Modeling of Multi-body Systems

- In this lecture, we shall discuss a few special problems that accompany the modeling of complex mechanical systems.
- We shall demonstrate how these problems have been resolved in Dymola.
- In particular, we shall look at the choice of the state variables and connectors.
- It shall be shown how matrix calculus makes it possible to keep the definitions very compact.

Table of Contents

- Multi-body systems
- Choice of state variables
- Kinematic loops
- Mechanical connectors
- Mechanical bodies
- Mechanical joints
- Example
What is a Multi-body System?

- A multi-body system consists of a combination of mechanical parts, which are connected with each other to support motion in three-dimensional space.

Hmm! Maybe this is not yet the most luxurious model… but abstraction is everything after all.

The Choice of State Variables

- Until now, we have always coupled the differential equations that result from modeling mechanical systems with the masses. This seemed to be meaningful as the D’Alembert Principle had been defined for the individual masses.
- However, a mass, that is rigidly connected with the ground (i.e., with the inertial system), does not lead to any differential equations. Differential equations occur only when the mass moves relative to the inertial system.
- Therefore, it may be better to relate the integrators with the relative motions between bodies. This concept was realized in the multi-body system library (MBS library) of Dymola.
- In the MBS library, the relative positions and relative velocities between bodies that are connected to each other are defined as state variables.
Structural Singularities I

- If the state variables are selected as proposed, no structural singularities occur for tree-structured multi-body systems.
- In the case of **kinematically closed loops** structural singularities do occur, since such structures exhibit a smaller number of mechanical degrees of freedom than connections between neighboring bodies. You may consider for example the **pentagraph**, sometimes also used in flexible lamp fixtures, in expandable trivets or in security doors. It has only one mechanical degree of freedom.

![Kinematically closed loops](image)

Structural Singularities II

- To avoid the structural singularities associated with these loops, one joint of each kinematically closed loop is declared as **cut joint**.

![Cut joints](image)

- Cut joints do not define any integrators, thereby avoiding the introduction of structural singularities. This is more efficient that to rely on the **Pantelides algorithm**.
Algebraic Loops

- *Closed kinematic loops* invariably lead to bad *algebraic loops* in the resulting equation systems. These are usually very large since they extend across all variables of the kinematically closed loop.
- The automatic determination of suitable *tearing variables* is expensive and inefficient.
- The *cut joints* of Dymola contain instructions, which enable the tearing algorithm, to quickly find a set of suitable tearing variables.

\[ \text{constrain}(q, qd, qdd) \]

Choice of Potential- and Flow Variables

- Die MBS library takes the view that the *position* of a body (and thereby also its *velocity* and *acceleration*) represent a *potential*, whereas the *forces*, that act on the body, represent a *flow*. The reverse assumption could have been made alternatively.
- The *inertial system* defines consequently the potential variables und sets them equal to zero (corresponding to the electric potential of the electrical ground node).

\[ q = \begin{bmatrix} x \\ y \\ z \\ \xi \\ \eta \\ \zeta \end{bmatrix} \Rightarrow \begin{align*} q &= 0 \\ qd &= \text{der}(q) = 0 \\ qdd &= \text{der}(qd) = 0 \end{align*} \]
Mechanical Connectors

connector Frame
Position r0[3] "Distance of the frame from the inertial system"
Real S[3, 3] "Transformation matrix of the frame to the inertial system"
Velocity v[3] "Absolute velocity of the frame"
AngularVelocity w[3] "Absolute angular velocity of the frame"
Acceleration a[3] "Absolute acceleration of the frame"
AngularAcceleration z[3] "Absolute angular acceleration of the frame"
flow Force f[3] "Force acting on the frame"
flow Torque t[3] "Torque acting on the frame"
end Frame;

Because of the sign conventions, an empty output frame ( ) must always be connected to a full input frame ( ).

Mechanical Bodies I

- Mechanical bodies define the D’Alembert Principle for the sum of acting forces and torques.

model BodyBase "Inertia and mass properties of a rigid body"
extends Frame a;
Mass m;
Position rCM[3] "Distance from frame to center of gravity"
Inertia I[3, 3];
equation
f = m*(a + cross(z, rCM) + cross(w, cross(w, rCM)));
t = I*z + cross(w, I*w) + cross(rCM, f);
end BodyBase;

The coordinates of the frames are first converted to the center of gravity.
The D’Alembert Principle is then formulated for the center of gravity.
The resulting force f and torque t are finally transformed back to the frame by means of their relative movement under introduction of the accompanying centripetal and Coriolis forces.
Mechanical Bodies II

```plaintext
model Body
    "Rigid body with one cut";
    extends Frame a;
    parameter Position rCM[3]=[0,0,0]
        "Vector from frame_a to center of mass, resolved in frame_a;"
    parameter Mass m=0
        "Mass of body [kg];"
    parameter Inertia I11=0
        "(1,1) element of inertia tensor;"
    parameter Inertia I22=0
        "(2,2) element of inertia tensor;"
    parameter Inertia I33=0
        "(3,3) element of inertia tensor;"
    parameter Inertia I21=0
        "(2,1) element of inertia tensor;"
    parameter Inertia I31=0
        "(3,1) element of inertia tensor;"
    parameter Inertia I32=0
        "(3,2) element of inertia tensor;"
    BodyBase body;
    equation
        connect (frame_a, body.frame_a);
        body.m = m;
        body.rCM = rCM;
        body.I = [I11, I21, I31; I21, I22, I32; I31, I32, I33];
end Body
```

Mechanical Bodies III

```plaintext
model Body
    "Rigid body with one cut";
    extends Frame a;
    parameter Position rCM[3]=[0,0,0]
        "Vector from frame_a to center of mass, resolved in frame_a;"
    parameter Mass m=0
        "Mass of body [kg];"
    parameter Inertia I11=0
        "(1,1) element of inertia tensor;"
    parameter Inertia I22=0
        "(2,2) element of inertia tensor;"
    parameter Inertia I33=0
        "(3,3) element of inertia tensor;"
    parameter Inertia I21=0
        "(2,1) element of inertia tensor;"
    parameter Inertia I31=0
        "(3,1) element of inertia tensor;"
    parameter Inertia I32=0
        "(3,2) element of inertia tensor;"
    BodyBase body;
    equation
        connect (frame_a, body.frame_a);
        body.m = m;
        body.rCM = rCM;
        body.I = [I11, I21, I31; I21, I22, I32; I31, I32, I33];
end Body
```
Bodies with more than two joints have to be constructed by the modeler using additional frame translations. Such elements are not available in the MBS library as pre-designed modules.

Geometry for the computation of mass and inertia matrix (not represented graphically, since modeled by means of equations).
Mechanical Bodies VI

model BoxBody "Rigid body with box shape (also used for animation)"

extends MultiBody.Interfaces.TwoTreeFrames;

parameter SIunits.Position r[3]={0.1,0,0} "Vector from frame_a to frame_b, resolved in frame_a";
parameter SIunits.Position r0[3]={0,0,0} "Vector from frame_a to left box plane, resolved in frame_a";
parameter SIunits.Position LengthDirection[3]=r - r0 "Vector in length direction, resolved in frame_a";
parameter SIunits.Position WidthDirection[3]={0,1,0} "Vector in width direction, resolved in frame_a";
parameter SIunits.Length Length=(sqrt((r - r0)*(r - r0))) "Length of box";
parameter SIunits.Length Width=0.1 "Width of box";
parameter SIunits.Length Height=0.1 "Height of box";
parameter SIunits.Length InnerWidth=0 "Width of inner box surface";
parameter SIunits.Length InnerHeight=0 "Height of inner box surface";
parameter Real rho=7.7 "Density of box material [g/cm^3]";
parameter Real Material[4]={1,0,0,0.5} "Color and specular coefficient";

SIunits.Mass mo, mi;

Real Sbox[3, 3];
SIunits.Length l, w, h, wi, hi;
FrameTranslation frameTranslation(r=r);
MultiBody.Interfaces.BodyBase body;
VisualShape box (Shape="box", r0=r0, LengthDirection=LengthDirection, WidthDirection=WidthDirection, Length=Length, Width=Width, Height=Height, Material=Material);

end BoxBody

Mechanical Bodies VII

equation

connect (body.frame_a, frame_a);
connect (frame_a, frameTranslation.frame_a);
connect (frameTranslation.frame_b, frame_b);

box.S = Sb;
box.r = r0a;
box.Shape = Sbox;

l = Length;
w = Width;
h = Height;
wi = InnerWidth;
hi = InnerHeight;

/*Mass properties of box*/

mo = 1000*rho*l*w*h;
mi = 1000*rho*wi*h;

body.m = mo - mi;

Rbox = r0 + 1/2*box.nLength;
body.I = Sbox*diagonal({mo*(w*w + h*h) - mi*(wi*wi + hi*hi),mo*(l*l + h*h) - mi*(l*l + hi*hi),mo*(l*l + w*w) - mi*(l*l + wi*wi)/12})*transpose(Sbox);

end BoxBody
Mechanical Joints I

model Prismatic "Prismatic joint (1 degree-of-freedom, used in spanning tree)"
  extends MultiBody.Interfaces.TreeJoint;
  parameter Real n[3]=[1,0,0]
  "Axis of translation resolved in frame a (= same as in frame b)");
  parameter SIunits.Position q0=0 "Relative distance offset(see info)"
  "true, if start values of q, qd are fixed"
  SIunits.Position q[final fixed=startValueFixed];
  SIunits.Velocity qd[final fixed=startValueFixed];
  SIunits.Acceleration qdd;
  SIunits.Position q q;
  SIunits.Velocity vaux[3];
end Prismatic;

connector Modelica.Mechanics.Translational.Interfaces.Flange_a axis;

Mechanical Joints II

equation
  axis.s = q;
  bearing.s = 0;
  axis.f = nn*fb;

  // define states
  q = der(q);
  qd = der(qd);
  q = q - q0;

  /*normalize axis vector*/
  nn = n/sqrt(n*n);
  /*kinematic quantities*/
  S_rel = identity(3);
  qq = q - q0;
  r_rela = nn*qq;
  vr_rela = nn*qd;
  wz_rela = zeros(3);
  wz_rela = zeros(3);
  wz_rela = zeros(3);
end Prismatic;
Causalities I

• The causality of the equations depends on the posed problem.
• In the **direct problem** (the simulation problem), the forces and torques are given, whereas the resulting movement is to be computed.
• In the **inverse problem** (the planning problem), the desired movements are predetermined, whereas the forces and torques, that are needed to produce the desired movements, are to be found.

Causalities II

• The efficiency of the generated code depends heavily on the formulation of the equations. Small changes of the formulation can modify the efficiency such the the number of resulting equations at the end grows either linearly with the number of bodies or with their fourth power.
• For this reason, the MBS library doesn’t rely on the automatic transformation of the equations by use of the Pantelides algorithm and the previously presented heuristics for finding small sets of tearing variables.
Causalities III

- **Matrix calculus**, as used so far, is very elegant and compact and therefore well suited for maintenance of the MBS library. However, this notation is not suitable for the automatic transformation of the resulting equations.

- For this reason, all matrix equations are symbolically expanded by *Modelica* into scalar equations prior to the determination of the correct causalities.
An Example II

An Example III

Cut joint

Kinematic loop
An Example IV

Equations after expansion of the matrix expressions

Elimination of trivial equations of the type: $a = b$

Remaining equations after the symbolic transformation.

An Example V
References