Point-to-Curve Constraints and other Contact Elements

Volker Beuter
Kämmerer AG
Wettergasse 18, 35037 Marburg
v.beuter@kaemmerer-group.com

Abstract

The `MultiBody` package of the Modelica Standard Library (MSL) contains a Prismatic Joint model with two `Frame` connectors where one frame can move with respect to the other along a direction *n*. This can be viewed as that the second frame can move along a straight line fixed in the first frame. In this work a constraint is developed where this line is replaced by some curve described by some suitable geometry: A Point-to-Curve (PtCv) constraint. It turns out that there are several options to define the orientation of the second frame with respect to the first one. Additional degrees of freedom are possible. These ideas can be applied to 2D: A Point-to-Surface (PtSf) constraint . PtCv and PtSf constraints seem to be suitable building blocks for higher order constraints: Curve-to-Curve (CvCv) resp. Surface-to-Surface (SfSf) constraints. As a by-product there are some Joint models not yet available in the MSL at all or not in that form, like an elementary Cylindrical joint.

Keywords: point-to-curve contact; osculating circle; point-to-surface contact

1 Introduction

Part of Kämmerer's involvement in the Eurosyslib project[4] is the development of a package with models for building convertible car roofs. One type of components of such a convertible roof are mechanisms where some kind of roller can move within something like a track. When we can abstract from effects like backlash and collisions with the bearings we can view this as a constraint with one translational degree of freedom. In this abstraction a point can move along a curve. The curve does not need to be fixed in space, it may be moving, but it is *rigid*. Conceptualized in Modelica MultiBody package terms this is a model with two `Frame` connectors. When we describe a roller - track component (where the track is fixed rigidly to some other part of the mechanism) the connecting point is not on the idealized line of the track. The moreover it is just a matter of the reference system what we consider the location of the connection. Translated to the Modelica model this means the curve is fixed to `frame_a` but it does not need to go through it. An idealized point can move along the curve. For convenience the other connector `frame_b` is located at this point on the curve.

The fact that there is just one translational degree of freedom along the curve does not imply that the point can move *freely* along the curve. There my be some friction, damping or even applied forces. But in a first stage we will not consider this.

Another question is if a Point-to-Curve constraint also ought to have *rotational* degrees of freedom. In the multi-body simulation program ADAMS[1] a point-to-curve contact always has all three rotational degrees of freedom (dofs), so it only constrains 2 translational dofs. Or think of the toddlers' toy where pierced pellets are beaded to a rigid wire. Here the pellets can rotate around the center axis of their hole, which is—again disregarding backlash—the tangential axis to the curve in the current contact point. An idealized model of this toy would be a point to curve constraint with two dofs: one translational dof along the curve and one rotational dof around the tangential axis in the contact point.

But it turns out that these additional rotational dofs can always be modeled—once tangential orientation along the curve can be represented—without putting them into the PtCv model itself: An ADAMS-like 4-dof PtCv can be built up from a PtCv without any rotational dof and a spherical joint connected to it. The mentioned toy can be modeled by a PtCv with tangential orientation with a revolute joint connected to it. Therefore here we will abstain from further complicating the models and will only consider PtCvs *with-*

DOI: 10.3384/ecp09430047

out any rotational dof: A PtCv constraint always has *just one dof*. (The idea of an additional rotational dof is only picked up for the special case of a straight curve in the section by-products where a cylindrical joint is described.)

The curve of an ideal PtCv constraint may be infinite or cyclic but it cannot be finite, i.e. cannot have end points: Such end points cannot be described by the notion of degrees of freedom. In contrast to this in technical realizations of PtCv elements we often have end points limiting the curve. But in most cases the stop is realized by means of introducing a repelling force and we can always describe a PtCv with stops as an ideal PtCv (with an unlimited curve) with additional force elements applying a repelling force preventing `frame_b` from leaving an admissible range of the curve too far. This will be described in some more detail in section PtCv with Stops.

As the Prismatic Joint from the MSL MultiBody package can be seen as the most simple case of a PtCv Joint, it is useful to have a look at it.

2 The Prismatic Joint

The Prismatic Joint has one translation dof in the direction specified by the direction parameter n: `frame_b` can move along a straight line trough `frame_a` in direction n, i.e the set $\{es \mid s \in R^3\}$ where $e = n/\|n\|_2$ is the normalized direction vector. As n is expressed in `frame_a` coordinates also the straight line is. (Here the roles of `frame_a` and `frame_b` are interchangeable, but we already view the straight line as fixed in `frame_a`.) Seen in this view the actual value of the position value s determines the "contact point" C in `frame_a` coordinates simply by $C(s) = es$. Here s is a one dimensional position variable (a distance, but may also become negative).

The only reasonable choice here is that both frames have the same orientation. The global positions of the frames is described by the equation $r_b = r_a + T^{-1}es$ where T is the orientation matrix of `frame_a` and T^{-1} is its inverse, which is simply the matrix transposed, because orientation matrices are symmetric. Disregarding the offset parameter `s_offset` we get `frame_a` = `frame_b` iff $s = 0$.

The sum of the forces acting on both frames is zero: $F_a + F_b = 0$. Regarding the torques at the frames we have $\tau_a + \tau_b + es \times F_b = 0$.

The tangential force, i.e. the force in direction n is $f = eF_a = -eF_b$. As there are no frictional, damping or applied forces in the basic Prismatic Joint model the tangential force is zero, i.e. $eF_a = 0$.

The question arising now is: What happens to these location and force balance equations when the straight line is distorted?

3 PtCv with parallel Orientation

Next we substitute the straight line by an arbitrary smooth curve, but keep the fact that both frames maintain parallel oriented permanently, i.e. the equation that both frame have the same rotation object. The curve is fixed in `frame_a` but does not need to run through it. Here we are not yet concerned with the concrete modeling of the curve. It is just a smooth mapping $C : R \to R^3$. What means "smooth" here will be elaborated later. It is not required that the curve is parameterized by it arc length. So we now use a variable s_0 for parameterizing the curve. $C(s_0)$ is the current contact point on the curve (in `frame_a` coordinates). The distance to $C(s_0)$ from the initial contact point along the curve, i.e. the arc length is denoted by the variable s. The same is with the velocities: v_0 is the curve parametrization velocity (i.e. the derivative of s_0) and v is the velocity along the curve. Note that s_0 and v_0 are not a physical length and velocity but just abstract `Real` variables. Only if the curve is parameterized by its arc length $s = s_0$ holds. (Another case where s_0 and v_0 are length and velocity is when the curve is parameterized with one of its components, say x, i.e. when $C(s_0) = \{s_0, C_y(s_0), C_z(s_0)\}$ holds, where C_y and C_z are the projections of the contact point function on y and z axis respectively. This is called a—lacking a better name—a "linear" curve in the PtCv package, because there is some main path in the curve, y and z can be viewed as deviations from this path.)

In the case of an arbitrary curve the equation $r_b = r_a + T^{-1}C(s)$ relates the global positions of the frames. Due to the parallel orientation of the frames their rotation object are the same. As the force at a frame is expressed in the frame coordinates we still have $F_a + F_b = 0$. The balance equation for the torques now becomes $\tau_a + \tau_b + C(s) \times F_b = 0$.

The property that the force along the axis of motion at the Prismatic Joint is zero here becomes that there is no force in the tangential direction along the curve in the contact point: $t_a F_a = 0$ where t_a is the (normalized) tangential vector in the contact point in `frame_a` coordinates.

A PtCv model with these equations is already suitable for building up an ADAMS-style PtCv by just

connecting a Spherical Joint to `frame_b`.

4 Orientation of the second Frame

When something is moving along a curve it is quite natural that also the orientation of that object changes while traveling along the curve: A cornering car is normally oriented in the current direction of travel. When we want an object to follow a given curve we could use the old ADAMS users' trick of connecting the object to the curve by means of two PtCvs like described above in a short distance. But this has the disadvantage, that the object only follows the curve approximately. The closer the distance between both PtCvs, the better is the approximation but the more likely are numerical problems. The moveover when we do not need the remaining rotational dof along the axis connecting both PtCvs, we have to get rid of it by means of an additional joint. (In ADAMS this is done by a Perpendicular Joint which is not yet available in the MSL.) All these joints result in a model with many equations to describe such a simple mechanism.

So it seems desirable to have a tangential orientation of the second frame directly in the PtCv model. But here the problem arises that "tangential to the curve" only determines *one* direction vector of the orientation. Even if `frame_b` of the PtCv is to be connected to a revolute joint in order to introduce a rotational dof around the tangential axis (and therefore the second direction vector of the orientation does not matter) it has to be determined in order to have a unique solution of the equations. There are several opportunities: Take an arbitrary direction vector, e.g. the local z-axis of the `frame_a` coordinate system. But this does not work when the tangential vector t_a gets too near to the selected second direction vector. The other opportunity is to take the direction of the *normal* vector to the curve (pointing inward to a local curvature) But if the curve locally becomes a straight line the normal vector is not defined and special considerations have to be taken. The moreover using the normal vector to the curve demands a higher differentiability of the curve. Fore these reasons both options are available in the PtCv implementation and can be selected due to situation by parameter.

5 PtCv with tangential Orientation

A tangential orientation of `frame_b` keeps the position equation of the frames unchanged.

In order to archive a tangential orientation at least the tangential vector to the curve in the current contact point t_a (in `frame_a` coordinates) has to be determined. For a *moving* contact point this can in principle be done in Modelica by just applying the `der()` operator. Problems arise when the contact point is at rest, especially at simulation start. (So to speak you have to know where the road is going without walking.) Currently spatial derivations cannot be directly expressed in Modelica. Therefore not only an equation for the contact point depending on the curve parametrization variable s_0 has to be provided but also an equation for the tangential vector. The moreover, when the normal vector to the curve is taken as the second direction vector for defining the orientation of `frame_b`, also the second derivation of the contact point function has to be provided. (The normal vector can be determined from this second (spatial) derivation easily.)

When t_a and n_a are normalized vectors also the binormal vector $b_n = t_a \times n_a$ is and $T = \{t_a, n_a, b_a\}$ constitutes the transformation matrix of the relative rotation object from `frame_a` to `frame_b`.

The force and torque balance equations now have to account for the fact that both frames are no longer equally orientated. Forces and torques are transformed my means of the `resolve1` and `resolve2` functions from the `Modelica.Mechanics.Multibody.Frames` package. (Depending on whether `frame_a` or `frame_b` is closer to a root in the connection tree the balance equations are expressed resolved for both frames separately in order to improve numerical performance.)

An important observation is that the property that there are no forces in tangential direction *does no longer hold:* Even if there are no friction, damping or applied forces there is a force acting in tangential direction on a body attached at its center of mass to the curve when the curvature of the curve changes. When the body (with not only mass but also inertia) starts entering a corner the rotational energy rises. Due to the law of energy preservation the translational energy has to be decreased for the same amount. This means nothing but there is a breaking force acting along the curve. (When the curvature gets less again also the angular velocity and rotational energy go down again, so the translational energy rises and there is an accelerating force. So the process is reversible: after leaving the corner the travel velocity along the curve is the same as before entering the curve.)

When the curve is a circle there is a direct correspondence between rotational and translational energy.

For any two times differentiable curve there is a unique circle approximating the curve in the best way locally in a given point. This circle is called the *osculating circle*. Its radius is the inverse of the length of the normal vector. So the rotational velocity of a point connected to the curve is the same as the one of a point on this circle. Thus the equation relating translational and rotational energy can be applied to any two times differentiable curve and by differentiating this equation an equation for the tangential force can be applied.

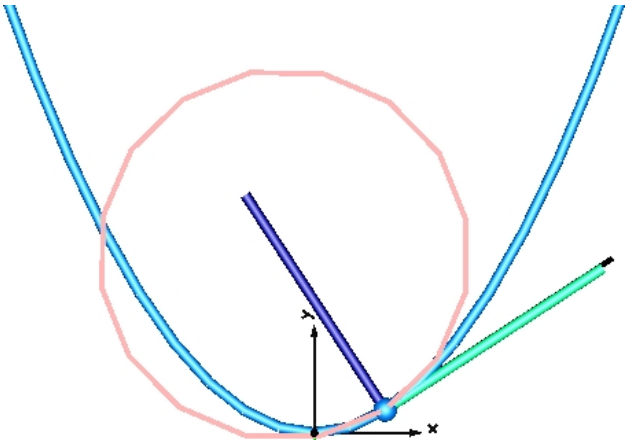

Figure 1: Planar curve (parabola) with osculating circle, tangential and normal vector in the contact point

The osculating circle can be visualized in the PtCv model. For the force equations we do not need its center coordinates and not even its radius but only its inverse, the *curvature* of the curve. This difference is important when the curve becomes a straight line locally, so that the radius of the osculating circle gets infinite and is not defined. The curvature simply gets zero and does not provide a problem.

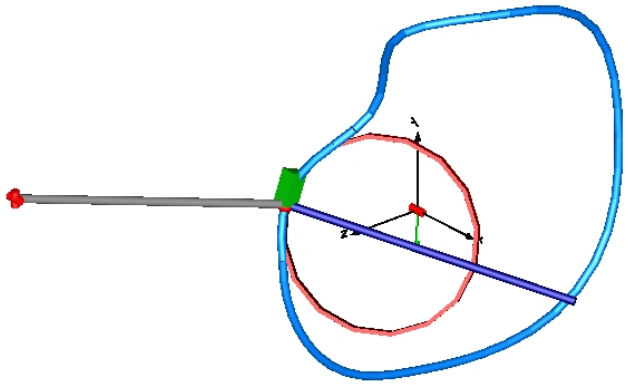

Figure 2: Osculating circle and normal vector at a 3D curve

Depending on the situation and the relation of tangential and rotational velocity this tangential force along the curve due to changing curvature can be neglected, e.g. when describing the cornering of an ICE train. On the other hand simulations of a body with a rather large inertia connected to a curve with parabola shape under the influence of gravity showed that the translational velocity is not highest in the lowest point of the parabola (as one might have expected) but at a symmetrical pair of points in a certain distance with a local minimum of the velocity in the lowest point of the parabola—where the curvature is highest.

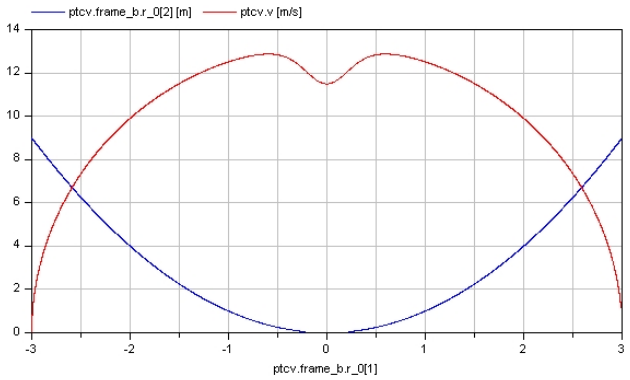

Figure 3: Velocity of a body with high inertia sliding freely along parabola

6 Geometry Definition

Up to now we only talked abstractly about the current contact point $C(s_0)$, the tangential vector t_a and the normal vector n_a (which are derivations of the contact point functions or are determined from derivations). In principle it is always possible to define the contact point function by its three cartesian components in the `frame_a` coordinate system. But it is rather inconvenient to define, say a helix curve with its center line in direction n by directly providing the three curve parametrization functions C_1, C_2, C_3 defining the curve in `frame_a` coordinates.

Therefore similar to the direction vector n in the Prismatic Joint two direction vector parameters n_x and n_y have been introduced defining a local x, y, z-system (with the z-direction orthogonal to both n_x and n_y) for a more convenient definition of the curve. E.g. a straight line in a direction n can be defined by setting $n_x = n$, taking for n_y any direction not parallel to n and the first coordinate function is the identity mapping, both other components are zero mappings.

The moreover there are model variants, where the curve is not defined in cartesian x, y, z coordinates but in cylinder coordinates: n_x here determines the axial

direction of the cylinder. Here you have to provide the radial and axial component of the curve. To define a PtCv with the mentioned helix curve take $n_x = n$, the radial component is a constant and the axial component a linear function with suitable slope.

Now remains the question how the cartesian or cylindrical components of curve definitions are described.

7 Geometry Component Definition

There are several methods of defining the $x-$, $y-$ and $z-$ or radial and axial component of a curve definition. There is the opportunity to use *replaceable functions*. You write a function returning a single `Real` defining the desired function component. This method is good for defining geometric curves like the helix mentioned above, but is not applicable for defining curves with an arbitrary given shape. The moreover currently it is required not only to implement the functions describing the curve component but also their first and second derivations (because we need the spatial derivations of the curve to calculate the tangential and normal vector). Therefore this method is rather of theoretical interest to investigate the properties of an analytical function of concern.

As apparently there was no package ready to use we decided to implement our own cubic spline interpolation package. (It is not part of the PtCv package, because splines are of course applicable in many areas different from PtCv modeling.) Natural cubic splines are implemented, i.e the second derivation at the definition range borders are zero. A spline may have an arbitrary number of definition points which need not be equally spaced. Extrapolation is possible as constant or linear continuation or as periodic extrapolation with a repetition of the definition range infinitely many times. Although the complete calculation of the spline evaluation (and of the calculation of the second derivations at the definition points) is completely implemented in Modelica (like in the PtCv package, no external functions are used) Dymola is not able to calculate the derivations of the interpolation and extrapolation functions itself, so the derivations had to be provided explicitly. At a PtCv with parallel orientation using the normal vector for calculating the orientation object time derivations up to the 4^{th} order and spatial derivations of the evaluation function are needed. So a great deal of the development of the `Spline` package was implementing derivations.

Depending of the type of PtCv splines for the $x-$, $y-$, $z-$, radial or axial component of the curve definition can be provided. The radial spline is automatically extrapolated periodically, but it is up to the user to ensure that the lower and upper definition range border have the same radius to ensure the contact curve is continuous. All other component splines are linearly extrapolated. All definition splines have suitable defaults: The axial spline defaults to the zero spline, the radial spline defaults to the unit radius, so the default curve for a circular PtCv is a circle in the local $y-z$-plane (orthogonal to the provided n_x direction vector). The x-spline defaults to the identity mapping, $y-$ and z-spline to the zero mapping. By this means in many cases not all the definition splines have to be provided.

After we implemented our own `Spline` package we discovered that there is already a package for evaluation of Bezier Splines[3] developed at the DLR, Oberpfaffenhofen in 2002. (It is available under the Modelica license.) In order to use this `BSpline` package we had to write extrapolation features for it. (They were not added to the `BSpline` package, which was kept unchanged, but were placed into our `PtCv` package.) Now also PtCvs using BSplines for the curve definition are available in this package.

8 The PtCv Model Family

The sections above already mentioned that there are several PtCv models with different coordinate systems (cartesian or cylindrical) and different types the curve component functions are defined (replaceable functions, cubic splines, BSplines). All these models are extended from one basic model in several steps.

The partial model `PartialPtCv` contains everything common to all PtCv models. These are all parameters which are not directly related to the curve geometry definition, most of the parameters concerning animation, the equations relating the position and orientation of the two frames, the force balance equations and the equations for the force tangential to the curve. This model mostly uses the cartesian x,y,z-coordinate system mentioned above. The curve parametrization variables (e.g. s_0 are defined using replaceable types defaulting to `Real`, so they can be redeclared in situations, where they really mean positions and velocities, or angles and angular velocities. What is missing here is the equations for the current contact point $C(s_0)$, the current tangential t_a and normal vector n_a.

The next extension step is optional and rather intended for development and debugging purposes: The partial model `PartialPtCvExtended`, extended from

PartialPtCv contains variables (and visualizers) not really needed for the PtCv contact calculation but providing useful additional information, like visualizers for the osculating circle, the tangential and the normal vector, the traveled distance along the curve s and variables calculating the potential, translational and rotational energy for the special case that a mass with its center of mass is connected to a curve fixed in space.

The partial model PartialCircularPtCv is intended for all PtCv models using cylinder coordinates. (This does not mean that the curve itself is circular, like seen in the helix curve example. Therefore the name may be changed in future versions.) This model contains variables for transforming the cylinder coordinates into the cartesian x,y,z-system of the base model. There is a parameter revolutionLength determining the length of one revolution, i.e. if revolutionLength $= 360$ the radial and axial component are scaled on degrees, if revolutionLength $= 2\pi$ they have to be defined in radiant. The curve parametrization variables are redeclared to angles, angular velocities and angular accelerations.

Remember in the PtCv package the term "linear" means that the cartesian x-component of the curve is the identity mapping, i.e. there are only possible deviations into the y- and z-direction. For this case there is the partial model PartialLinearPtCv. In this case the curve parametrization variable s_0 is a distance, not along the curve but along the x-axis instead. Therefore the curve parametrization types are redeclared to SI.Position, SI.Velocity and SI.Acceleration. As also linear PtCvs use cartesian coordinates no transformation is required.

The partial model PartialGeneralPtCv only redeclares all curve parametrization types to Real, just to prevent further redeclaration. All these three partial models are currently extended from PartialPtCvExtended in order to have the additional debugging information at hand. In a final release they may be directly extended from PartialPtCv skipping the extra variables and visualizers.

The next step in this extension hierarchy are the completed (non partial) PtCv models, extending from one of the three models PartialCircularPtCv, PartialLinearPtCv or PartialGeneralPtCv. Here only the parameters for defining the curve components are declared (i.e. the replaceable functions for the geometry definition components currently together with their derivatives or the spline or BSpline parameters) and also the equations for determining the current contact point $C(s_0)$, the tangential vector t_a and the normal vector n_a, by evaluation of the functions or the spline resp. BSpline functions.

This separation into several model layers makes it easy to add new PtCv models with a custom geometry. There is even an instruction how to do so in the package documentation. On the other hand it enables to protect the base models by encryption in a version to be released in future.

9 PtCvs with Stops

So far we only dealt with PtCvs with an unlimited curve. For building a PtCv with a limited admissible range of the parametrization variable s_0, we take an existing full PtCv model and add the stops by extending it. (So we go one step further in the model extension hierarchy.) The implementation of a PtCv with stops has been performed exemplarily on a Linear PtCv where the curve is defined by two replaceable functions in y- and z-direction, but it can be implemented in the same way for any type of PtCv model.

It is important to note that adding stops to a PtCv does *not impose a new constraint* to it, but only applies an new additional force in tangential direction depending on the position and velocity of the contact point. If frame_b is forced to proceed by some prescribed motion it will do, regardless of the repelling forces. They may become huge, but as we have ideal elements nothing will break the mechanism like it will happen in a physical realization. A PtCv with stops still has one translational dof.

The stop position is defined by two new curve parametrization parameters $stop_1$ and $stop_2$. In this way the stop position is automatically located on the curve, namely at the positions $C(stop_1)$ and $C(stop_2)$. In case $stop_1 < s_0 < stop_2$ there is no additional force in tangential direction.

The stop is established by applying strong repelling forces to the point frame when it leaves the admissible parameter range. The repelling force consists of a non-linear spring force and linear damping where the damping coefficient is dependent on the actual penetration: If $s_0 < stop_1$ holds, there is contact to the lower stop and there is a force like the IMPACT force defined in ADAMS with a spring and a damping ingredient:

$$F(x) = \begin{cases} max(k(x_1 - x)^{exp} - cv, 0) & x < x_1 \\ 0 & else \end{cases}$$

where k is the spring stiffness, exp the stiffness exponent and $c = STEP(x, x_1 - d, c_{max}, x_1, 0)$ is the current

damping coefficient depending on the position, which returns c_{max}, when $x < x_1 - d$, but zero, when $x > x_1$ and is smooth in between. This means that the damping force does not directly apply fully at the moment of contact, but is increased from contact to a penetration d where full damping c_{max} is archived. The `IMPACT` and `STEP` functions are implemented as separate functions to be used in other contexts than the PtCv stops.

The stops are visualized by cylinders with the origins at $C(stop_1)$ and $C(stop_2)$ pointing into the directions $-t_{stop_1}$ and t_{stop_2} (out of the admissible parametrization range) where t_{stop_1} and t_{stop_2} are the tangential vectors to the curve at $C(stop_1)$ and $C(stop_2)$. When the contact point leaves the range $stop_1 < s_0 < stop_2$ and there are contact forces, the relevant stop visualization cylinder changes its color.

Subject to further work on this topic is to establish a new partial model containing the stop implementation. So a particular PtCv model with stops ought to be little more than an extension of both the corresponding normal PtCv model (without stops) and the stop model.

9.1 Curve-to-Curve (CvCv) Constraints

A Curve-to-Curve (CvCv) constraint between two curves is defined by the property that at any time both curves have a common contact point and both curves are oriented tangentially to each other. This is a local condition. It does not require that the total shape of the curve would admit this constellation when physically built. (The curves may cross each other at regions away from the contact point.)

Some multi body dynamics programs (like ADAMS) provide Curve-to-Curve constraints only for *planar* curves and there is already a CvCv constraint implementation for planar curves in the `PlanarMultiBody` package [2]. But the concept of Curve-to-Curve constraint can also be transferred to smooth curves in 3D space.

In 2D CvCv constraint modeling when the contact point is found the position of both curves to each other is determined: A 2D CvCv constraint has just one (translational) dof. Her in 3D it is plausible to admit a rotational dof around the common tangential axis of both curves also.

Instead of modeling a CvCv constraint elementarily by stating its position and force balance equations we follow the approach of using two PtCv constraints connected to each other with their "point sides" to each other with a revolute joint in between. In case no rotational dof is admitted, there is a fixed rotation component instead where is can be set if both curves are to be oriented opposite to each other by using a rotation angle of 180 degrees or not.

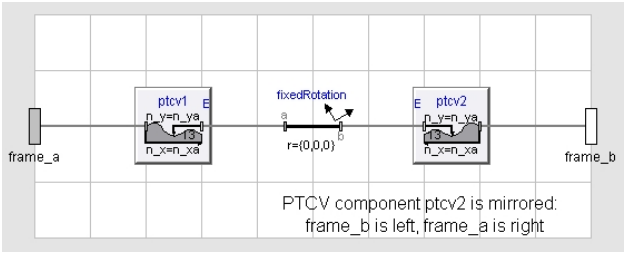

Figure 4: Diagram layer of a CvCv constraint

10 Point-to-Surface Constraints

It is quite natural to transfer the notion of a Point-to-Curve constraint to 2D: At a Point-to-Surface (PtSf) constraint a point can move along a smooth surface. This means a PtSf constraint has 2 dofs. For surfaces constituting analytical functions there are PtSf models with the surface described by replaceable functions. For practical applications there are PtSf versions where the surface is defined by 2D-Splines from the `AreaSpline` package. Currently only the option that the point frame is oriented parallel to the first frame is implemented, but in future orientation tangential to the surface will be an alternative.

Like at the PtCv models for all PtSf models there is one common partial base model and extensions with coordinate transformations for cylindrical and spherical coordinate systems (besides the core Cartesian coordinate system) from which the specific PtSf models are extended.

10.1 The AreaSpline Package

Here again it is straight forward to try to transfer cubic spline interpolation to 2D, i.e. to return the z-coordinate for a given (x,y) location. The spline is to be defined on a rectangular grid $(x_i, y_j)_{i=1,...m, j=1,...n}$. (This can be view as a landscape, where the height is tabulated at the points of this grid. Interpolation is the task to calculate the height $z = h(x,y)$ at any point (x,y) in between, under the assumption that the landscape is smooth.

10.1.1 Area Spline Interpolation

The idea here is to interpolate in the x- and y-direction rather independently. For any given x_0 location the projection $f(y) = h(x_0, y)$ can be considered a usual

cubic spline (in the y-coordinate). In case x_0 is one of the grid values $(x_1, ...x_m)$ we can determine the second derivations at these positions by solving the equations system like in the one-dimensional case. So we can even determine $h(x_0, y)$. (The same holds when the y-coordinate y_0 is one of $(y_1, ...y_n)$: We can calculate $h(x, y_0)$.) To calculate the height at an arbitrary position (x, y) not matching any of the grid lines we can first determine the definition rectangle to which (x, y) belongs, i.e. indices i and j so that $x_i < x < x_{i+1}$ and $y_j < y < y_{j+1}$. Now we can calculate both $h(x_i, y)$ and $h(x_{i+1}, y)$. The moreover we can determine $h(x_i, y)$ for all $i = 1, ...m$ and consider these values as the definition points of a cubic spline in the x-coordinate. The problem is that for calculating this spline we'd had to solve the system of equations for this particular arbitrary y, i.e. at evaluation time, what would be time consuming at larger definition grids. But as the interpolation function of a spline is a 3rd order polynome between definition points, the 2nd derivation is a 1st order polynome which can be linearly interpolated easily.

The approach is now as follows: Instead of calculating the 2nd derivations of the spline defined by $(x_i, h(x_i, y_0))_{i=1,..m}$ by solving a system of equations, we take the coefficients of the splines $(x_i, h(x_i, y_j))_{i=1,...m}$ and $(x_i, h(x_i, y_{i+1}))_{i=1,...m}$, interpolate each pair linearly and take them as the coefficients of the spline through $(x_i, h(x_i, y))_{i=1,...m}$. By interpolating this spline at x we can finally calculate $h(x, y)$.

Here we started by working in y-direction, i.e. by first calculating $h(x_i, y)$ and $h(x_{i+1}, y)$ but that is not crucial. It can be shown that we end up at the same result, when we calculate $h(x, y_j)$ and $h(x, y_{j+1})$ first and determine the coefficients of the spline trough $(y_j, h(x, y_j))_{j=1,...n}$ by linear interpolation of the coefficients in the columns x_i and x_{i+1}.

This approach has two advantages:

1. All spline coefficients can be calculated once for all when defining the spline (or when modifying it at an event). No solving of systems of linear equations is required at evaluation time.

2. The coefficients in x-and in y-direction can be calculated independently. The moreover the coefficients in each row and column can be determined independently. We simply can calculate the usual coefficients of 1D splines along all the definition grid lines. With an $m \times n$ definition grid we have just m systems of equations of size n and n systems of size m instead of one or two big systems of size mn or so.

Unfortunately this approach has one big disadvantage: Although it is true, that the 2nd derivation of the interpolation function between two definition points is a first order polynome which can be linearly interpolated without any loss of precision, we just get an approximation, when we interpolate between the 2nd derivations in y-directions at (x_i, y_j) and (x_i, y_{j+1}) in order to get the value at (x_i, y). Linear interpolations is exact here only in x-direction between (x_i, y_j) and (x_{i+1}, y_j) but not in y-direction.

As a result of this inexactness we have the following effects. The interpolation function is:

1. continuous,

2. two times continuously differentiable in any point not matching one of the definitions grid lines,

3. two times partially continuously differentiable along the definition line grids, but in general

4. not partially differentiable when crossing the definition line grids, i.e. not (totally) differentiable at points on the grid lines.

So the resulting interpolation surface looks folded at the definition grid lines. The distances of the definition lines are the smaller the closer the definition grid lines get. Of course we are a lot better off than with interpolating the definition grid just linearly.

10.1.2 Area Spline Implementation

Like at the 1D splines there is a function `makeAreaSpline` to initialize a spline record by calculating the spline coefficients like described above. In the evaluation function `evalAreaSpline` extrapolation is possible constantly, linearly and periodically. It can be chosen between these three options independently in the x- and the y-direction.

Unfortunately we had to implement the 1st and 2nd derivation of this function manually. (Higher order derivatives were not yet needed because at the PtSf constraints up to now only parallel orientation of the point frame was implemented.) But on the other hand having these time derivatives it was easy to implement the partial derivatives into x- and y-directions that we also needed anyway.

10.1.3 Area Spline Visualization

There is a sub-package `Visualizers` for displaying area spline surfaces using the `Plot3D` package. An area spline can be displayed by entering its definition

data into a function call. There are variants also displaying normal vectors to the surface or one of the partial derivatives instead of the spline surface itself. The definition grid lines are displayed also like in the example below.

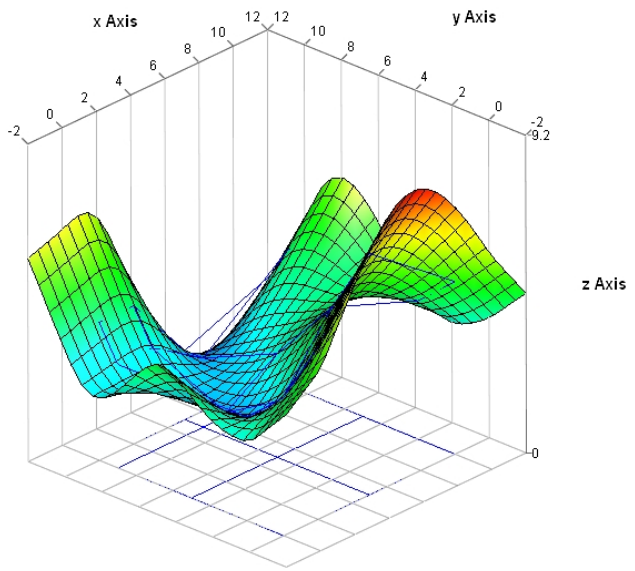

Figure 5: Example of Surface definition with `AreaSpline` Package

Unfortunately this package cannot be used for visualizing a surface in a mytt MultiBody model with the animation used there.

11 By-Products

In the development of the PtCv package an attempt was made to integrate a revolute dof to a PtCv with tangential orientation. This did not yet work properly. But it does work in the special case that the curve is simply a straight line. So there is a translational dof along an axis and a rotational dof around that axis. This is noting but a cylindrical joint. Therefore this was turned to a separate model where the arbitrary curve with all its parameters was reduced to a direction vector n and the hierarchy of partial models was turned into one model.

Of course there is a Cylindrical Joint model in the MultiBody package of the MSL, but this is composed of the connection of a Prismatic and a Revolute Joint. But compared to this standard implementation the Cylindrical Joint model in this package is described directly by equations. The number of equations is about 10% less than in the standard implementation and simple test models are considerably faster.

A PtSf constraint where the surface is a plane is a planar parallel joint, i. e. `frame_b` can move along a plane through `frame_a` defined by two direction vectors n and m. This is like the planar joint in the MultiBody library, but without the rotational degree of freedom. As such a joint is of general interest it has been implemented as a separate model. Despite the MultiBody planar joint here the translation in the plane is not modeled by two orthogonal prismatic joints but elementarily.

12 Conclusions

Although especially the PtCv models are up and running the packages described in this paper are to be seen as a work in progress. It seems valuable to incorporate some more ideas from the `PlanarMultiBody` package like providing the user with a collection of predefined curves like circles and elipsoids. Cubic splines will be then just one type of predefined curve. This applies even more to the `PtSf` package which will become much more usable if there would be a set of predefined parametrizised shapes.

The original plan to develop also contact *force* elements besides the constraints will not be addressed anymore within the Eurosyslib project due to lack of time but are subject to further work.

13 Acknowledgements

The `PtCv` (Point-to-Curve) and the `PtSf` (Point-to-Surface) packages and the used geometry packages `Spline` and `AreaSpline` have been developed as part of the ITEA2 **Eurosyslib** project (WP 8.6).

References

[1] http://www.mscsoftware.com/products/adams.cfm

[2] M. Höbinger, M. Otter: PlanarMultiBody - a Modelica Library for Planar Multi-Body Systems. Proceedings of the 5th Modelica Conference, Bielefeld, Germany, 2008, pp. 549-556

[3] Schillhuber, Gerhard, `BSpline` package, Copyright Modelica Association and DLR, 2002.

[4] http://www.itea2.org/public/project_leaflets/EUROSYSLIB_profile_oct-07.pdf

