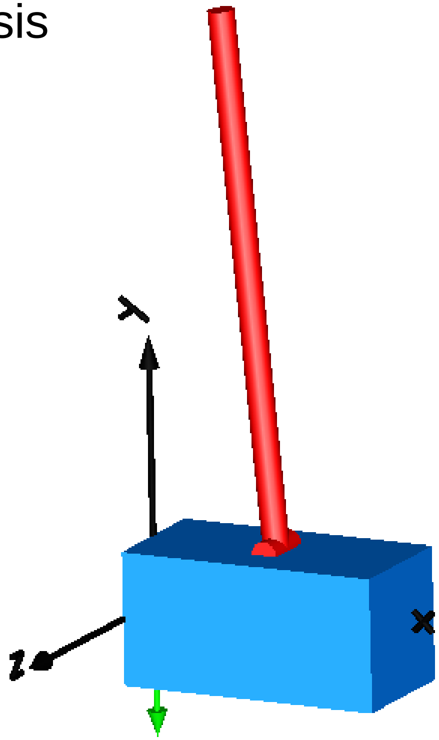
- Model check
- Experiment setup, translation and time simulation
- Eigenvalue analysis
  - Menu: File→Libraries→LinearSystems



Example 1: Control of an inverse pendulum I

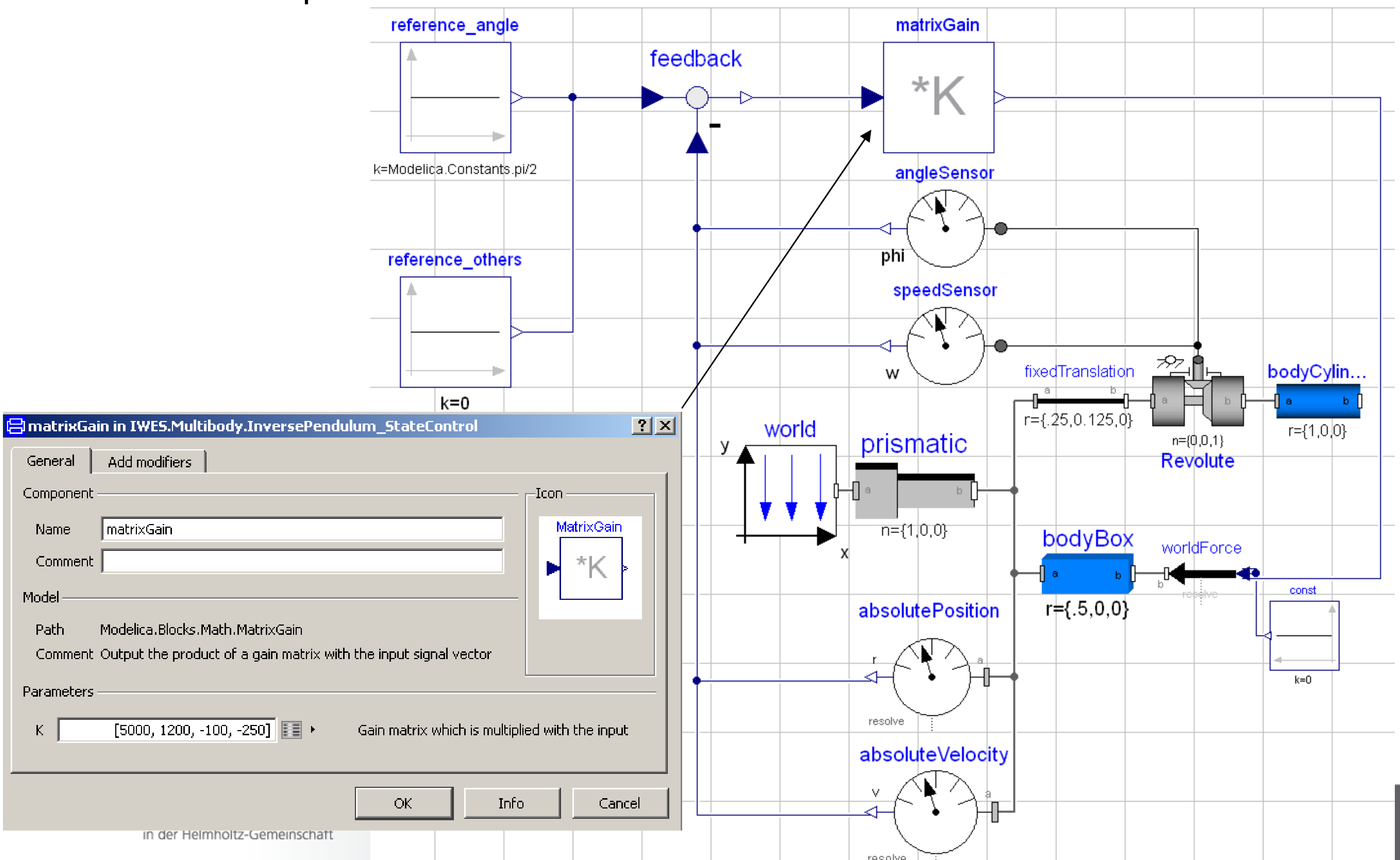
- Initial model
  - Box: 0.5 x 0.25 x 0.25 m
  - actuatedRevolute: phi.start =95°, fixed=true
  - perform time simulation and eigenvalue analysis





Exercise 1: Control of an inverse Pendulum II

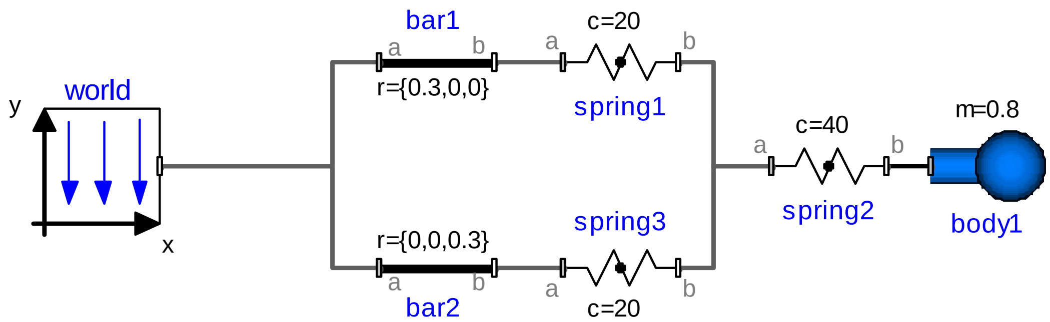
- state space control



Contents

Modelica Multibody Advanced: State selection I

- Joints AND bodies have potential states
  - number of joints is independent from number of bodies
  - an assignment of joints to bodies is not mandatory
  - force elements may be connected to each other
  - e.g.:

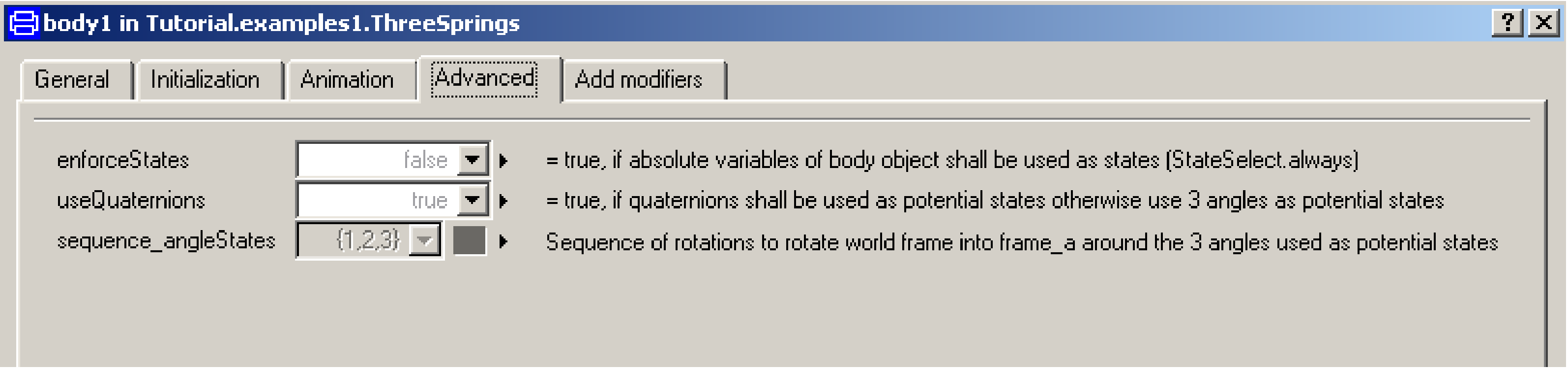


  - here: body coordinates: position, quaternions and their derivatives are used as states

Modelica Multibody Advanced: State selection II

- relative joint coordinates are used as states if possible
  - default: stateSelect = StateSelect.prefer
  - e.g. Multibody.Joints.Prismatic

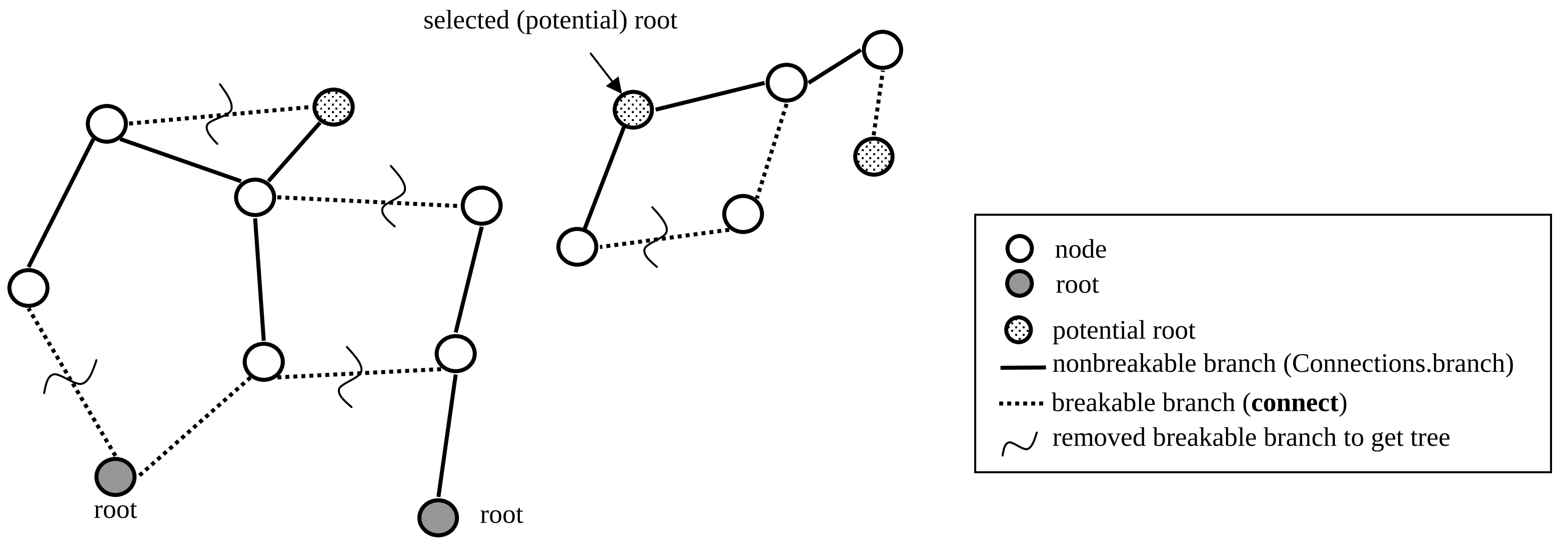

```
final parameter Real e[3]=Modelica.Mechanics.MultiBody.Frames.normalize(n)
  "Unit vector in direction of prismatic axis n";
SI.Position s(stateSelect=if enforceStates then
      StateSelect.always else StateSelect.prefer)
  "Relative distance between frame_a and frame_b";
SI.Velocity v(stateSelect=if enforceStates then StateSelect.always else
      StateSelect.prefer) "First derivative of s (relative velocity)";
```



  - Advanced user may influence state selection directly

Modelica Multibody Advanced: Loops I

- Standard case
  - no specific action by the user is required
  - every connector is one node in the virtual connection graph
  - roots of the virtual connection graph are found, e.g. world.frame_b
  - loops are virtually broken



Modelica Multibody Advanced: Loops I

- Standard case
  - no specific action by the user is required
  - every connector is one node in the virtual connection graph
  - roots of the virtual connection graph are found, e.g. world.frame_b
  - loops are virtually broken
  - the related constraint equations are provided
    ⇒ DAE

$$\mathbf{0} = \boldsymbol{f}(\dot{\boldsymbol{x}}, \boldsymbol{x}, \boldsymbol{y}, t, \ldots) \quad \dim(\boldsymbol{f}) = \dim(\boldsymbol{x}) + \dim(\boldsymbol{y})$$

  - Equations are rearranged to get a sequence for model evaluation (**B**lock **L**ower **T**riangle-partitioning)

$$\begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{array} \begin{array}{c} \begin{array}{ccccc} z_1 & z_2 & z_3 & z_4 & z_5 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array} \Rightarrow \begin{array}{c} \\ f_2 \\ f_4 \\ f_3 \\ f_5 \\ f_1 \end{array} \begin{array}{c} \begin{array}{ccccc} z_1 & z_2 & z_3 & z_4 & z_5 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

Modelica Multibody Advanced: Loops I

- Standard case
  - no specific action by the user is required
  - every connector is one node in the virtual connection graph
  - roots of the virtual connection graph are found, e.g. world.frame_b
  - loops are virtually broken
  - the related constraint equations are provided
    $\Rightarrow$ DAE

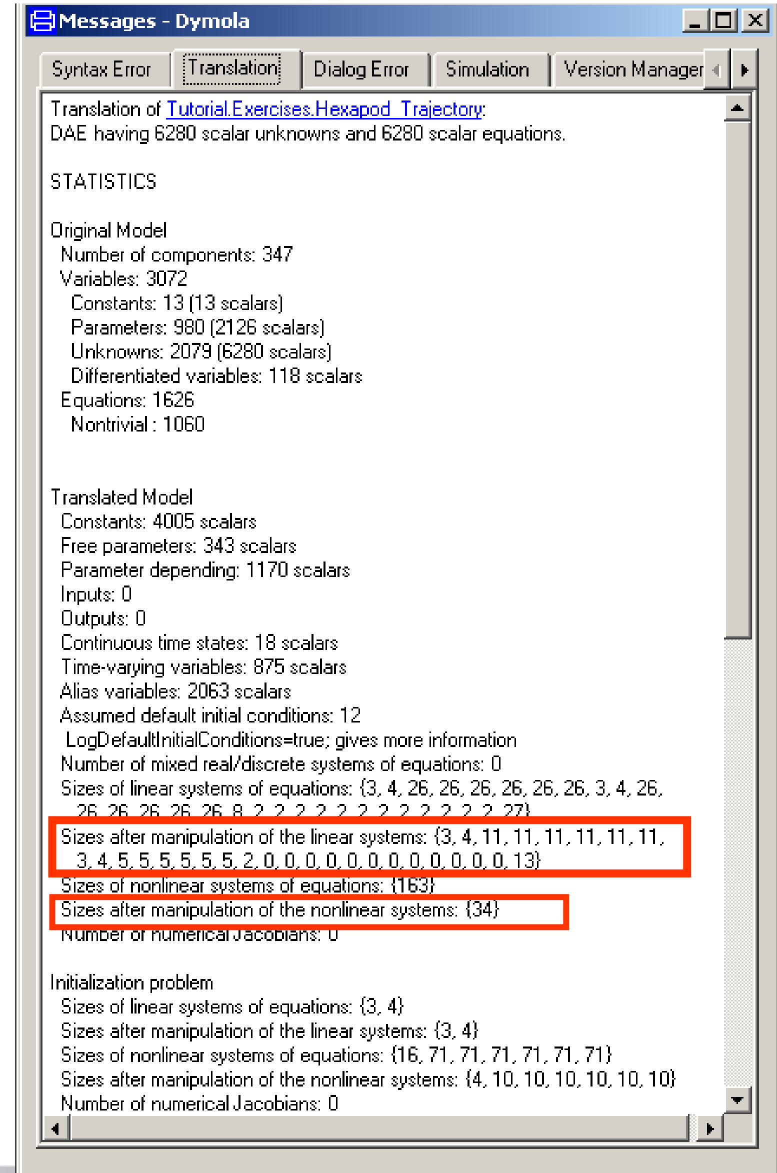
$$0 = f(\dot{x}, x, y, t, \ldots) \quad \mathrm{dim}(f) = \mathrm{dim}(x) + \mathrm{dim}(y)$$

  - Equations are rearranged to get a sequence for model evaluation (**B**lock **L**ower **T**riangle-partitioning)
  - Equations to be differentiated are determined (Pantelides algorithm)
  - superflous potential states are deselected dynamically (dummy derivative method) $\Rightarrow$ ODE:

$$\dot{x} = f(x, t, \ldots)$$

Modelica Multibody Advanced: Loops II

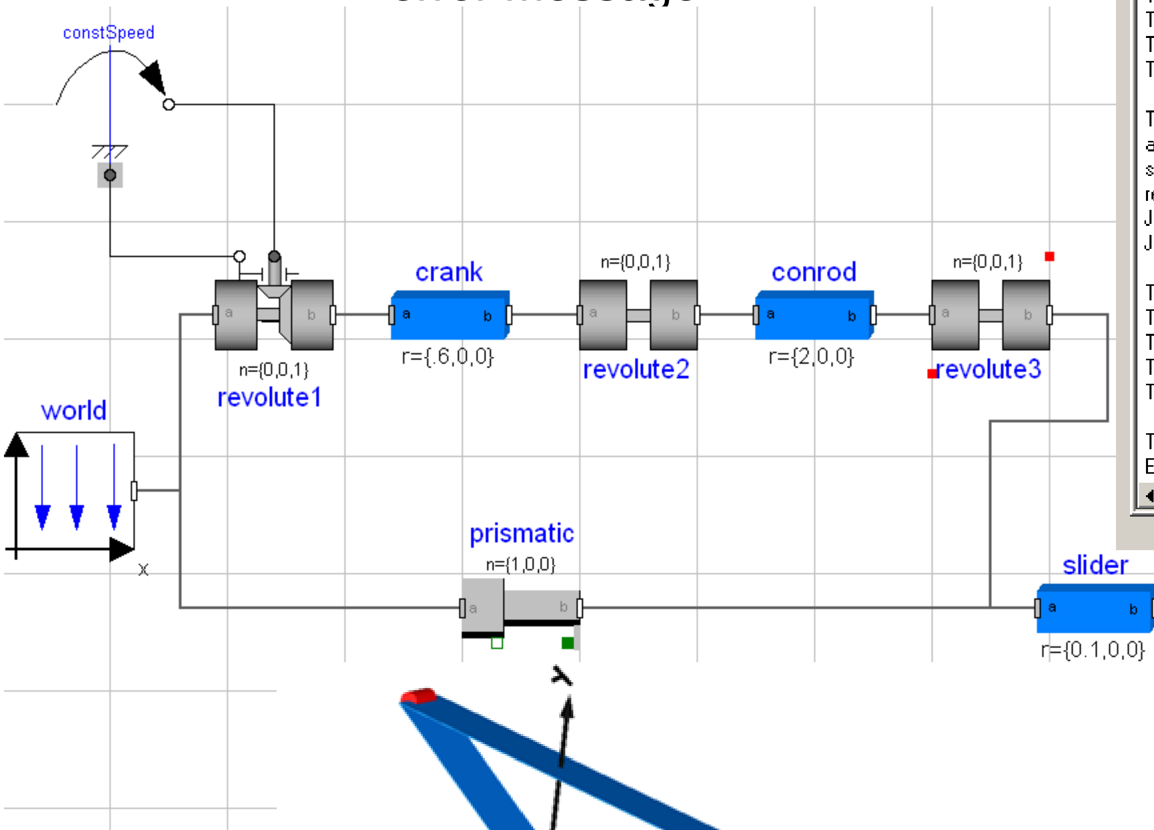
- review Translation Log in order to streamline simulation performance with model adjustments

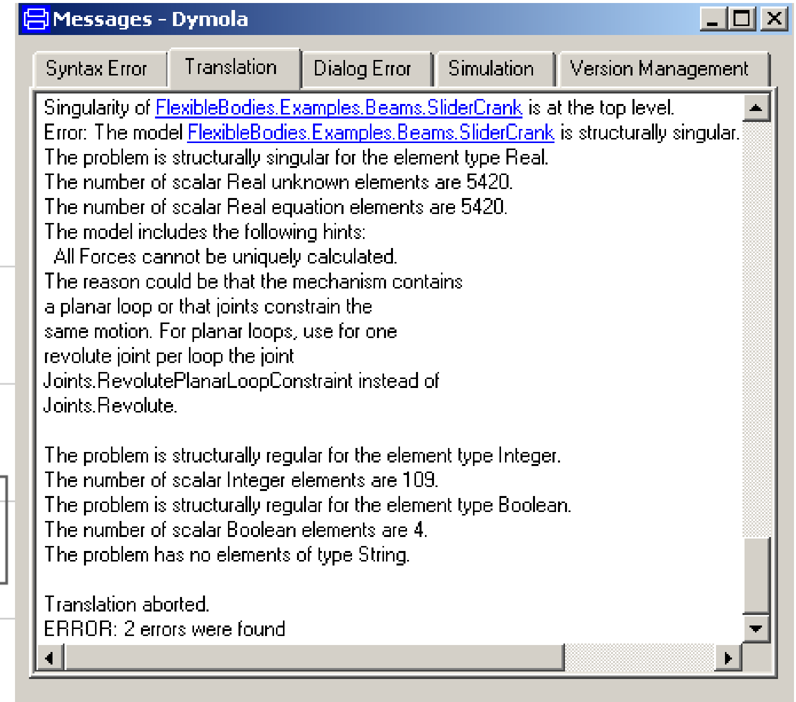


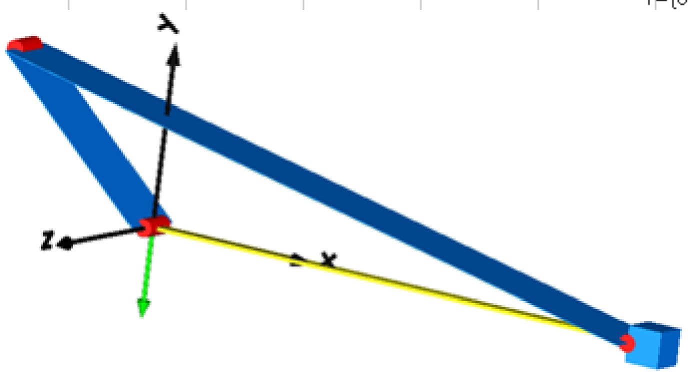
Deutsches Zentrum für Luft- und Raumfahrt e.V. in der Helmholtz-Gemeinschaft

Modelica Multibody Advanced: Loops III

- Planar loops
  - error message

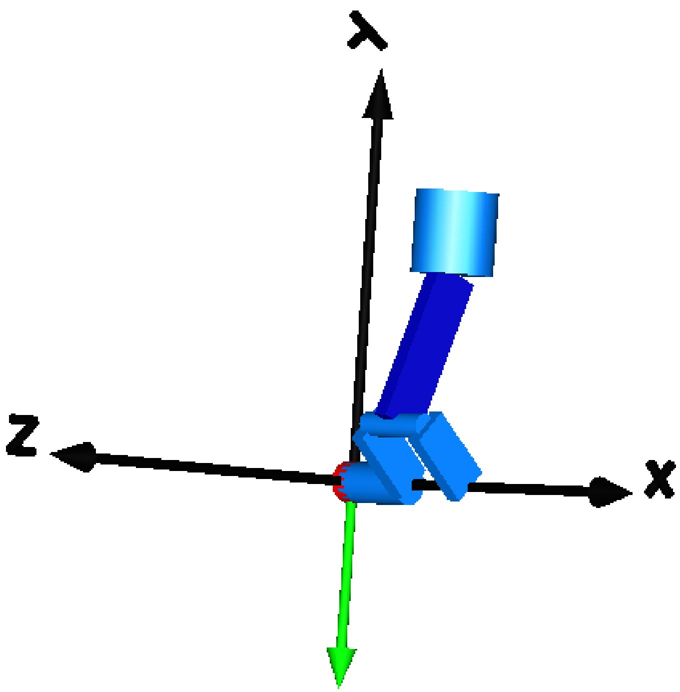


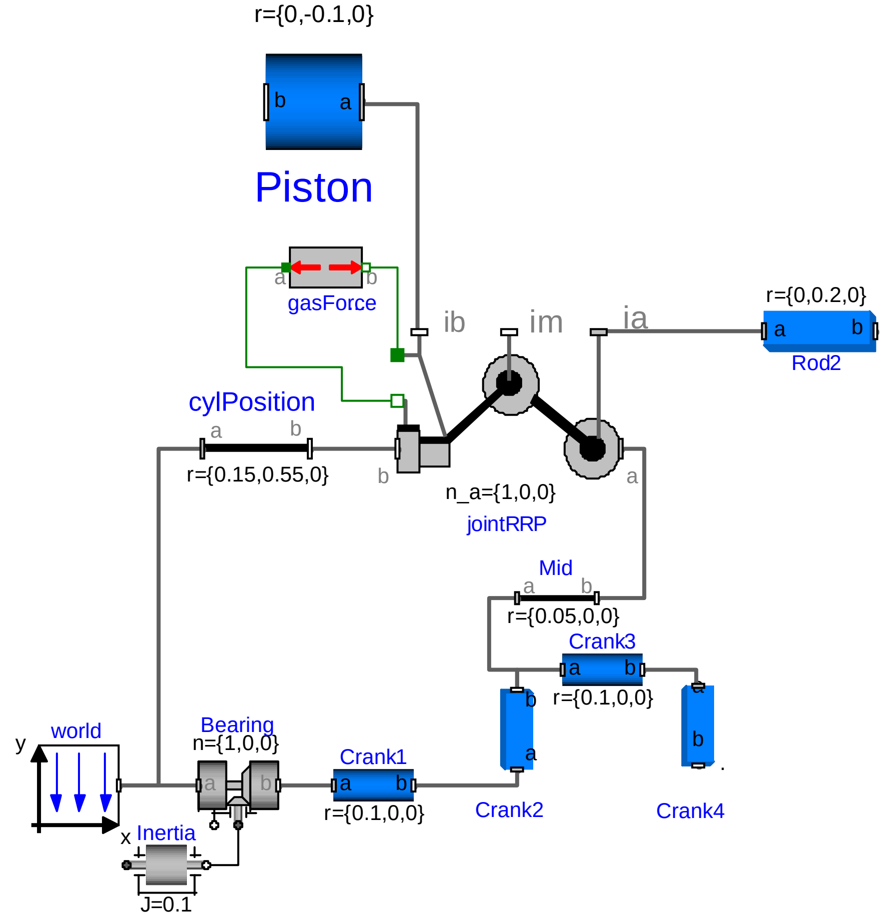




Modelica Multibody Advanced: Loops IV

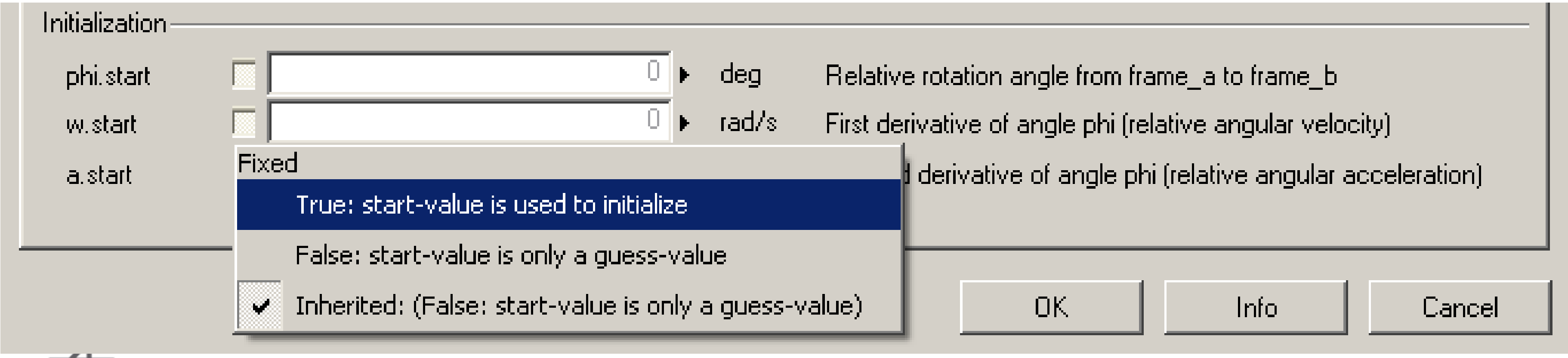
- Use of aggregrated joint objects
  - to profit from analytical loop handling according to the „characteristic pair of joints“ method by the group of Prof. Hiller



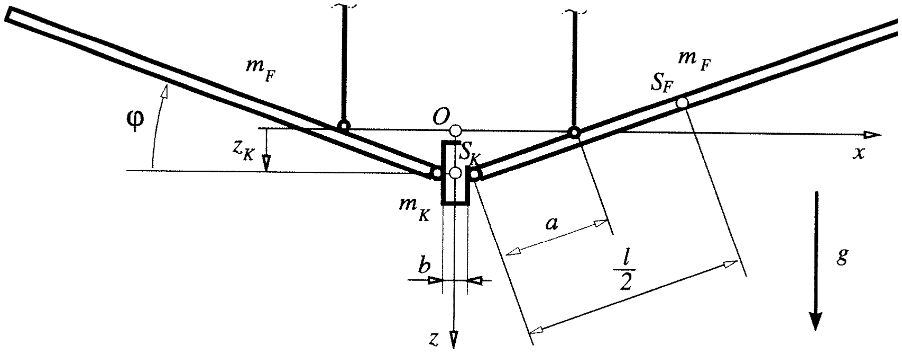


Modelica Multibody Advanced: Initialisation

- Initialisation default:
  - every state is assumed to be arbitrary unless otherwise provided
  - Newton solver starts with guess value zero in order to find consistent initial states unless otherwise provided
- If initialisation fails
  - determine, i.e. fix, characteristic variables/states in order to influence the system of equations to solve
  - provide „good“ guesses for initial states
  - be aware of singular positions, e.g. piston at bottom dead center
  - keep initialisation system consistent



Exercise 2: The Flying Gull

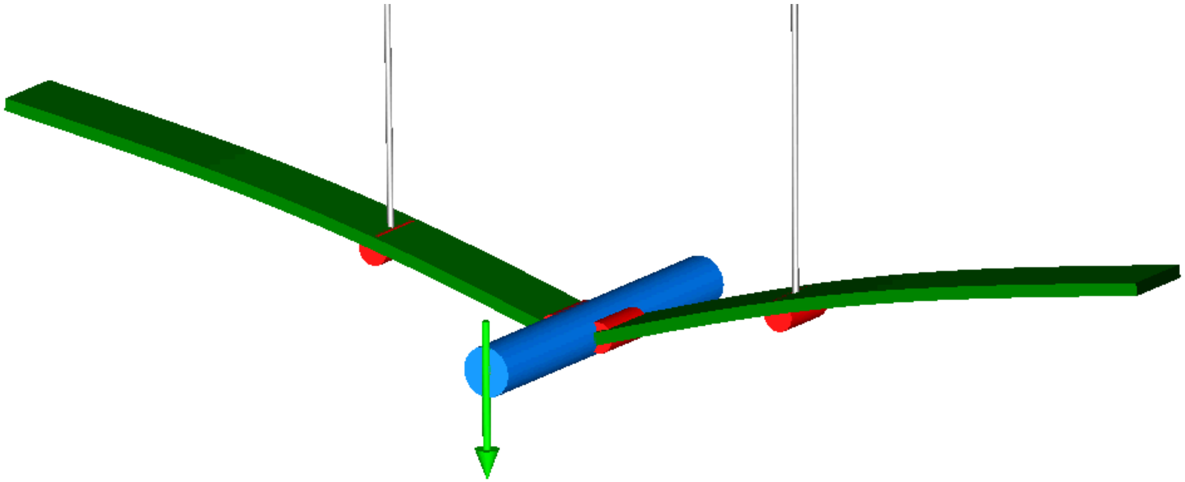


Aufgabe nach Weidemann/Pfeiffer TM 94:

Aufgabe 10

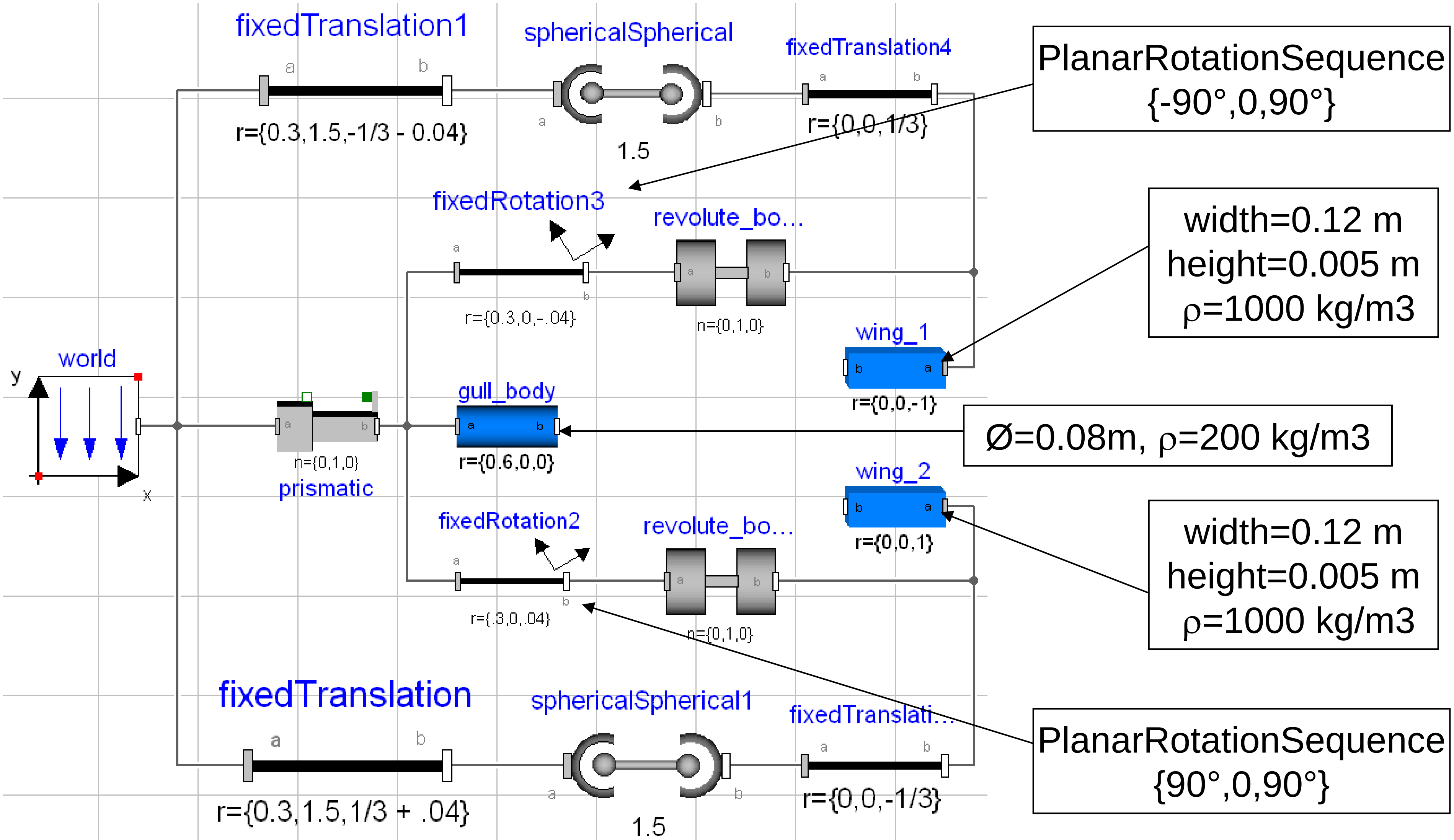
Ein beliebtes Kinderspielzeug ist die 'Fliegende Möwe'. Sie besteht aus zwei identischen Flügeln (schlanke, homogene Balken, jeweils Länge l und Masse m_F), welche um die Längsachse der Möwe drehbar am Zentralkörper (Masse m_K, Schwerpunkt S_K) aufgehängt sind. Die Breite b des Zentralkörpers sei vernachlässigbar klein. Die Möwe ist an zwei masselosen, sehr langen Fäden jeweils im Abstand a vom Zentralkörper so aufgehängt, daß die Aufhängepunkte immer auf der $x-$Achse des raumfesten $x-, y-, z-$Koordinatensystems (Ursprung O) liegen. Zur Beschreibung des Systems dient neben der Auslenkung z_K des Zentralkörpers auch der Winkel φ der Flügel gegenüber einer Waagerechten.

a) Wie lautet die kinematische Abhängigkeit zwischen φ und z_K ?
b) Wie groß ist die kinetische Energie des Gesamtsystems ?
c) Wie groß ist die potentielle Energie des Gesamtsystems ?
d) Wie lautet die Bewegungsgleichung für die Koordinate φ ?



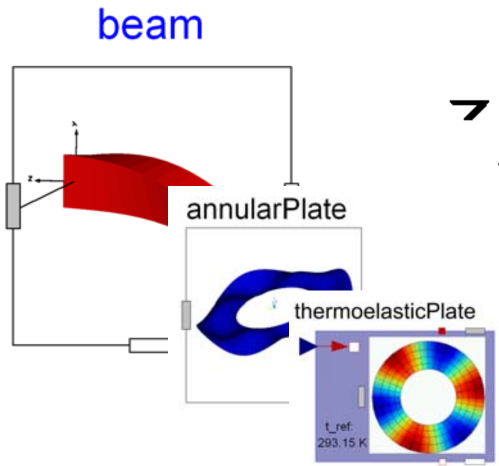
- Look for the equilibrium position !

Exercise 2: The Flying Gull I



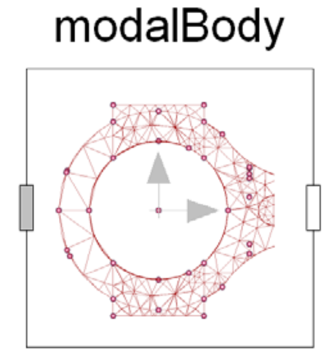
Contents

The name of the game : 2 types of modelling elements



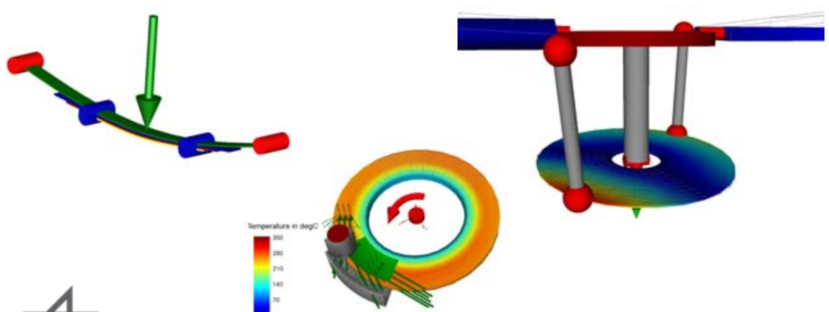
What do they have in common ?

- Floating frame of reference approach
  - Structure of equations of motion
    - Data structure, so called SID
      (Standard-Input-Data: Wallrapp '94)

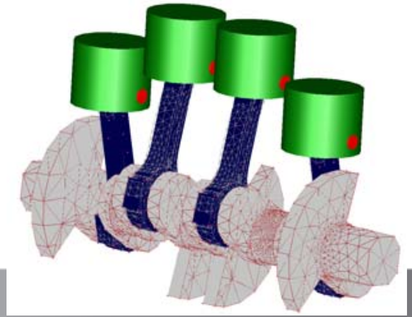


In what do they differ ?

- Semi-analytical description implemented in Modelica
- Modelica generates SID
- Animation uses analytical description



- FEM-based body description (Abaqus-SID-interface, SIMPACK-FEMBS)
- Modelica reads externally generated SID file
- Modelica reads externally generated animation data (wavefront) file



Theory: the equations of motion

- principle of virtual power

$$\int \delta \boldsymbol{v} \; (\boldsymbol{f} - \boldsymbol{a}) \; \mathrm{d}m = 0$$

- equations of motion: here

$$\boldsymbol{\omega} := \boldsymbol{\omega}_R \,, \quad \tilde{\boldsymbol{\omega}} := \boldsymbol{\omega} \times$$

$$\begin{pmatrix} m\boldsymbol{I}_3 & & \text{sym.} \\ m\tilde{\boldsymbol{d}} & \boldsymbol{J} & \\ \boldsymbol{C}_t & \boldsymbol{C}_r & \boldsymbol{M}_e \end{pmatrix} \begin{pmatrix} \boldsymbol{a}_R \\ \boldsymbol{\alpha}_R \\ \ddot{\boldsymbol{q}} \end{pmatrix} + \begin{pmatrix} 2\tilde{\boldsymbol{\omega}}\boldsymbol{C}_t^T\dot{\boldsymbol{q}} + \tilde{\boldsymbol{\omega}}\tilde{\boldsymbol{\omega}}\boldsymbol{d} \\ \boldsymbol{G}_r\dot{\boldsymbol{q}}\boldsymbol{\omega} + \tilde{\boldsymbol{\omega}}\boldsymbol{J}\boldsymbol{\omega} \\ \boldsymbol{G}_e\dot{\boldsymbol{q}}\boldsymbol{\omega} + \boldsymbol{O}_e\boldsymbol{\Omega} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \boldsymbol{K}_e\boldsymbol{q} + \boldsymbol{D}_e\dot{\boldsymbol{q}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_a \\ \boldsymbol{f}_\alpha \\ \boldsymbol{f}_q \end{pmatrix}$$

the generalized Newton-Euler-equations of motion of an unconstrained deformable body

- SID structure: definition of file format to file volume integrals

$$\boldsymbol{C}_r, \boldsymbol{C}_t, \boldsymbol{J}, \boldsymbol{M}_e, \boldsymbol{K}_e, \boldsymbol{D}_e, \boldsymbol{G}_r \ldots$$

Theory: 2nd order beam theory

- Bending in xy- und xz-plane, torsion and lengthening

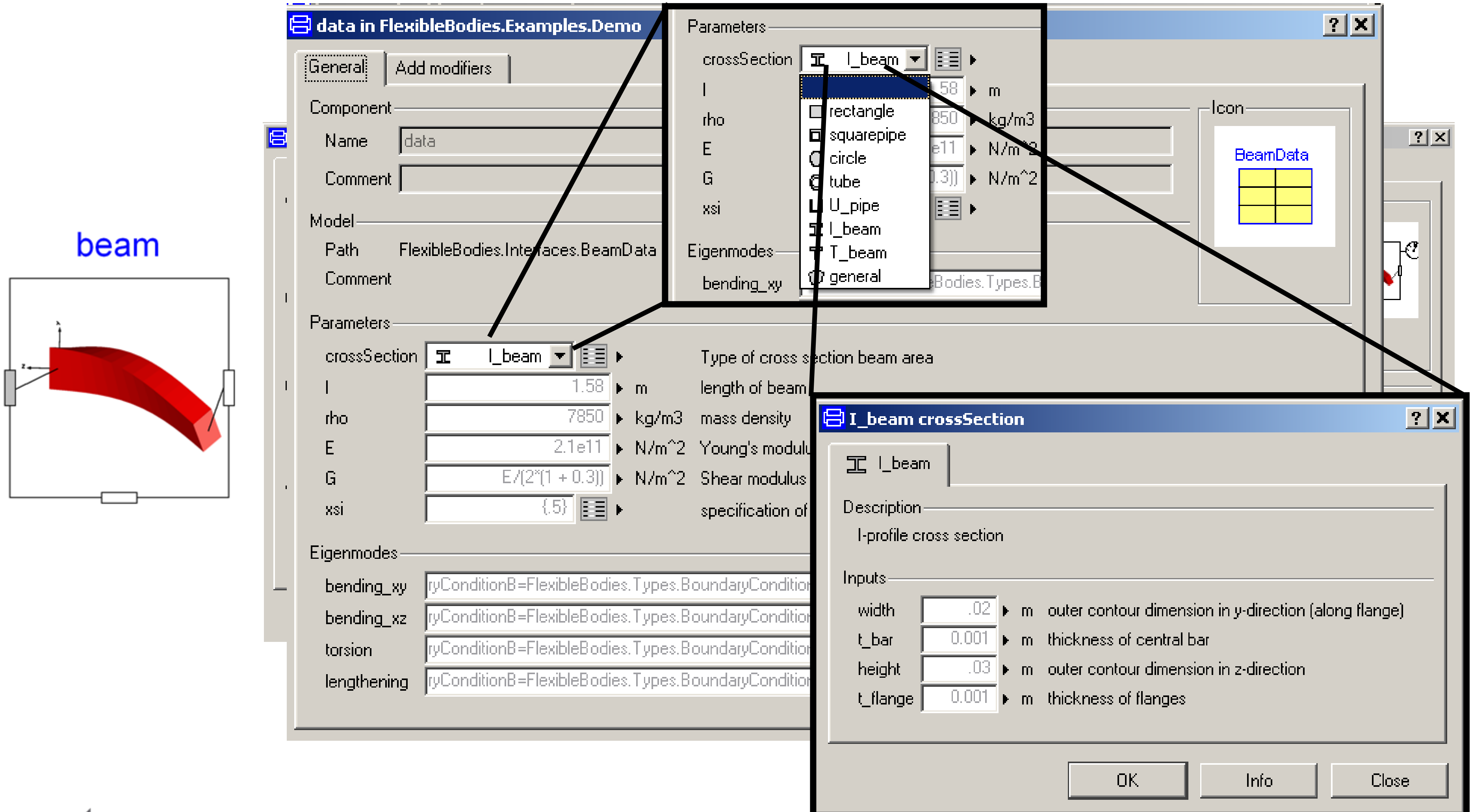
$$\boldsymbol{u}(x,t) = \begin{pmatrix} u \\ v \\ w \end{pmatrix} + \begin{pmatrix} -\frac{1}{2}\int_0^x v'^2 + w'^2 \,\mathrm{d}x \\ -\int_0^x\int_0^{\bar{x}} \theta\, w'' \,\mathrm{d}\bar{\bar{x}}\,\mathrm{d}\bar{x} + \int_0^x u'v' \,\mathrm{d}\bar{x} \\ \int_0^x\int_0^{\bar{x}} \theta\, v'' \,\mathrm{d}\bar{\bar{x}}\,\mathrm{d}\bar{x} + \int_0^x u'w' \,\mathrm{d}\bar{x} \end{pmatrix}$$

- e.g. for bending in xy-plane: $v(x,t) = \boldsymbol{\Phi}_v(x)\boldsymbol{q}_v(t)$
- analytical solutions ot the eigenvalue problen f the Eule ernoulli-beam

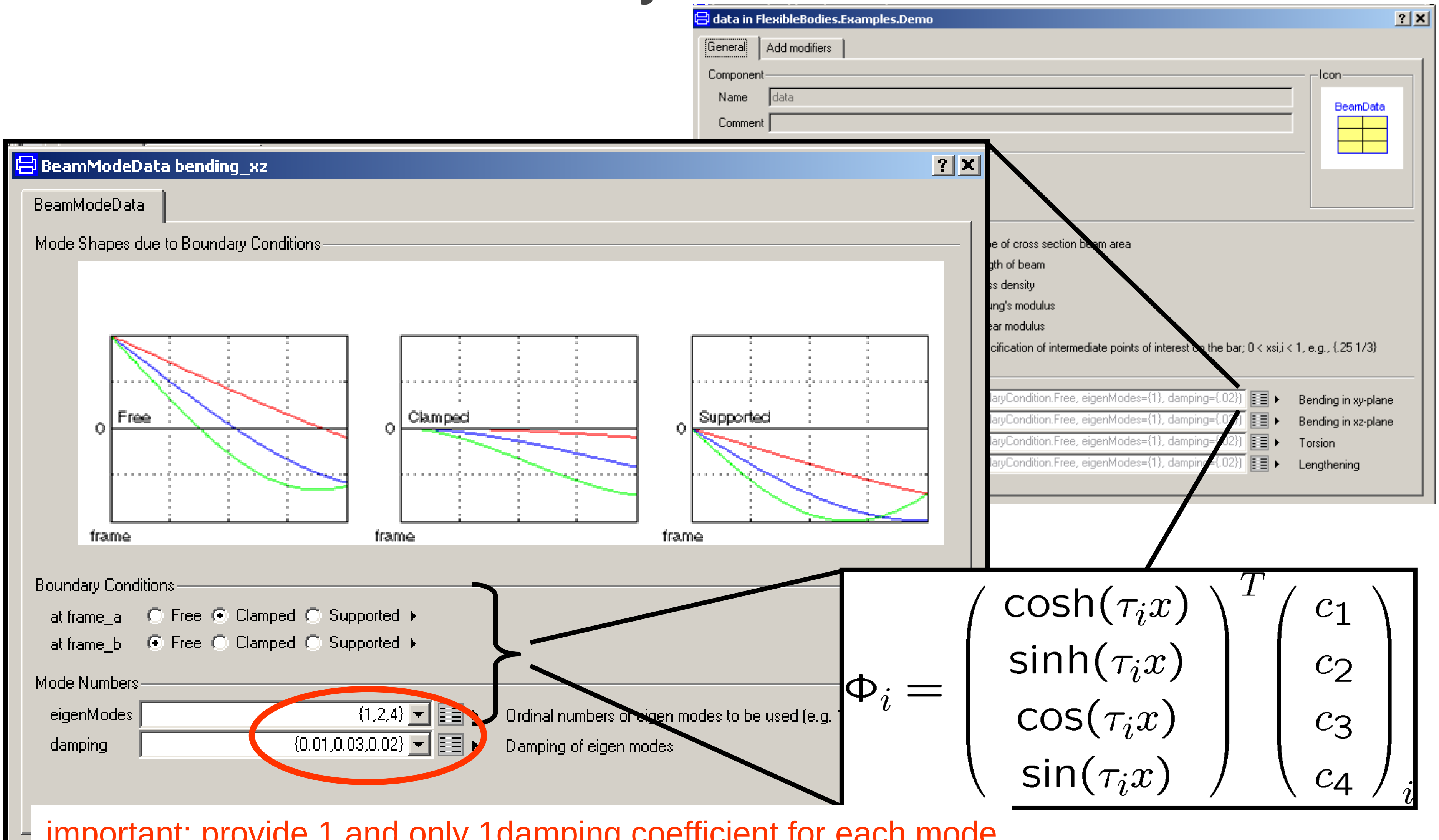
$$\Phi_i = \begin{pmatrix} \cosh(\tau_i x) \\ \sinh(\tau_i x) \\ \cos(\tau_i x) \\ \sin(\tau_i x) \end{pmatrix}^T \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}_i$$

$$\boldsymbol{u}(\boldsymbol{c},t) = \boldsymbol{\Phi}(\boldsymbol{c})\,\boldsymbol{q}(t) + \frac{1}{2}\begin{pmatrix} \boldsymbol{q}^T\boldsymbol{\Phi}_x \\ \boldsymbol{q}^T\boldsymbol{\Phi}_y \\ \boldsymbol{q}^T\boldsymbol{\Phi}_z \end{pmatrix}\boldsymbol{q}$$

FlexibleBodies Library: Beam Menu I



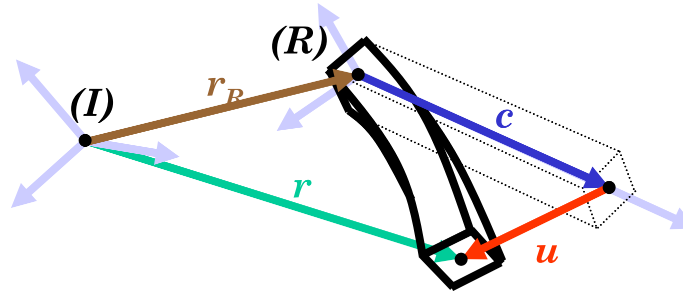
FlexibleBodies Library: Beam Menu II



$$\Phi_i = \begin{pmatrix} \cosh(\tau_i x) \\ \sinh(\tau_i x) \\ \cos(\tau_i x) \\ \sin(\tau_i x) \end{pmatrix}^T \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}_i$$

important: provide 1 and only 1damping coefficient for each mode

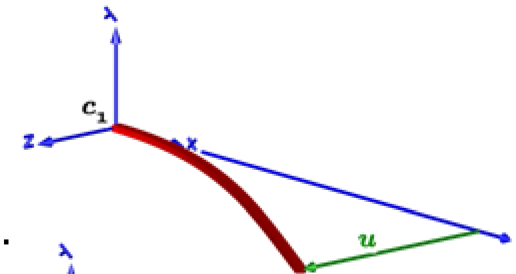
Boundary Conditions I



$$\boldsymbol{r}(\boldsymbol{c},t) = \boldsymbol{r}_R(t) + \boldsymbol{c} + \boldsymbol{u}(\boldsymbol{c},t)$$

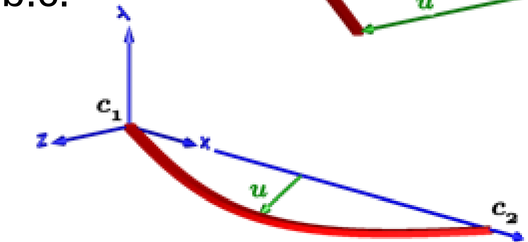
- 1st Option: tangent frame: clamped-free b.c. correponds to cantilever beam

$$\boldsymbol{u}(\boldsymbol{c}=0,t)=0 \qquad \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{c}}(\boldsymbol{c}=0,t)=0$$



- 2nd Option: chord frame: supported-supported b.c.

$$\boldsymbol{u}(\boldsymbol{c}_1)=0 \qquad \boldsymbol{u}(\boldsymbol{c}_2)\cdot\overline{\boldsymbol{c}_1\boldsymbol{c}_2}=0$$



- 3rd Option: Buckens frame: free-free b.c. $^0\boldsymbol{C}_r = {}^0\boldsymbol{C}_t = {}^1\boldsymbol{d}_C = \boldsymbol{O}$

$$\begin{pmatrix} m\boldsymbol{I}_3 & & \text{sym.} \\ m\tilde{\boldsymbol{d}}_C & \boldsymbol{J} & \\ \boldsymbol{C}_t & \boldsymbol{C}_r & \boldsymbol{M}_e \end{pmatrix}\begin{pmatrix} \boldsymbol{a}_R \\ \boldsymbol{\alpha}_R \\ \ddot{q} \end{pmatrix} = \boldsymbol{h}_\omega - \begin{pmatrix} 0 \\ 0 \\ \boldsymbol{K}_e\,\boldsymbol{q} + \boldsymbol{D}_e\,\dot{\boldsymbol{q}} \end{pmatrix} + \begin{pmatrix} \boldsymbol{f}_a \\ \boldsymbol{f}_\alpha \\ \boldsymbol{f}_e \end{pmatrix}$$

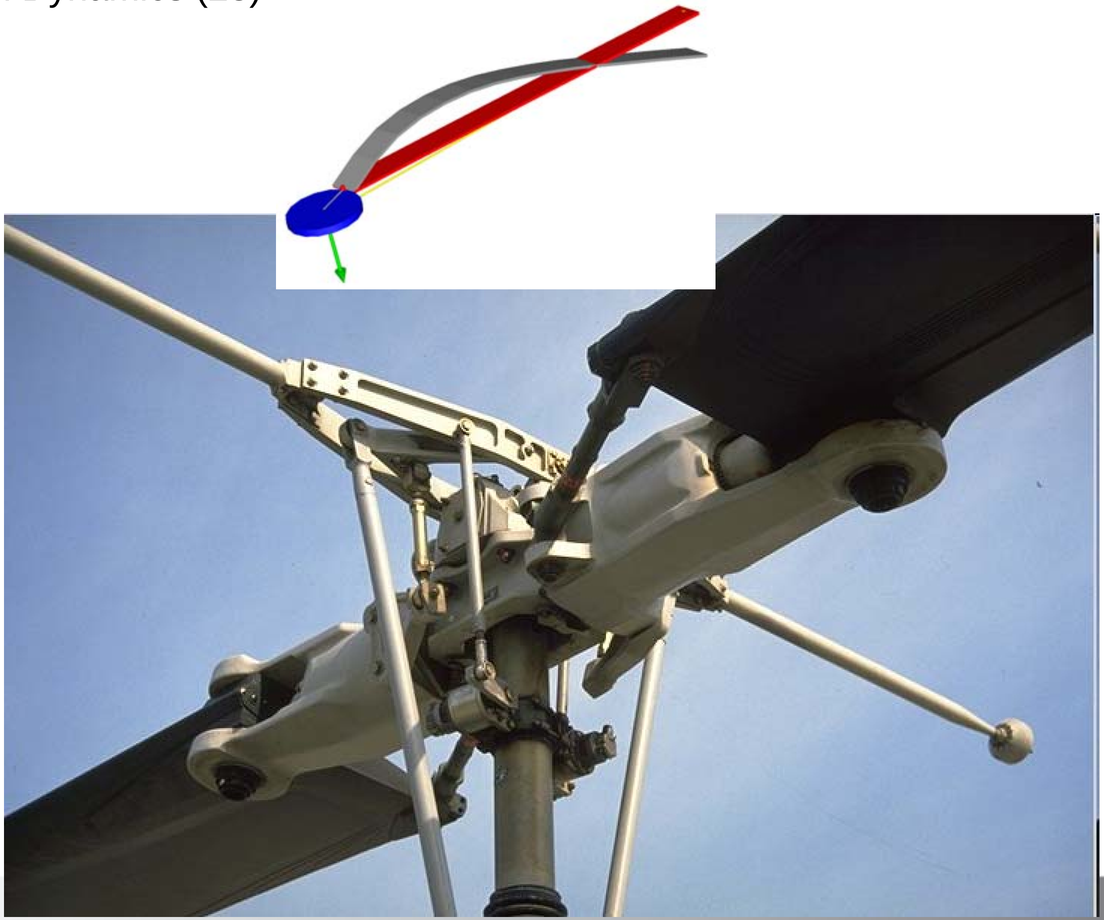
- Linearisation: choose reference frame in such a way that is as small as possible

$\boldsymbol{u} \ll 1 \quad \Rightarrow$ prefer Buckenssystem

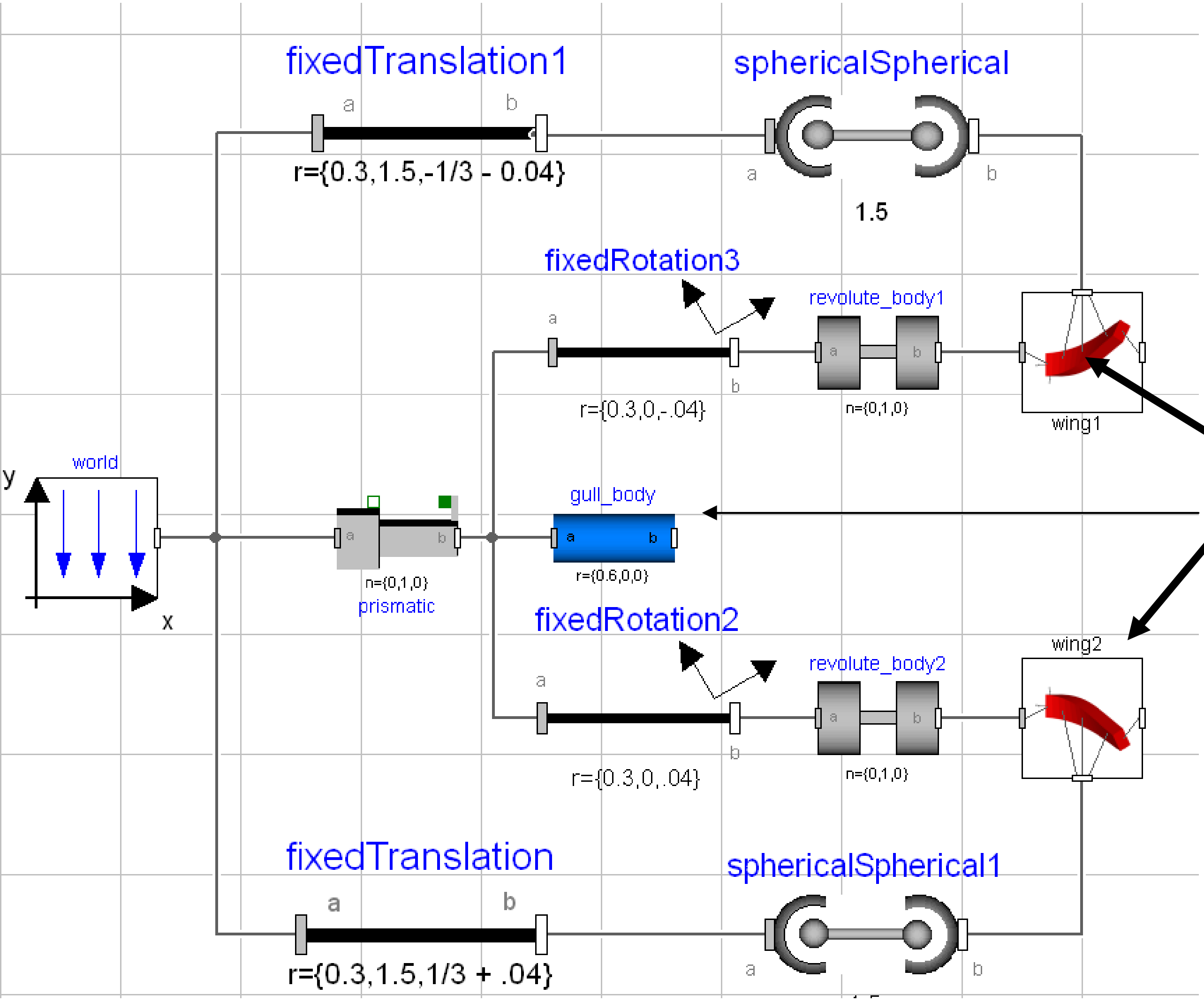
see Schwertassek/Wallrapp/Shabana99

Boundary Conditions II

- Helikopter-Rotor (see Examples/Beam)
  - choose the boundary conditions according to the attachment joint
  - Heckmann2010: On the Choice of Boundary Conditions for Mode Shapes, Mulibody System Dynamics (23)



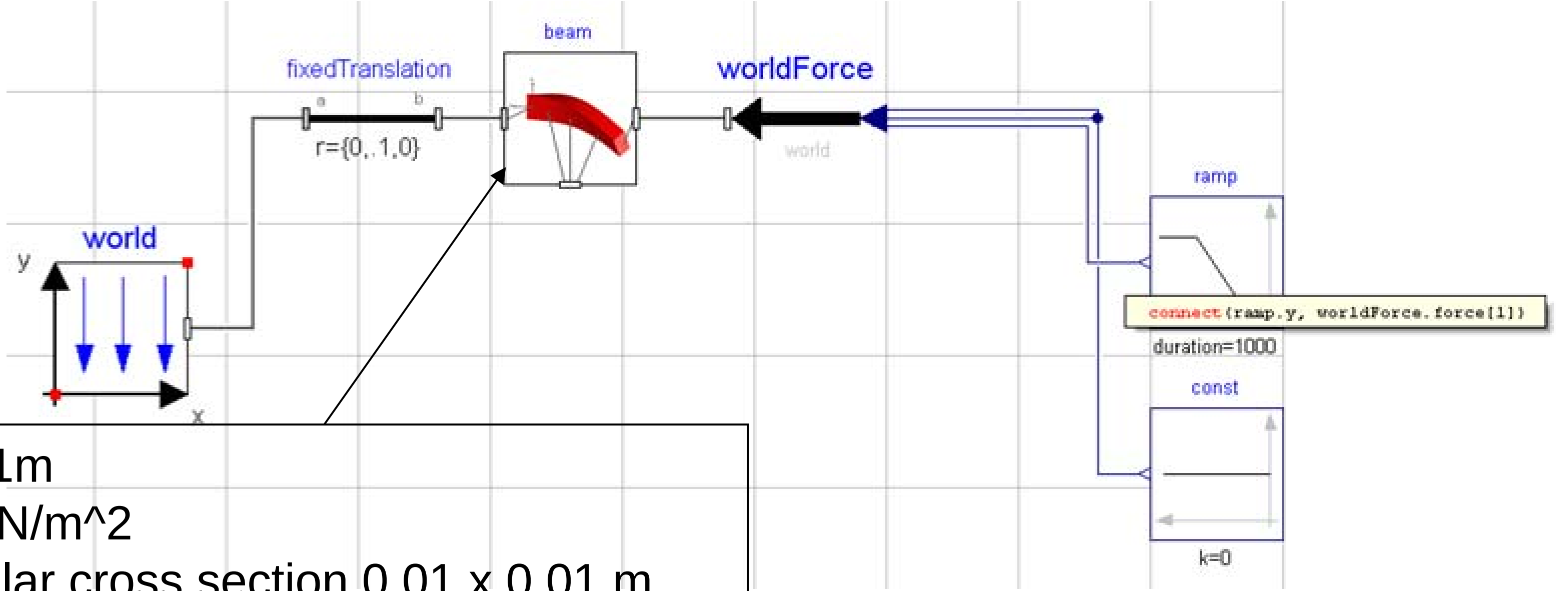
Exercise 3: The Flying Gull II



rectangle crosssection
width=0.12 height=0.005
l= 1m
ρ=1000 kg/m3
E=1e9 N/m2
xsi={1/3}
bending_xz
supported/free, {1}, {0.02}
all other BeamModeData
fill(0,0), fill(0,0)

Exercise 4: a classic Pitfall I

- Model the following system
  - (quasi-) static deformation:a thrust-force shortens the beam

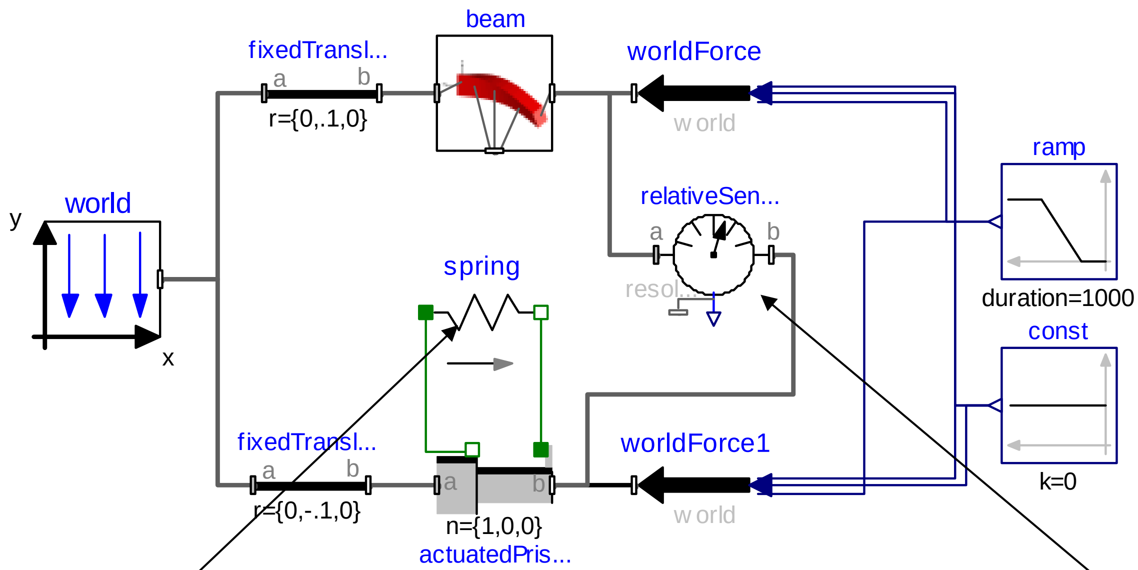


length= 1m
E=1e10 N/m^2
rectangular cross section 0.01 x 0.01 m
1st eigenmode for lengthening deformation
all other deformations are disregarded
switch off exaggerated animation

Simulate 1000s with Radau5 !
Plot beam.frame_b.r_0[1] !

Exercise 4: a classic Pitfall II

- the system is now extended by an equivalent spring !



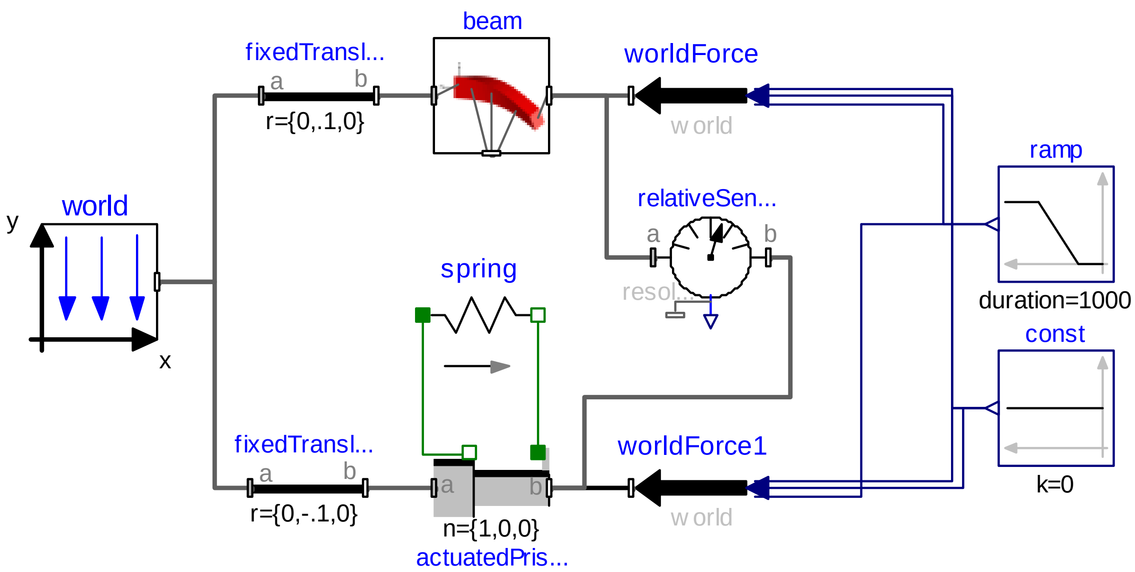
c=0.01m*0.01m*1e10 N/m^2 / (1 m)
unstretched length = 1m

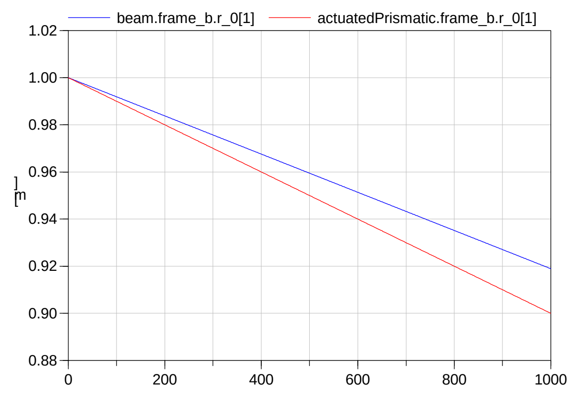
compare the deformations by measurements !

Plot the relativeSensor.r_rel[1] !
Gradually increase the number of modes !

Exercise 4: a classic Pitfall III

- static deflection: thrust force shortens beam and equivalent spring

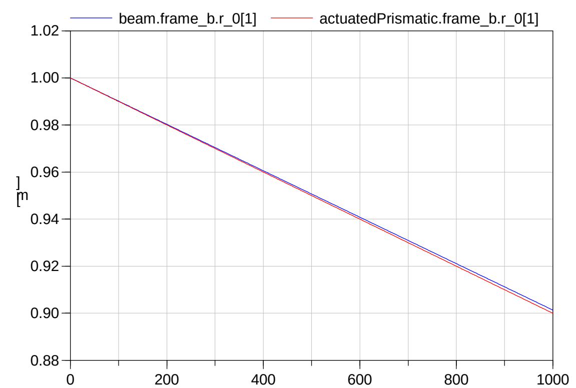




1 eigenmode

	spring	beam	error
1 eigenmode	-10 cm	-8.1 cm	19 %
5 eigenmodes	-10 cm	-9.6 cm	4 %
10 eigenmodes	-10 cm	-9.8 cm	2 %
15 eigenmodes	-10 cm	-9.9 cm	1 %

comparison: deflections at the end



15 eigenmodes

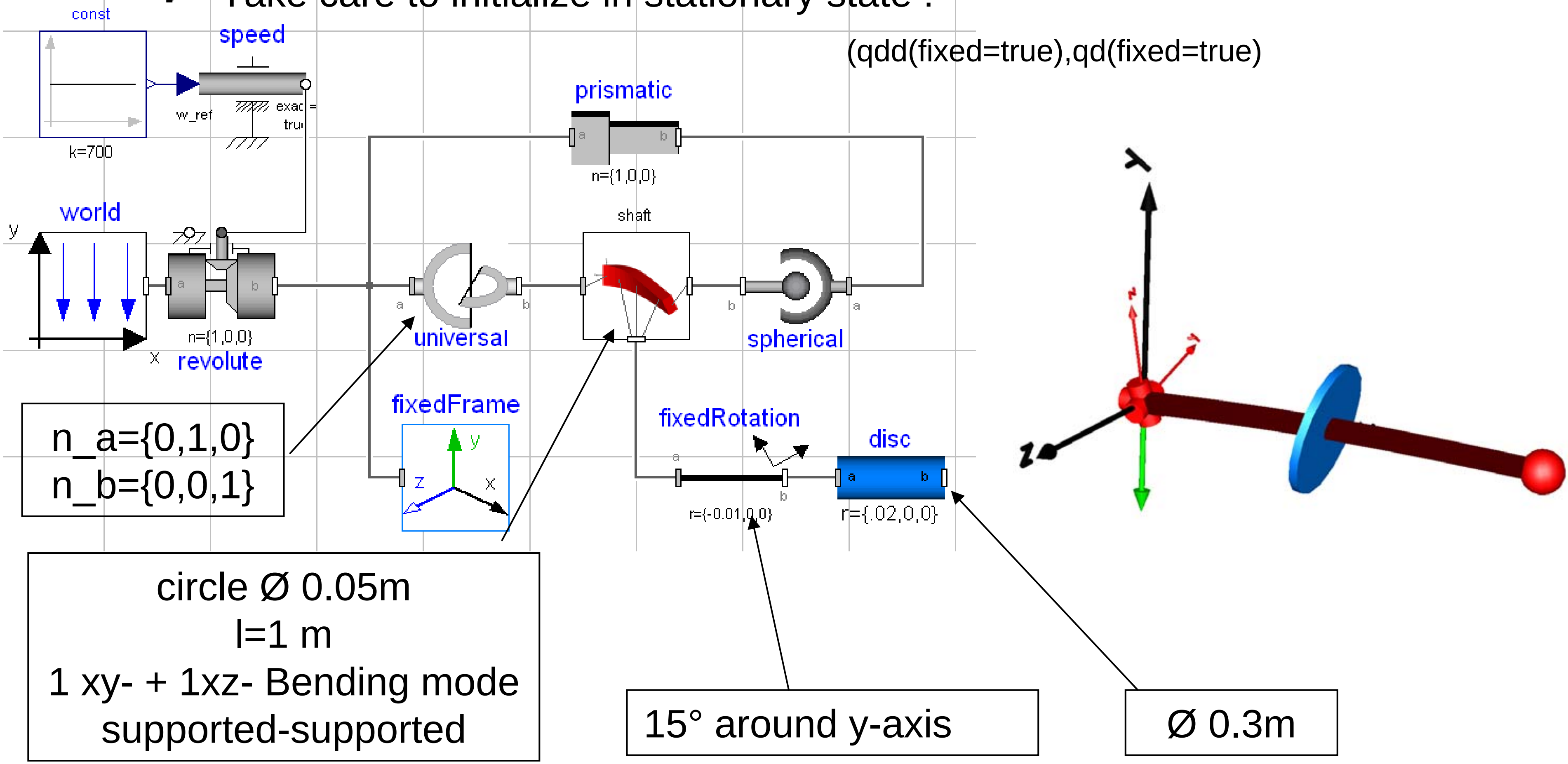
Exercise 4: a classic Pitfall IV

- Mechanical background
  - static deflections rely on elastic properties only
  - eigenmodes consider elastic and interia properties
    - that‘s why they are well suited for dynamic problems
- Geometrical background
  - analytically: $u = c \cdot x$
  - expansion with eigenmodes: $u = \sin(\frac{2x}{\pi l}) + \sin(\frac{2x}{3\pi l}) + \dots$
- It is proven that Raleigh-Ritz approach converges against true value
  - but how fast ?
  - this is an extreme example, e.g. bending is less sensitive
- Check whether a higher number of modes changes results !

Erxercise 5: unbalanced Shaft

- Instability at which rotational velocity ?
- Take care to initialize in stationary state !

(qdd(fixed=true),qd(fixed=true)



Contents

Recall Theory: the equations of motion

- principle of virtual power

$$\int \delta \boldsymbol{v} \, (\boldsymbol{f} - \boldsymbol{a}) \, \mathrm{d}m = 0$$

- equations of motion: here

$$\boldsymbol{\omega} := \boldsymbol{\omega}_R \,, \quad \tilde{\boldsymbol{\omega}} := \boldsymbol{\omega} \times$$

$$\begin{pmatrix} m\boldsymbol{I}_3 & & \text{sym.} \\ m\tilde{\boldsymbol{d}} & \boldsymbol{J} & \\ \boldsymbol{C}_t & \boldsymbol{C}_r & \boldsymbol{M}_e \end{pmatrix} \begin{pmatrix} \boldsymbol{a}_R \\ \boldsymbol{\alpha}_R \\ \ddot{\boldsymbol{q}} \end{pmatrix} + \begin{pmatrix} 2\tilde{\boldsymbol{\omega}}\boldsymbol{C}_t^T\dot{\boldsymbol{q}} + \tilde{\boldsymbol{\omega}}\tilde{\boldsymbol{\omega}}\boldsymbol{d} \\ \boldsymbol{G}_r\dot{\boldsymbol{q}}\boldsymbol{\omega} + \tilde{\boldsymbol{\omega}}\boldsymbol{J}\boldsymbol{\omega} \\ \boldsymbol{G}_e\dot{\boldsymbol{q}}\boldsymbol{\omega} + \boldsymbol{O}_e\boldsymbol{\Omega} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \boldsymbol{K}_e\boldsymbol{q} + \boldsymbol{D}_e\dot{\boldsymbol{q}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_a \\ \boldsymbol{f}_\alpha \\ \boldsymbol{f}_q \end{pmatrix}$$

the generalized Newton-Euler-equations of motion of an unconstrained deformable body

- SID structure: definition of file format to file volume integrals

$$\boldsymbol{C}_r, \boldsymbol{C}_t, \boldsymbol{J}, \boldsymbol{M}_e, \boldsymbol{K}_e, \boldsymbol{D}_e, \boldsymbol{G}_r \ldots$$

SID-Data from FE: Where do they come from ?

- Consider the linear FE-equation

$$\boldsymbol{M}\ddot{\boldsymbol{u}}_{fe} + \boldsymbol{K}\boldsymbol{u}_{fe} = \boldsymbol{f}_{fe}$$

- the related eigenvalue problem

$$[\boldsymbol{M}\omega_i^2 + \boldsymbol{K}]\boldsymbol{v}_i = 0$$

- a set of eigenvectors $\boldsymbol{v}_1, \boldsymbol{v}_2, ...$
- a selection of nodes $\boldsymbol{c}_1, \boldsymbol{c}_2, ...$
- for each node mode shapes are collected from set of eigenvectors

$$\boldsymbol{\Phi}(\boldsymbol{c}_1), \boldsymbol{\Phi}(\boldsymbol{c}_2), ...$$

- the related rotational terms (non-volume-elements only)

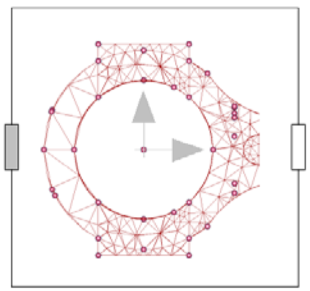
$$\boldsymbol{\Psi}(\boldsymbol{c}_1), \boldsymbol{\Psi}(\boldsymbol{c}_2), ...$$

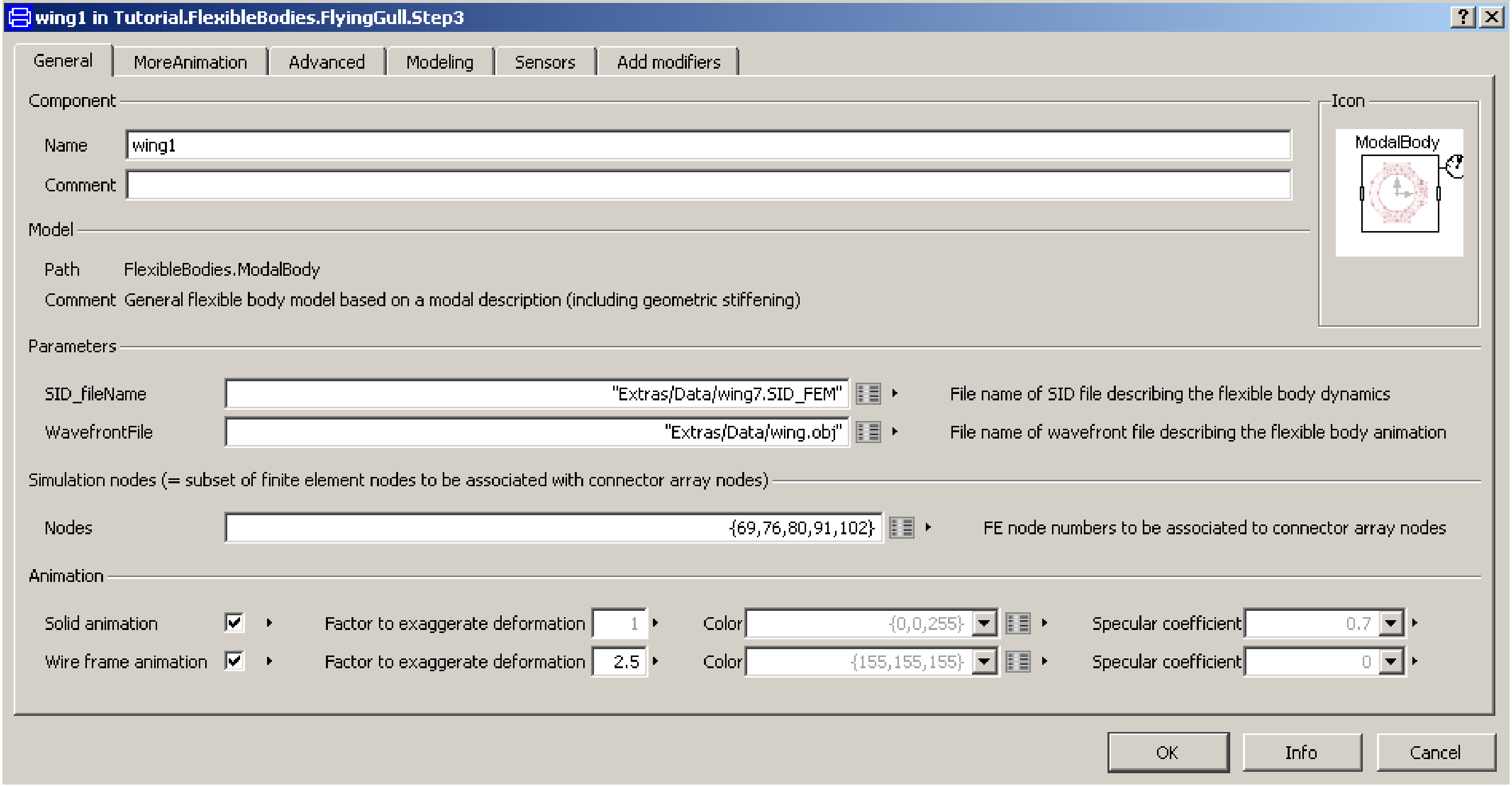
- the volume integrals are reassembled from (substructure) element inertia and stiffness data

$$\boldsymbol{C}_r, \boldsymbol{C}_t, \boldsymbol{J}, \boldsymbol{M}_e, \boldsymbol{K}_e, \boldsymbol{D}_e, \boldsymbol{G}_r...$$

FlexibleBodies Library: ModalBody Menu I

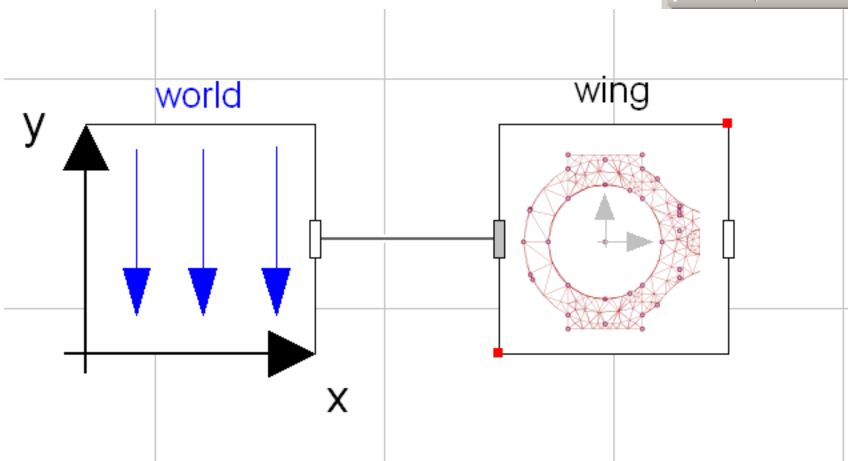
modalBody

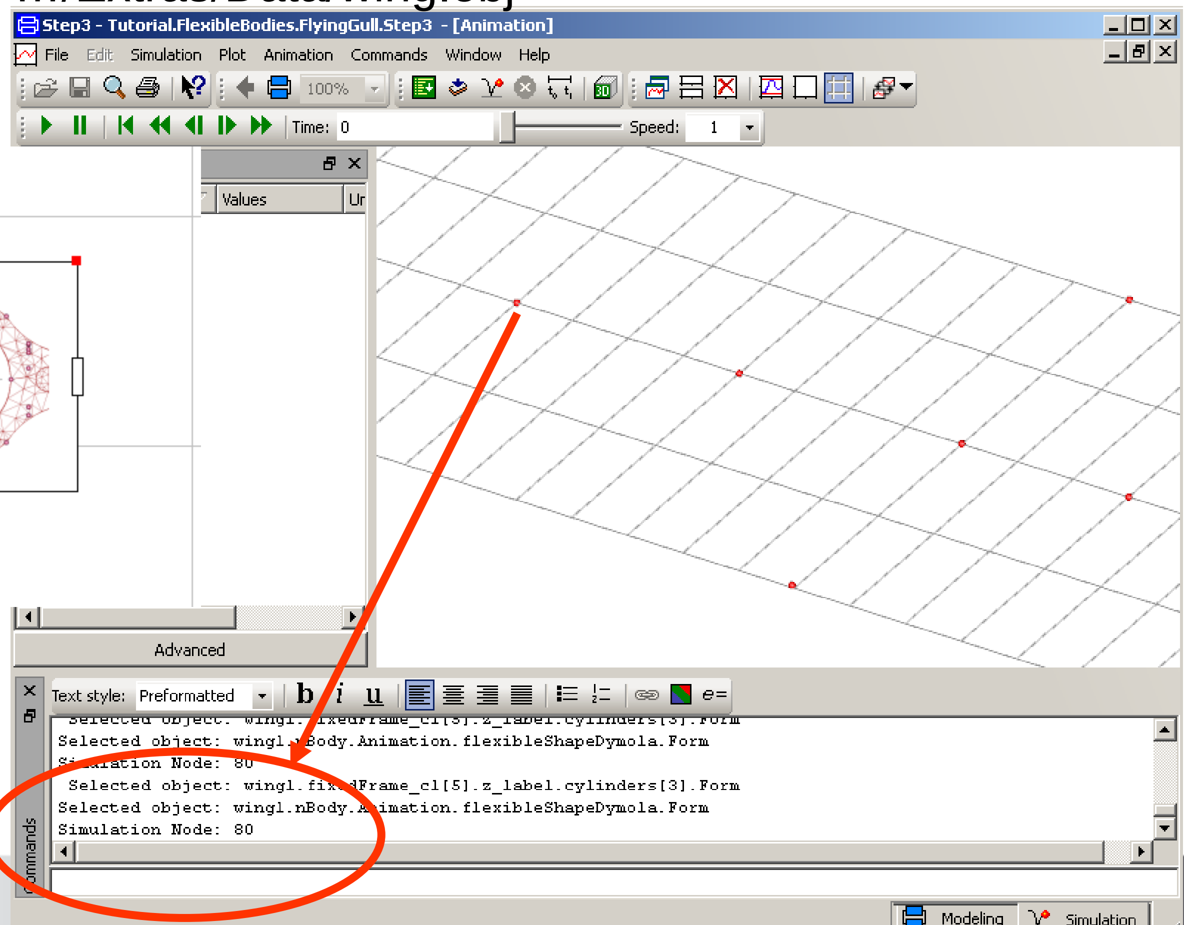




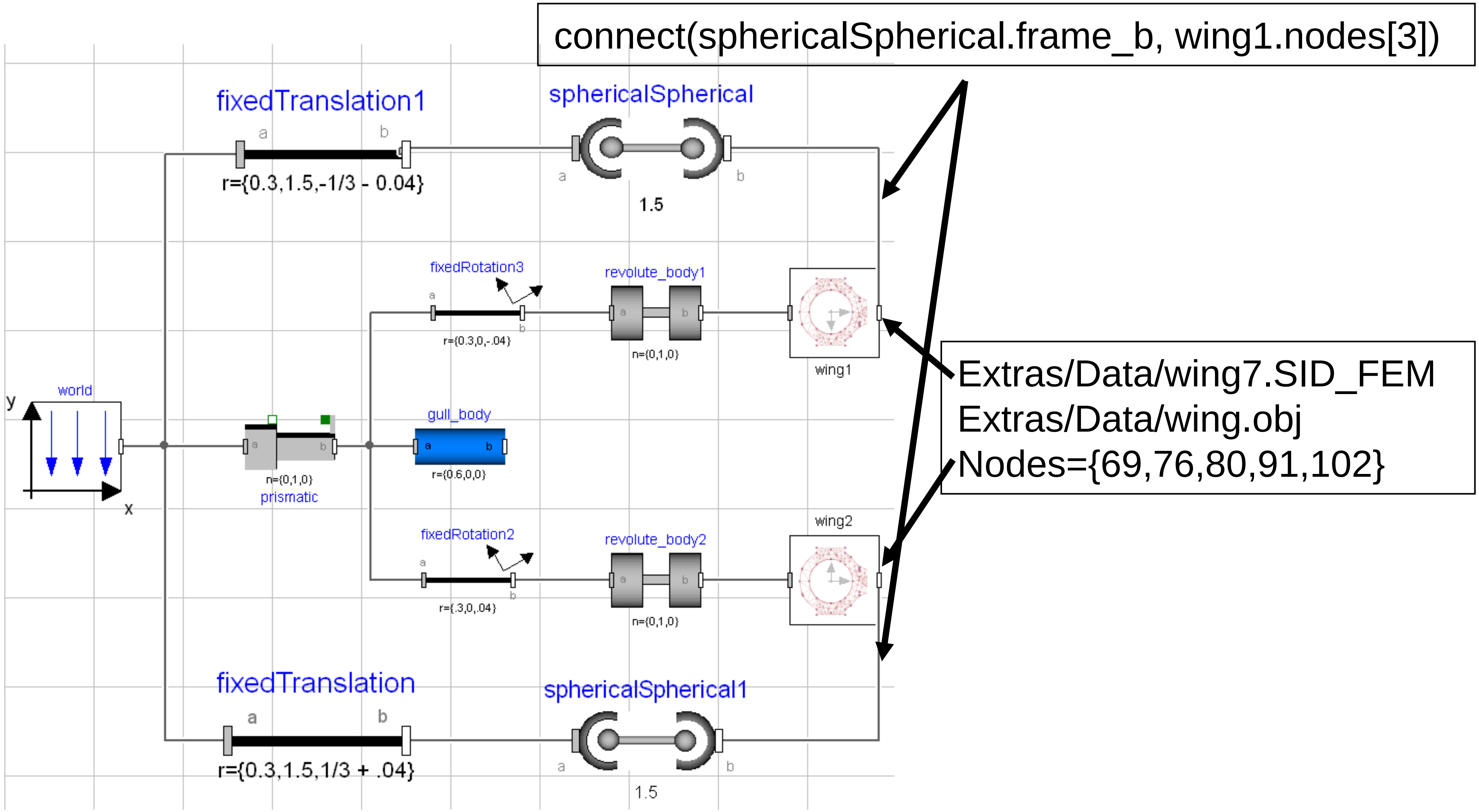
Exercise 6: The Flying Gull III

- 1st step:
  - introduce world and ModalBody- model
  - assign SID-file …/Extras/Data/wing7.SID_FEM
  - assign OBJ-file …/Extras/Data/wing.obj

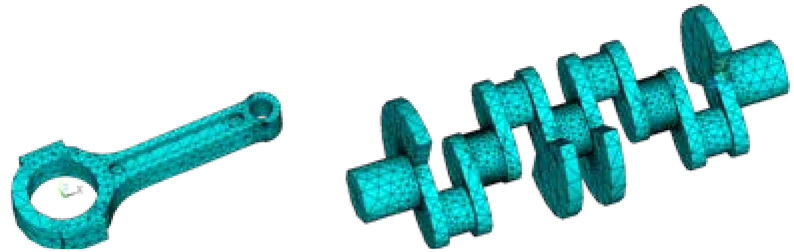




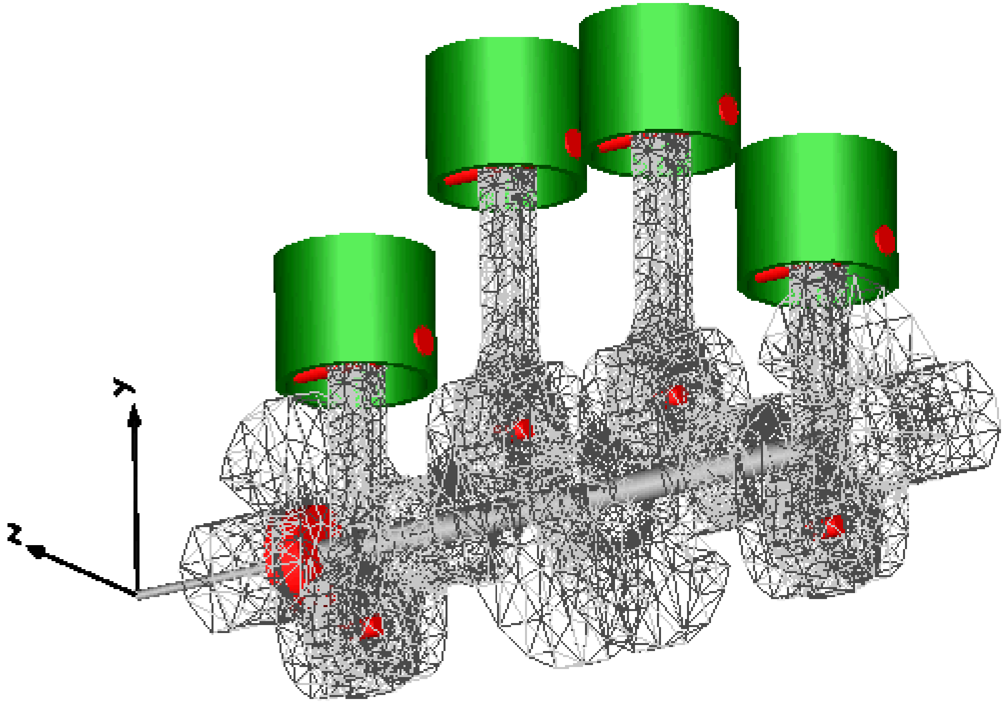
Exercise 6: The Flying Gull III



ModalBody example: 4-Cylinder-Engine

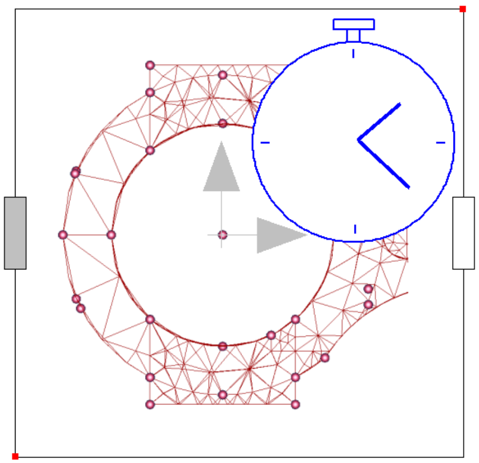


- FEM-models
  - Crankshaft : 106.789 nodes
  - Rod: 22777 nodes
- Multibody representation
  - < 1900 Hz
  - Crankshaft:
    - 2 torsional eigenmodes
    - 305 simulation nodes
  - Rod
    - 4 eigenmodes each
    - 148 simulaltion nodes each
  - Time-integration with gas forces 38 states,~6 cpu-s for 1 s

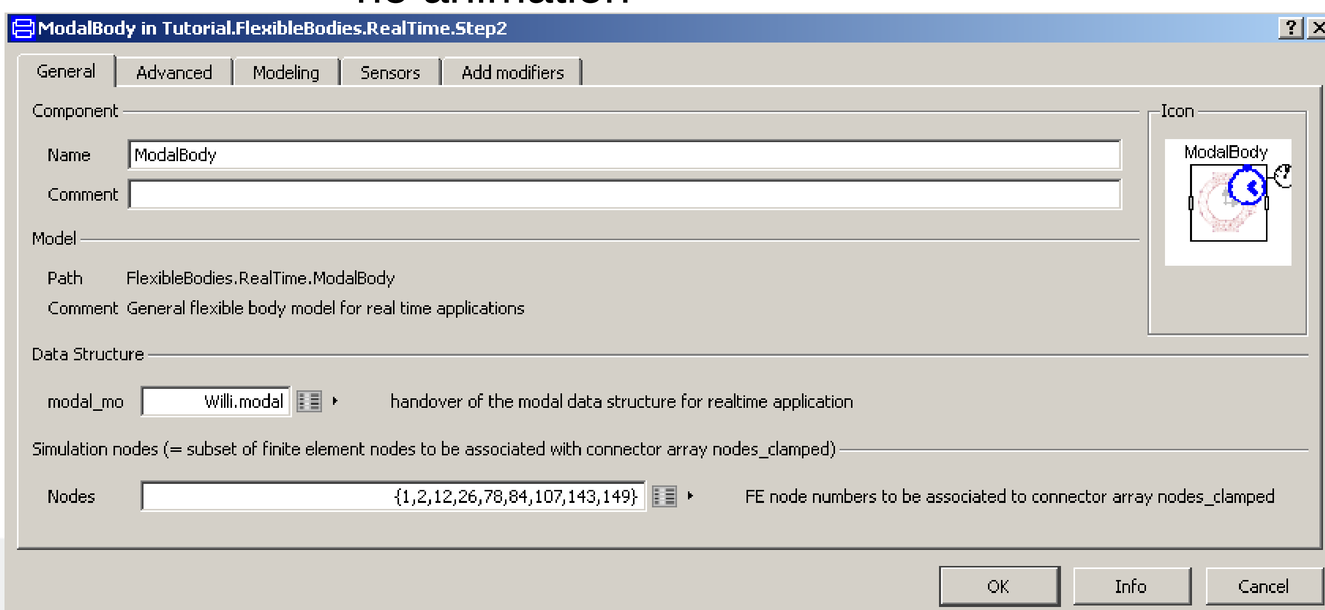


RealTime Modal Body

modalBody



- no external C-Code
  - 2. implementation(= parameter native=true)
  - con‘s: not suitable for large models
- no file access
  - SID-data filed as Modelica-record
  - ⇒ dsmodel.c contains all code and all data
  - no animation



Contents

FE-preprocessing: in summary

1. FE-modelling
2. generate wavefront–file (export mesh-information)
3. prepare and select nodes to retain
4. solve FE-eigenvalue problem
   - care for boundary conditions and frequency range
5. generate FE- substructure
6. generate SID-file FE-from substructure

---

7. introduce SID- and wavefront-file in Modelica

FE-preprocessing Step 2: wavefront-file

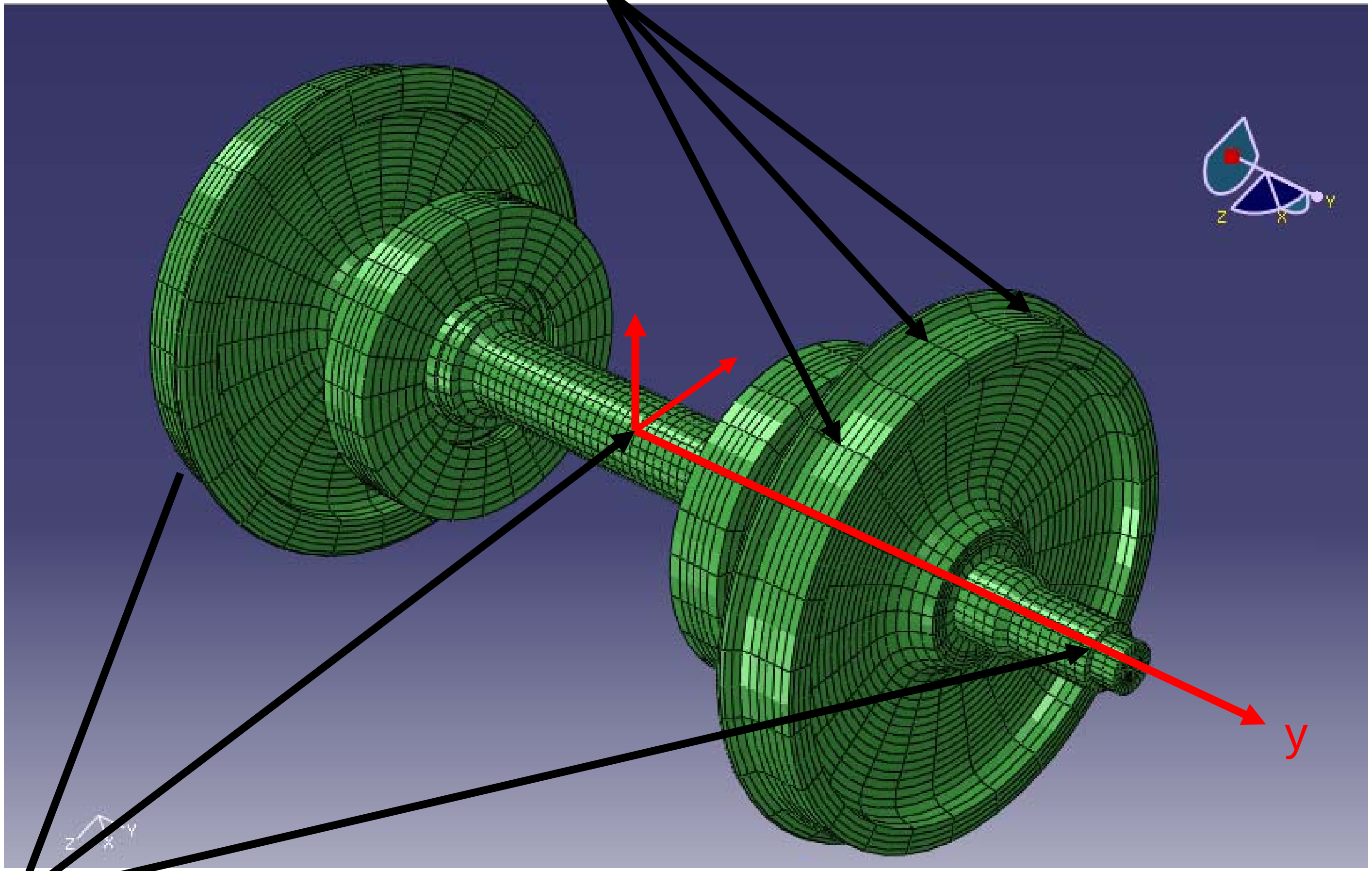
- the animation file in wavefront format *.obj
  - an open (very) low level geometry format
  - freely available tools exist
  - represents geometrical shape of the boby
  - interpolation for animation is completely independent from MBS-simulation
  - due to limited animation performance,
    - the „outside“ geometry is sufficient, e.g. the mesh of the surface

FE-preprocessing Step 3: retained nodes

- retained nodes
  - prepare the body-model for interconnections of the MBS
    - select nodes where MBS-elements are supposed to be attached to
      - define of such nodes and associated MPCs
      - consider rotational degrees of freedom if needed
  - select an additional set of nodes necessary to support a „nice animation“
    - roughly equally distributed over surface of the body
  - in most cases all together 200, 250 retained nodes
  - you may use the specific Abaqus comand line

    *Nset SID_SELECTED_NODES

12 AttachmentPoints at radius 460 y 750 equally distributed at the circumference of each wheel (to introduce wheel/rail forces and torques)



3 AttachmentPoints # 90000, 90003, 90006 on the axis line of the wheelset (to attach suspension and measurements devices)

FE-preprocessing Step 5: substructuring

- standard FE-capability
  - Gyuan-, Craig-Bampton-…..method
- Abaqus comand line

*SUBSTRUCTURE GENERATE, FLEXIBLE BODY=S

SID assumes SI units

- -slength : scaling factor for the length unit (default: 1.0)
- -smass : scaling factor for the mass unit (default: 1.0)
- -stime : scaling factor for the time unit (default: 1.0)

alternative: number of modes

- -fmin : lower boundary of the frequency range (default: 0.001Hz)
- -fmax : higher boundary of the frequency range (default: 1.E16Hz)
- -tol : zero cutoff tolerance (default - 1E-12)
- -help : this usage info

FE-preprocessing Step 6: SID-file-generation

- abqtoSid
  - additionally provided with Abaqus executable control of SID-generation by “substructureName.inp”
    - ASCII-file with keywords e.g.

*NSET

*GENERATE

*BOUNDARY

*SELECT EIGENMODES

Set DEFINITION=MODE NUMBERS / FREQUENCY RANGE

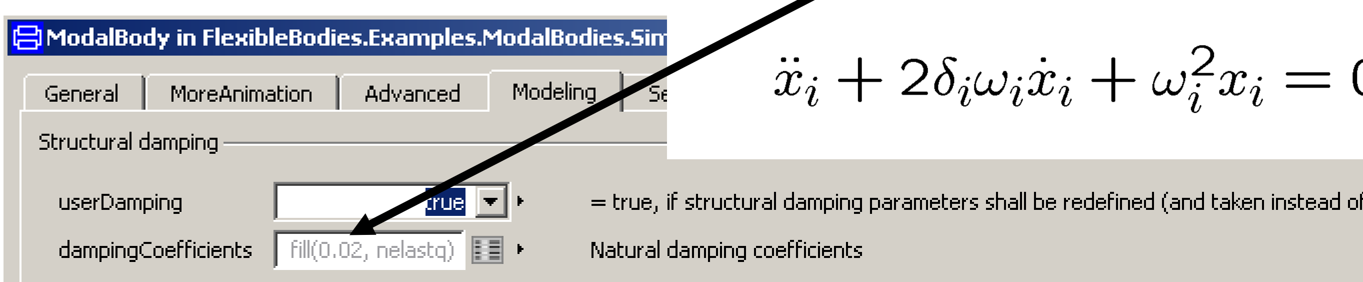
*DAMPING CONTROLS , VISCOUS=FACTOR

*DAMPING, ALPHA=0.0, BETA=0.02

Rayleigh-Damping:
$D = \alpha M + \beta K$

Alternative: natural damping: $D_{ii} = 2\delta_i\sqrt{K_{ii}\cdot M_{ii}} = 2\delta_i\omega_i$

$$\ddot{x}_i + 2\delta_i\omega_i\dot{x}_i + \omega_i^2 x_i = 0$$

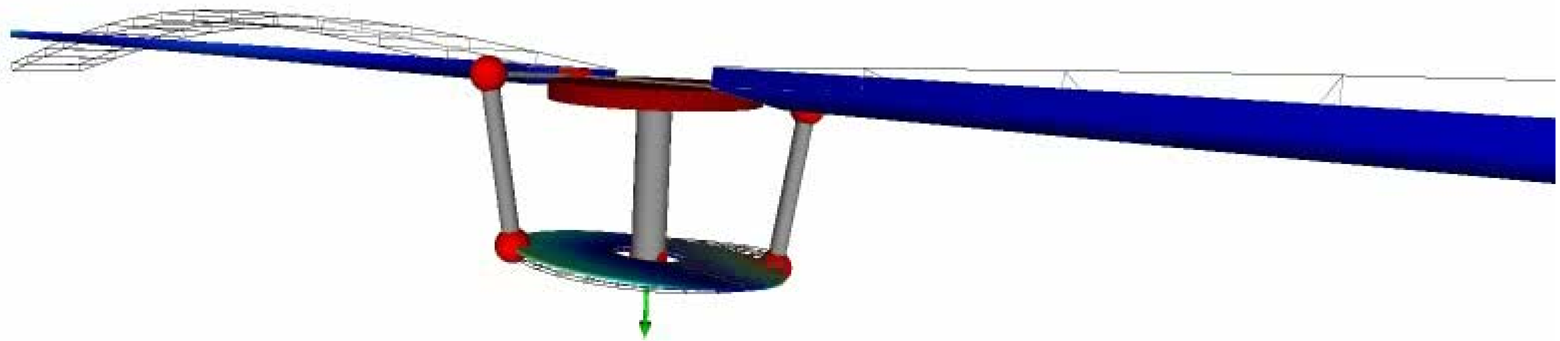


Contents

FlexibleBodies Library extensions at this conference

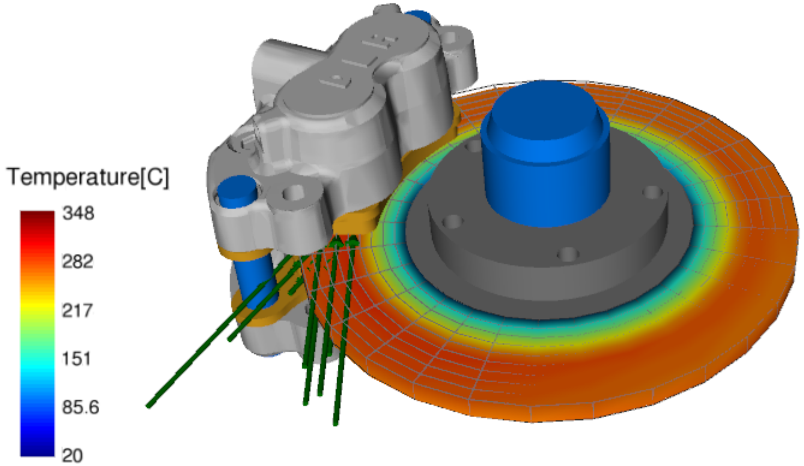
S. Hartweg, Monday HS3 12:00:

- An Annular Plate Model in Arbitrary Lagrangian-Eulerian Description for the DLR FlexibleBodies Library



L . Reyes Perez, Monday HS2 15:35

- A thermoelastic annular plate model for the modeling of brake systems



Thank you very much for your attention !