

PlanarMultiBody

A Modelica Library for Planar Multi-Body Systems

Mathias Höbinger¹, Martin Otter²

¹Vienna University of Technology, Austria

²German Aerospace Center (DLR), Institute of Robotics and Mechatronics, Germany
mathias.hoebinger@gmx.at, martin.otter@dlr.de

Abstract

A new Modelica library for the modeling and simulation of 2-dimensional mechanical systems has been developed. It is based on the existing Modelica.Mechanics.MultiBody library and implements a number of simplifications and optimizations for 2-dimensional environments, which bring the advantages of a reduced complexity of the modeling process as well as a reduced computational effort. Additionally, new components are present for joints with curve-curve contact (e.g. cam follower joints). The basic approach is, to have a 1:1 mapping of packages, models and functions, if this makes sense, and specialising them to 2 dimensions.

Keywords:

Modelica, planar multi-body, contact mechanics.

1 Introduction

The *PlanarMultiBody* library is a Modelica package providing 2-dimensional mechanical components to model in a convenient way planar mechanical systems. The main design goal of the library was to utilize the fact that in such systems coordinates, directions and rotations can be expressed and computed in a much simpler way than in 3-dimensional systems.

A typical example of this library is a mechanism with 2 kinematic loops as shown in the Figure 1.

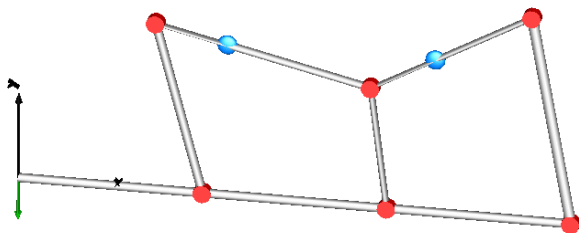
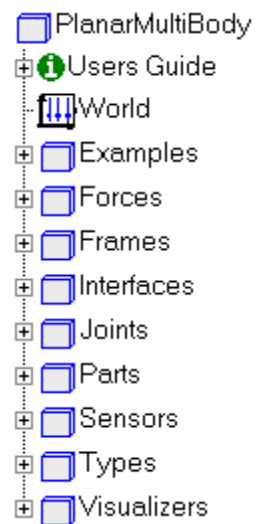


Figure 1: A planar mechanical system containing 2 coupled kinematic loops

The PlanarMultiBody library, see screenshot to the right, has the following main features:

- In 2-dimensional systems, the orientation of any object with respect to another one can be described by a single angle. This simplifies the notation for orientation of objects considerably. The use of the “orientation objects” from the Modelica.Mechanics.MultiBody library can be dropped completely, as well as the special handling of the orientation object with Connections.Root(..), Connections.Branch(..) operators to define the connected network of coordinate systems in order to handle over-determined DAEs. The requirements for a Modelica translator to process models of this library are therefore much less as for the 3-dim. Modelica.Mechanics.MultiBody library.
- The visualizer objects used in the MultiBody library for the animation of objects have been altered to achieve two aims: Firstly, all animation objects can be addressed as 2D objects, e.g., the bars used to animate a fixed translation have a length and a width, no height. The *Visualizers.Advanced.Shape* object, as well as the objects used for animating all kinds of arrows, includes input values for length, width and position. Secondly, because the actual animated shapes are still 3D-objects, the height is automatically set to a very low value which gives the animation a “pseudo-planar” look.
- The possibility to model joints based on two curves sliding along each other. In model PlanarMultiBody.Joints.CurveCurveJoint, different curve objects can be selected. They all contain functions used to compute three vectors depending on a curve parameter s : the *curvePosition*, the *curveTangent* and the *curveNormal*. The Planar-



MultiBody.Joints.CurveCurveJoint object includes two instances of arbitrary curve objects, each connected to a frame. This joint constrains the movement of its two frames by requiring proper contact conditions for the two curves. These are computed using the two curve parameters $s1$ and $s2$. Additional curves needed by a user can easily be added by just providing the necessary equations of the curve and its normal and tangent vectors.

2 Describing Orientation

The simplified way of describing absolute and relative orientation of objects is the most significant improvement for modeling planar systems compared to model the same system using the 3-dimensional MultiBody library. For notational convenience the word “frame” is used in the sequel as a synonym for “coordinate system”. Instead of using three orthogonal unit vectors to define a specific frame we can do that with a single angle φ that describes the rotation of that frame with respect to the global coordinate system around the only possible axis of rotation, the z-axis. To define the position and rotation of a second frame relative to the first one is equally simple: a two-dimensional vector r_{rel} and a relative angle φ_{rel} are everything that is needed. Given the absolute angles φ_a and φ_b of two different frames, the relative angle can be computed by simply stating $\varphi_{rel} = \varphi_b - \varphi_a$.

3 PlanarMultiBody Frame Connector

The “Frame” connector is used to connect planar multibody components together. It is rigidly fixed at an attachment point of a mechanical part. A frame “frame a” is described with respect to the world frame using the

- 2-element vector r that is directed from the origin of the world frame to the origin of frame a and is resolved in the world frame and by the
- angle φ between the x-axis of the frame and the x-axis of the world-frame.

It is assumed that a free body diagram is constructed, i.e. that a cut is performed between mechanical parts that shall be connected together at frame a. In the cut plane a resultant cut force \mathbf{f}_a and resultant cut torque τ_a act on frame a. Since in planar multi-body systems there are no advantages to express vectors in local frames, all vectors, and especially \mathbf{f}_a , are expressed in

the world-frame. The resultant cut torque is a scalar along the z-axis of the world-frame. To summarize, the connector is defined as:

```
connector Frame
  import SI = Modelica.SIunits;
  SI.Position r[2]
    "Absolute position vector";
  SI.Angle phi
    "Angle from x-axis world to frame";
  flow SI.Force f[2]
    "Constraint force in world frame";
  flow SI.Torque t
    "Constraint torque in world frame";
end Frame;
```

As usual, if velocities or accelerations are needed, they can be obtained by applying the derivative operator $der(...)$. This also holds for the angular velocity which is simply $der(phi)$, where as in the Modelica.Mechanics.MultiBody library the computation of the angular velocity is complicated and is performed with a function.

4 Elementary Components

Using the “Frame” connector and the utility functions in PlanarMultiBody.Frames, it is straightforward to implement the elementary components that are usually available in multi-body programs. The PlanarMultiBody library has about 40 components. The most important ones are shown in Table 1. Exactly like in the Modelica.Mechanics.MultiBody library, equations are only defined on “position” level.

4.1 PlanarMultiBody.World

This model represents a global coordinate system fixed in ground. It is used as inertial system in which the equations of all elements of the PlanarMultiBody library are defined and is the world frame of an animation window in which all elements of the PlanarMultiBody library are visualized. Furthermore, the gravity field of the multi-body model is defined here. Default is a uniform gravity field; a point gravity field can also be selected. The world object is also used to define default settings of animation properties (e.g. the width of the rectangles representing a revolute joint). The world object itself is animated as a coordinate system with 2 axes and labels.

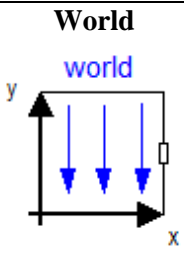
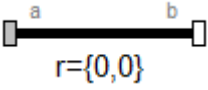
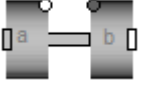
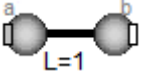
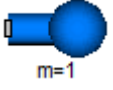
Abbreviations:	
$\mathbf{r}_a, \varphi_a, \mathbf{f}_a, \boldsymbol{\tau}_a := \text{frame_a.r, .phi, .f, .t}$ $\mathbf{r}_b, \varphi_b, \mathbf{f}_b, \boldsymbol{\tau}_b := \text{frame_b.r, .phi, .f, .t}$ $\text{resolve1}(\dots) := \text{Frames.resolve1}(\dots)$ $\text{grav} := \text{world.gravityAcceleration}(\dots)$	
World 	$\mathbf{r}_b = \mathbf{0}$ $\varphi_b = 0$
Parts.Fixed Translation 	$\mathbf{r}_b = \mathbf{r}_a + \text{resolve1}(\varphi_a, \mathbf{r}_{\text{rel}})$ $\varphi_b = \varphi_a$ $0 = \mathbf{f}_a + \mathbf{f}_b$ $0 = \boldsymbol{\tau}_a + \boldsymbol{\tau}_b + \mathbf{r}_{\text{rel}} \times \mathbf{f}_b$
Joints.Revolute 	$\mathbf{r}_b = \mathbf{r}_a$ $\varphi_b = \varphi_a + \varphi_{\text{rel}}$ $0 = \mathbf{f}_a + \mathbf{f}_b$ $0 = \boldsymbol{\tau}_a + \boldsymbol{\tau}_b$
Joints.JointRR 	$\mathbf{r}_{\text{rel0}} = \mathbf{r}_b - \mathbf{r}_a$ $L * L = \mathbf{r}_{\text{rel0}} * \mathbf{r}_{\text{rel0}}$ $0 = \mathbf{f}_a + \mathbf{f}_b$ $\mathbf{f}_a = \mathbf{f}_{\text{rod}} * \mathbf{r}_{\text{rel0}} / L$ $0 = \boldsymbol{\tau}_a$ $0 = \boldsymbol{\tau}_b$
Parts.Body 	$w = \text{der}(\varphi_a)$ $z = \text{der}(w)$ $\mathbf{r}_{\text{CM0}} = \text{resolve1}(\varphi_a, \mathbf{r}_{\text{CM}})$ $\mathbf{r}_{\text{absCM0}} = \mathbf{r}_a + \mathbf{r}_{\text{CM0}}$ $\mathbf{g} = \text{grav}(\mathbf{r}_{\text{absCM0}})$ $\mathbf{v} = \text{der}(\mathbf{r}_{\text{absCM0}})$ $\mathbf{a} = \text{der}(\mathbf{v})$ $\mathbf{f}_a = m * (\mathbf{a} - \mathbf{g})$ $I * z = \boldsymbol{\tau}_a - \mathbf{r}_{\text{CM0}} \times \mathbf{f}_a$

Table 1: Elementary components of PlanarMultiBody.

4.2 PlanarMultiBody.Parts.FixedTranslation

This component defines a fixed translation of a frame. It is, e.g., used to define frames for several attachment points on a body. The equations state that the position vector of frame_b is defined from the position vector of frame_a and the relative position vector \mathbf{r}_{rel} from frame_a to frame_b (\mathbf{r}_{rel} is defined as parameter “r”). Since frames are translated, the angles in the two frames are set equal. Finally, a force

and torque balance of this massless part is present in the Modelica model.

4.3 PlanarMultiBody.Joints.Revolute

In planar systems, the only possible axis of rotation is the z-axis, so this component always defines such a rotation using a vector φ_{rel} . When $\varphi_{\text{rel}} = 0$, frame_a and frame_b coincide. Unlike in the Modelica.Mechanics.MultiBody library, the absolute orientation vector of frame_b, frame_b.phi, can easily be obtained by stating

$$\text{frame_b.phi} = \text{frame_a.phi} + \varphi_{\text{rel}}.$$

As with most other joints, the generalized coordinates (here: φ_{rel} and its derivative ω_{rel}) have the attribute *stateSelect* = *StateSelect.prefer* in order that they are selected as states if possible. The position vectors of the two frames are identical and there is a force and torque balance present. Instead of implementing an additional model “ActuatedRevolute”, a conditional 1-dim. flange connector is present onto which a drive train can be attached driving the revolute joint, e.g, with components from the Modelica.Mechanics.Rotational library. There is a Boolean parameter *drivenFlange* present to activate or deactivate the additional flange.

4.4 PlanarMultiBody.Parts.Body

This component defines the mass and inertia properties of a body. They are defined using the following parameters: *m* for the mass, the position vector r_{CM} from the origin of frame_a to the center of mass (resolved in frame_a) and the inertia value *I*. There is a Boolean parameter *enforceStates* present which defines if the position vector *r* and orientation angle φ of frame_a should be used as states. These variables have the attribute *stateSelect* = *if enforceStates then StateSelect.always else StateSelect.avoid*. The feature to have potential states both in joints and in bodies makes it easier to model systems with bodies which are connected to the environment without using a joint or freely moving bodies.

4.5 PlanarMultiBody.Joints.JointRR

This component fixes the distance between its two frames to parameter *L*, but does not constrain the orientation angles of any of them. Therefore it can be used as a replacement for two revolute joints connected by a fixed translation. Using this component reduces the order of the nonlinear equation system and helps avoiding problems with non-linear equation systems caused by kinematic loops. The cut force is constrained to act only along the vector be-

tween the origins of the two frames. Finally, a force and torque balance is present in this component. There is an additional object called `PlanarMultiBody.Joints.JointRRWithMass` present which includes a mass fixed relative to the two frames of the joint.

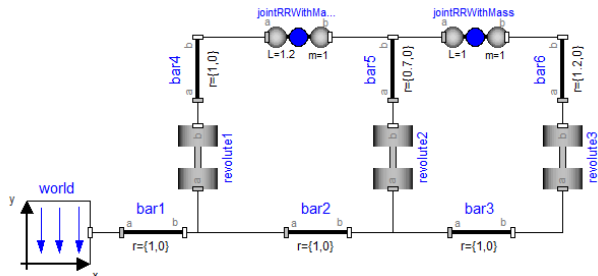


Figure 2: The diagram level of the model animated in Figure 1 using two instances of `JointRRWithMass`

5 Force Elements

Force elements exert forces and torques between two frames. Because these elements, although they have obviously been altered to fit into the different orientation setup of this new library, are virtually identical in their functionality and structure to the ones in the `MultiBody` library, we will not discuss them here in great detail. For a more detailed description of the most important force elements, see [1].

6 Animation

The animation environment in `Dymola` [2] is natively a 3-dimensional one, and all animated objects therefore have to be programmed in that way. However, the `Modelica.Mechanics.MultiBody` library

utilizes a single model to realize virtually of all its animations, `MultiBody.Visualizers.Advanced.Shape`. The following features were implemented into the `PlanarMultiBody` animation engine:

- Having a user-interface with purely 2-dimensional animation parameters gives the user the convenience of not having to deal with a z-coordinate that only exists in the animation and has nothing to do with the planar system being modeled.
- To provide users with a maximum of freedom of design, either side of a 3d-object displayed by the “FixedFrame” component of the library can be used as a “pseudo-2d” object. E.g. a cylinder can be used as a circle or a rectangle. For this purpose, a boolean parameter “zDirection” was added to the `Shape` object which rotates the animated object by 90° around the y-axis.
- To avoid overlapping of objects in the “pseudo-2D” animation, it is possible to shift an object along the z-axis of the animation using the parameter “heightShift”.
- The height of all objects is automatically set to a low value which results in the desired “pseudo-2D” look of the animation.

Table 2 shows all the parameters of the `PlanarMultiBody.Visualizers.Advanced.Shape` object with their default values and a short description of their functionality.

Type	Name	Default	Description
ShapeType	shapeType	"box"	Type of shape (box, sphere, cylinder, pipecylinder, cone, pipe, beam, gearwheel, spring)
Boolean	zDirection	false	= true, if shape object should be rotated 90° around the y-axis
Angle	phi	0	Angle to rotate the world frame into the object frame [rad]
Position	r[2]	{0,0}	Position vector from origin of world frame to origin of object frame, resolved in world frame [m]
Position	r_shape[2]	{0,0}	Position vector from origin of object frame to shape origin, resolved in object frame [m]
Real	lengthDirection[2]	{1,0}	Vector in length direction, resolved in object frame
Length	length	0	Length of visual object [m]
Length	width	0	Width of visual object [m]
ShapeExtra	extra	0.0	Additional size data for some of the shape types
Real	color[3]	{255,0,0}	Color of shape
SpecularCoefficient	specularCoefficient	0.7	Reflection of ambient light (= 0: light is completely absorbed)
Length	heightShift	0	used if object should be shifted by {0,0,heightShift} in the Animation to e.g. avoid overlapping [m]

Table 2: Parameters of the `PlanarMultiBody.Visualizers.Advanced.Shape` object

7 Curve-Curve Contact

With `Joints.CurveCurveJoint`, the `PlanarMultiBody` library includes a new joint making it possible to simulate two surfaces having to remain in contact with each other. In every instance of this joint, the user can choose two out of a library of curves used to simulate the connected surfaces. Each curve is fixed to one frame of the joint, in the sequel we will use the name `curve_1` for the curve object connected to `frame_a` and `curve_2` for the one connected to `frame_b`. The main idea is to have two variables s_1 and s_2 , one for each curve, in the `CurveCurveJoint` model, which stand for the path parameter of the respective curve, describing the current contact point on the curve with respect to a fixed starting point. Usually “ s ” is the arc-length along the curve, but this need not to be the case in general. For a given value of their respective curve-variables, `curve_1` returns a relative position vector from `frame_a` to the point of contact as well as the normal and tangent vector at that point on the curve.

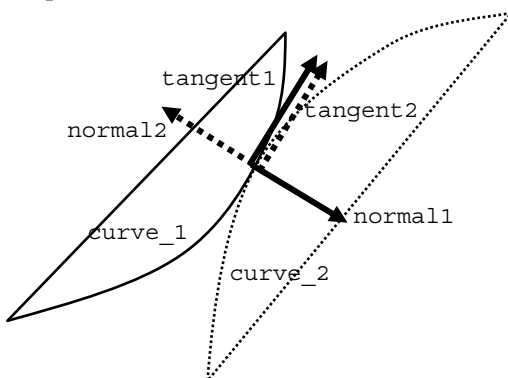
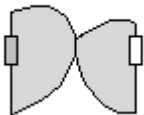


Figure 3: Normal and tangent definition of curve-curve contact

7.1 Joints.CurveCurveJoint

curveCurveJoint



As mentioned above, this model includes two frames as well as two instances of a “curve object”. The possibility of choosing the curves inside the

actual instance of the joint is realized by including them as “replaceable” objects:

```
replaceable Joints.Internal.Circle
  curve1(phi=frame_a.phi,r_0=frame_a.r)
  extends
    PlanarMultiBody.Interfaces.BaseCurve
      (phi=frame_a.phi, r_0=frame_a.r)
```

In the equation section of the `CurveCurveJoint` model, position, normal and tangent variables are

connected to the respective variables in the curve objects.

```
r1_rel = curve1.position(s1);
r2_rel = curve2.position(s2);
r1 = Frames.resolve1(frame_a.phi,r1_rel);
r2 = Frames.resolve1(frame_b.phi,r2_rel);
normal1 = Frames.resolve1(frame_a.phi,
  curve1.normal(s1));
normal2 = Frames.resolve1(frame_b.phi,
  curve2.normal(s2));
tangent2 = Frames.resolve1(frame_b.phi,
  curve2.tangent(s2));
```

More importantly, the kinematic constraint equations as well as the force and torque balances of the joint and the curves are defined here:

First, the distance between the contact point on `curve_1` and the one on `curve_2` is set to zero:

```
{0,} = frame_b.r + r2 - (frame_a.r + r1);
```

Then, additional equations ensure that the contact point is actually an osculation point of the two curves, meaning that their standard normal vectors point in the same direction with different signs:

```
0 = Modelica.Math.atan2(
  normal1*tangent2, -normal1*normal2);
```

The formulation of this condition is from Hans Olsson [3] and requires some explanation: The contact conditions on the normal could be formulated as “ $normal1*normal2=0$ ”. However, this equation has a singular Jacobian and therefore every solver would have severe difficulties. The condition could also be formulated as “ $normal1*tangent2 = 0$ ”, as often suggested in literature. Here, we have the problem that a contact where `normal1` and `normal2` are directed in the same direction, will also fulfill this equation and therefore it can happen that during simulation suddenly a wrong contact appears. The formulation used in the `CurveCurveJoint` is basically using the “ $normal1*tangent2 = 0$ ” formulation, but uses this as the first argument to the “`atan2(..)`” function. As second argument “ $-normal1*normal2$ ” is used. The “`atan2(..)`” function has the property that the signs of the two arguments determine the quadrant of the solution. Especially, only if the second argument is positive, $-\pi/2 \leq atan2(x,y) \leq \pi/2$. Therefore, in the solution point “ $0 = atan2(x,y)$ ”, the second argument “ $-normal1*normal2$ ” must be positive which means that the two normal’s have to be directed in opposite direction.

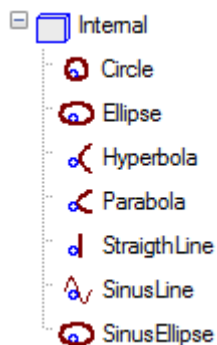
Finally, force and torque balances are included:

```
// Force and torque balance of joint
zeros(2) = frame_a.f + frame_b.f;
0 = frame_a.t + frame_b.t +
    Frames.cross(frame_b.r - frame_a.r,
                frame_b.f)

// Force and torque balance of curve1
f_contact1 = -normal1*f_N;
zeros(2) = frame_a.f + f_contact1;
0 = frame_a.t +
    Frames.cross(r1, f_contact1);
```

7.2 Predefined Contact Curves

The package `PlanarMultiBody.Joints.Internal` includes the models which are predefined in the `CurveCurveJoint` object. Additional curves can easily be added by a user. We will use the `Ellipse` model to explain the functionality of these objects. All curve-definition models extend a model called `PlanarMultiBody.Interfaces.BaseCurve` which defines the basic input variables r_0 and ϕ which are the absolute position vector and orientation angle of the frame to which the curve is attached.



The `BaseCurve` model also establishes the three functions *position*, *normal* and *tangent* and their basic input and output variables. The input variable s is the curve parameter; the 2-dimensional output vector is called r , n or t depending on the function. To enable the different curve-definition models to have different versions of these functions, they are defined as “replaceable encapsulated partial functions” in `BaseCurve`.

```
replaceable encapsulated partial
function normal
  input Real s "Curve parameter";
  output Real n[2] "Normal to curve";
end normal;
```

Every curve model has its own set of parameters used to adjust the actual curve surface. In case of the ellipse there are two of them: a and b , defining the length of the two ellipse-axis.

Furthermore, there is always at least one parameter defining the path parameter of the animated curve. In case of the ellipse, the final parameter C is the approximated circumference of the Ellipse computed from the given parameters a and b . In the models which define non-closed curves, e.g. “`StraightLine`”, there is an input parameter instead of this final parameter allowing the user to define how long a part of the curve should be animated.

Additionally, there are the usual animation-concerned parameters *animation*, switching the animation of the curve on or off, and *color*, defining the color of the animated curve. Finally, the parameter *ns* defines how many points should be used to interpolate the animated curve and the *SwitchSide* parameter defines on which side of the curve the contact should occur.

The most important part of a curve-definition model are of course the three functions actually defining the shape of the curve: *curvePosition*, *curveNormal* and *curveTangent*. They extend the respective functions in the `BaseCurve` model by including the necessary additional parameters and adding an “algorithm” section with the statement computing their output variable. Here we present the `CurvePosition` function from the `Ellipse` model as an example:

```
model Ellipse "Ellipse contact curve"
  extends
    PlanarMultiBody.Interfaces.BaseCurve (
      redeclare final function position =
        curvePosition(a=a,b=b,C=C),
      redeclare final function normal =
        curveNormal(a=a,b=b,C=C,sw=sw),
      redeclare final function tangent =
        curveTangent(a=a,b=b,C=C));
  protected
  function curvePosition
    extends PlanarMultiBody.Interfaces.
      BaseCurve.position;
  input Modelica.SIunits.Length a
    "Length of a-axis of ellipse";
  input Modelica.SIunits.Length b
    "Length of b-axis of ellipse";
  input Modelica.SIunits.Length C
    "Approximated circumference";
  algorithm
    r := { a*sin(s*2*pi/C),
          -b*cos(s*2*pi/C) };
  end curvePosition;
  ...
end Ellipse;
```

Finally, the model includes an algorithm computing the points used to animate the curve in its current position defined through the curve parameter. This is done by filling three coordinate vectors with length ns . These vectors are actually realized as $ns \times 2$ matrices, the second columns being filled with slightly shifted values to ensure better visibility of the animated curve. The animation is performed with `Dymola`’s built-in support for parameterized surfaces.

```

final parameter Real s_min=0
    "Minimum value of s";
final parameter Real s_max=C
    "Maximum value of s";
algorithm
for i in 1:ns loop
    s := s_min + (i - 1)*
        (s_max - s_min)/(ns - 1);
    r := Frames.resolve1(phi,
        position(s));
    x[i,1] := r_0[1] + r[1];
    x[i,2] := r_0[1] + r[1] + 0.01;
    y[i,1] := r_0[2] + r[2];
    y[i,2] := r_0[2] + r[2] + 0.01;
    z[i,1] := 0;
    z[i,2] := 0.01;
end for;
    
```

7.3 Examples

Package `PlanarMultiBody.Examples.CurveCurveJoint` includes a number of examples demonstrating the use of this new joint. The most obvious example is probably the classic Cam-Follower setup. In this model, an elliptic object driven by gravity acting upon a body attached to it turns on a revolute joint fixed to the ground. It is connected to an object with a straight surface being attached to a prismatic joint and forced into movement by the ellipsoid (see model schematic and animation in next Figure 4).

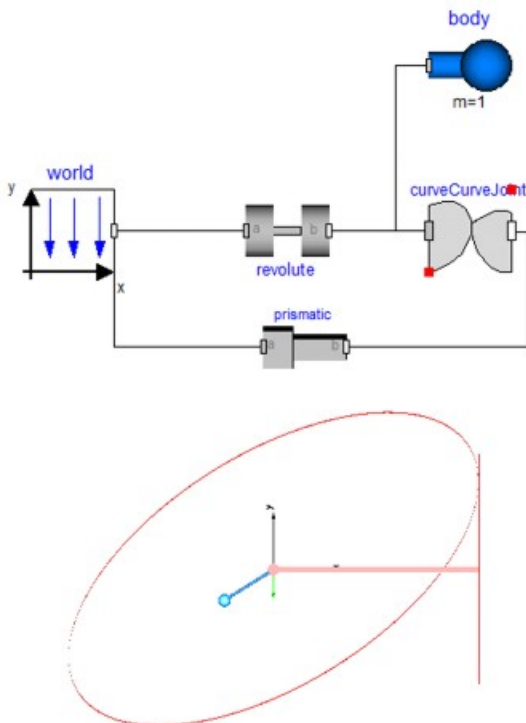


Figure 4: Model and animation of CamFollower

It is realized by connecting `frame_a` of a `CurveCurveJoint` to the world frame through a revolute joint and doing the same with `frame_b` using a prismatic

joint. Then the appropriate curves have to be selected by double clicking on the joint and selecting them from a dropdown menu (see next Figure 5).

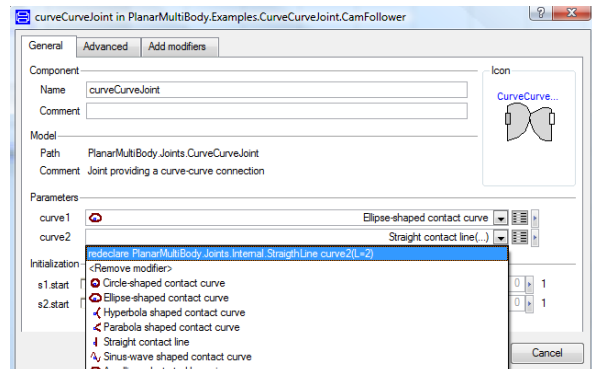


Figure 5: Selecting a curve in the CurveCurveJoint menu

Finally, a body is attached to `frame_a` of the joint and the start value of the ellipses curve parameter is set to an appropriate value to ensure that the system is not in an idle position at time 0.

Another example from this package demonstrates the effect of the `switchSide` parameter, see Figure 6. Two `CurveCurveJoint` objects are present, both describing the contact between two circles. In the upper circle-circle contact, `switchSide = true`, whereas in the lower circle-circle contact, the default `switchSide = false` is used. The effect can be seen in Figure 6.

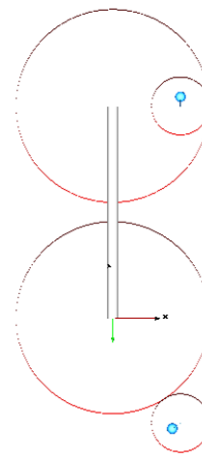


Figure 6: Example CurveCurveJointSwitchSides demonstrating the switchSide parameter

The third example, see Figure 7, shows the possibility of more complex curves by using an ellipse distorted by a sinus wave. This curve has amplitude and frequency of the wave as additional parameters. Here, a very small circle attached to a small body runs along the distorted ellipse. It is connected to the world frame using a prismatic joint.

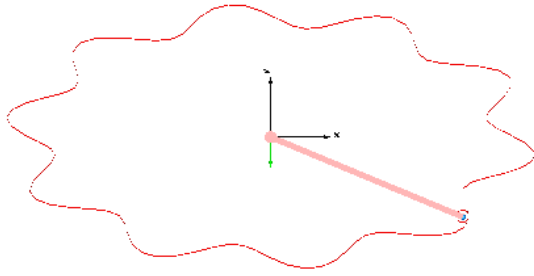


Figure 7: Example SinusEllipse demonstrating more complicated curve-curve contacts

8 Conclusions

The PlanarMultiBody library is a mechanical library to model planar mechanical systems. The main advantage is its simplicity and that no special symbolic manipulation features of the Modelica simulation environment is needed, contrary to the Modelica.Mechanics.MultiBody library that describes 3-dim. mechanical systems. Therefore, the PlanarMultiBody library is well suited for teaching, but also for a quite large class of technical problems that are 2-dim. in nature. Besides standard joints, the PlanarMultiBody library allows the definition of curve-curve contacts, especially to describe cam-follower types of contact. The non-standard formulation [3] of the contact condition with the atan2(.) function has proven to result in reliable solutions of the occurring non-linear algebraic equation systems.

It is planned to include this library as free package in the Modelica Standard Library after an evaluation phase. Currently, there is also an Interpolation package under development. Once available, it is planned that the curve descriptions in the curve-curve contact description can be optionally described by splines of this package.

References

- [1] Otter M., Elmqvist H., Mattson S.E. **The new Modelica Multibody Library**. Proc. of the 3rd International Modelica Conference, pp. 311-330, 2003.
http://www.modelica.org/Conference2003/papers/h37_Otter_multibody.pdf
- [2] Dynasim. **Dymola Users Guide**, Version 6.0, <http://www.dynasim.se>.
- [3] Olsson H.: **Formulation of contact conditions**. Personal communication to M. Otter, Sept. 2007.