Flexible Tubing Function

Jennifer Gass

November 30, 2012

1 Problem Definition

1.1 Introduction

The current plant design requires the use of flexible tubing to join certain sections. This type of plumbing is needed to connect the end of the CDC to the dosing in the rapid mix pipe, the end of the stock tank plumbing to the constant head tank, the bottom of the constant head tank to the new CDC manifold system, and more. However, the design tool does not depict these flexible tubes in the CAD drawings, so I will create a function that automates the drawing of the flexible tubing in the plant where it is required. Figure 1 shows the current CDC drawing which does not display any actual flexible tubing. It's important for us to convey to our users how the CDC is incorporated into the plant by drawing flexible tubing and provide a simple depiction of the connection.

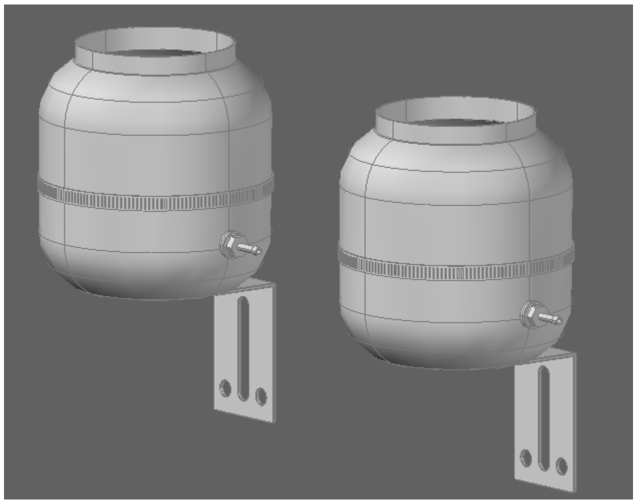


Figure 1: Current CDC drawing within the Design Tool

1.2 Design Details

The Fall 2011 design team worked on creating a flexible tubing function FlexTubeF which draws out a pipe between two points, given a path defined by an array of points. Although their functions draw a proper flexible tube, there are still two steps that need to be taken in order to integrate it into the overall design tool. First, I have to find a way to calculate the points required in the path of the tube, based on the constraints of the design. Second, I'll need to generalize the function enough (or, break into separate functions) so that any two inlet and outlet orientations can have flexible piping implemented.

FlexTubeF (located in PlumbingF), which calls on FlexCylinderF (located in MtoA Translators), creates a spline based on the point array given, and the sweeps a circle across this spline to form the pipe. It doesn't orient the ends of the cylinder in any sort of direction, so first I'll need to determine a method of orienting the spline to reflect the nature of the flexible tube being held in a rigid pipe at the ends.

I'll use a parabolic function to model the path of a hanging tube fixed at two ends, from which I'll easily be able to extract the points for the array. These path points will comprise an array that can be used in FlexTubeF. This function will run differently based on what the orientation of the two ends of flex pipe are–both ends vertical, both ends horizontal, or one end vertical and one horizontal. 2, 3, and 4 show a few of the different orientations that need to be considered when designing the flexible pipe function. ParPointF, the function I'm creating which calculates the path points, will take in the locations of the end points, the tube length, and the orientation of inlet and outlet of the pipe.

Figure 2: Both ends of tube horizontal

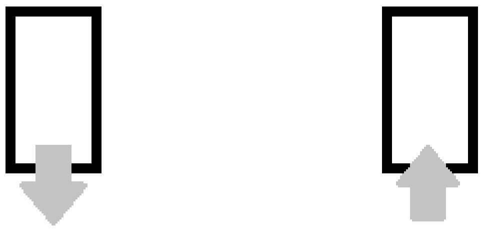


Figure 3: Both ends of tube vertical

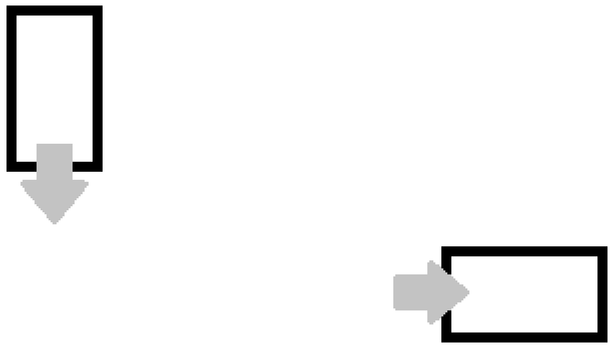


Figure 4: One end of the tube horizontal, one vertical

2 Documented Progress

2.1 Catenary Solution

Originally I had planned on modeling the flexible tubing with a catenary function. For the catenary function, the constants [u, v, β] would be calculated using a system of 3 equations solved simultaneously.[1]

$x(t) = x_{in} + \frac{t\cdot(x_{out}-x_{in})}{n}$

$y(t) = m\left(x_{in} + \frac{t\cdot(x_{out}-x_{in})}{n}\right) + b$

$z(t) = u \cdot cosh(\frac{x(t)-v}{u}) + \beta$

Plug x(t) into z(t):

$z(t) = u \cdot cosh(\frac{n\cdot x_{in}-n\cdot v+t\cdot(x_{out}-x_{in})}{n\cdot u}) + \beta$

[1]http://www.infogoaround.org/JBook/Catenary.pdf

We also know the inlet and outlet points for z (z$_{in}$ and z$_{out}$) to acquire two equations in terms of the 3 unknown variables:

$z_{in} = u \cdot cosh(\frac{x_{in}-v}{u}) + \beta$

$z_{out} = u \cdot cosh(\frac{x_{out}-v}{u}) + \beta$

Since L as a pre-defined constant we can get one more equations from this. The equation for arc length of a three dimensional equation is given by the following, where t/n is a predetermined arbitrary step size (so, t acts as an index in these equations)[2]:

$L = \int \sqrt{[x'(t)]^2 + [y'(t)]^2 + [z'(t)]^2} dt$

The integral would be evaluated from 0 to however many divisions, n, are chosen. This will be an arbitrary number; a higher n will give a curve with more defined points. While Mathcad can calculate the set of points for a low n, I have found that AutoCAD produces some window errors when I change n, which may be an issue with the spline function within FlexCylinderF.

The three equations are now:

[1]$z_{in} = u \cdot cosh(\frac{x_{in}-v}{u}) + \beta$

[2] $z_{out} = u \cdot cosh(\frac{x_{out}-v}{u}) + \beta$

[3] $L = \int \sqrt{\left[\frac{(x_{out}-x_{in})}{n}\right]^2 + \left[\frac{m\cdot(x_{out}-x_{in})}{n}\right]^2 + [sinh(\frac{n\cdot x_{in}-n\cdot v+t\cdot(x_{out}-x_{in})}{n\cdot u})]^2 dt}$

From here, the identities of the three constants could ideally be determined using a Mathcad solve block. However, this point is where Mathcad has issues solving the integral of the square root part of the function, and indicates that the block is unsolvable. I have tried rewriting sinh(t) as $\frac{e^t-e^{-t}}{2}$, but this did not help. Eliminating the square root operator results in a solution, indicating that my indexing and calculations are probably correct, so I will consider a different way of approaching this problem.

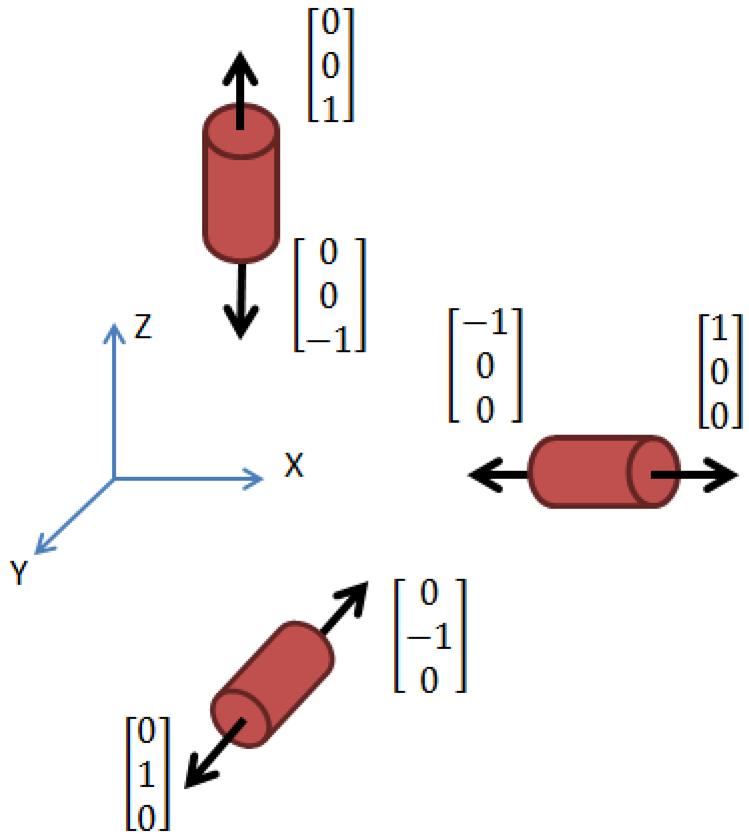
I found that the path points in a parabola given the inputs are more solvable in Mathcad than are the points in a catenary. I looked into using an equation for a parabola instead of a catenary (i.e. $y = ax^2 + bx + c$), which may not model the curve as accurately, but is a much simpler calculation that Mathcad is definitely able to perform. The parabolic equation will still accurately depict the flexible tubing in the design tool.

2.2 ParPointF

Inputs and Theory The five fixed inputs for ParPointF are the inlet coordinates, the outlet coordinates, orientation of inlet, orientation of outlet, tubing length, and the number of divisions which will be iterated through in the calculation of the parabolic points. The inlet and outlet coordinates should correspond to the points in the center of the end of each of the fittings. The orientations can be determined by referencing Figure 5, in the form:

$[inlet \quad outlet]$.

[2]http://members.chello.nl/j.beentjes3/Ruud/catfiles/catenary.pdf



$\begin{bmatrix}0\\0\\1\end{bmatrix}$
$\begin{bmatrix}0\\0\\-1\end{bmatrix}$
Z
X
Y
$\begin{bmatrix}-1\\0\\0\end{bmatrix}$
$\begin{bmatrix}1\\0\\0\end{bmatrix}$
$\begin{bmatrix}0\\-1\\0\end{bmatrix}$
$\begin{bmatrix}0\\1\\0\end{bmatrix}$

Figure 5: Inlet and outlet orientation conventions

Overall, the function will estimate the curve by calculating the path of the points in one dimension (either x or y) with respect to z, then project this path onto the diagonal to make the equation less complicated. The determination of which of these dimensions will be chosen will just be based on a comparison of Δx and Δy. The parabola function will be applied to the longer side, and the other variable (x or y) will just be linear with respect to z, i.e. $y(t) = mx(t) + b$. A 'straight' looking part will be appended to each of the ends so that each will be tangent to the respective fitting.

To solve the end tangency issue, I am going to add an offset the x and y coordinates of each pair of points so that there is a small segment in each the inlet and outlet where the pipe is flat. This solution works for the case when there are two horizontal ends, and in a vertical case the ends will be tangent to the z-axis. Since ParPointF will use the orientation of the inlet and outlet as an input, it will first create the offset for the inlet and outlet, and then draw out the parabola between those offset points.

I'll use the same linear relationship between x(t) and y(t), and the same comparison of Δx and Δy to determine which variable the z(t) function will be based on. The following equations will still apply for what I'll call Case X ($|\Delta x| > |\Delta y|$) :

$x(t) = x_{in} + \frac{t \cdot (x_{out} - x_{in})}{n}$

$y(t) = m \left(x_{in} + \frac{t \cdot (x_{out} - x_{in})}{n} \right) + b$

$z(t) = A \cdot x(t)^2 + B \cdot x(t) + C$

To find m and b, I've set up a solve block with the following equations:

$y_{in} = m \cdot x_{in} + b$

$y_{out} = m \cdot x_{out} + b.$

Since the actual function needs to account for the case in which $|\Delta y| > |\Delta x|$ (Case Y), the code also includes an alternate solve block with the following equations:

$x_{in} = m \cdot y_{in} + b$

$x_{out} = m \cdot y_{out} + b.$

Going back to Case X, the next solve block will determine z(t) using the length equation and the boundary conditions at the ends of pipe:

$L = \int \sqrt{\left[\frac{(x_{out} - x_{in})}{n}\right]^2 + \left[\frac{m \cdot (x_{out} - x_{in})}{n}\right]^2 + [2A \cdot \left[x_{in} + \frac{t(x_{out} - x_{in})}{n}\right] + B]^2 dt}$

$z(t) = A \cdot x(t)^2 + B \cdot x(t) + C$

$z(t) = A \cdot x(t)^2 + B \cdot x(t) + C$

The length integral will be evaluated from 0 to n; these three equations together will determine the values for A, B, and C which will create a parabola between the end points specified. Case X uses A, B, and C as the outputs to the These variables are then used in ParPointF, which loops through values of t from 0 to n and creates a point array containing the path points of the parabola. ParPointF includes calculations for both Case X and Case Y, and chooses the appropriate case based on the comparison of Δy and Δx.

Parabola Details One issue encountered with the parabola is that it can point either up or down (think convex or concave), leading to two sets of solutions for the variables A, B, and C. The direction of the parabola is defined by whether the constant A is positive or negative, so the solve blocks will choose a solution based on the sign of that variable.

One of the primary modifications I was working on is the orientation of the ends of the tube. At first, I had the function draw a parabola between the two input points. It would then concatenate a row of points directly before and after the parabola points based on whichever orientation is specified for each the inlet and outlet. My goal was to have Autocad connect the straight parts with the parabola with the spline function to ensure a smooth inlet and outlet of the tube and make the flexible tube look rigid over the couplings. However, an issue I ran into was that the straight ends would sometimes change orientations and cause the appearance of a shifted tube (Figure 6). While some end orientations worked out fine, there was no easy way to eliminate the spiraling at the ends. I tried changing the distance and number of points within the straight portions, and changing the number of points within the parabola, but nothing seemed to fix that issue. Finally I found a spline property that would make the ends of the spline tangent to a given point. It seems like such a simple fix, but I originally overlooked changing the actual spline code, as it had been previously written by other members of the design team.

While the tangency issue is now fixed, there remains a problem where AutoCAD will occasionally not sweep a circle over a spline, for unknown reasons. To solve this, I looked into possible causes for the error. I tried drawing the circle at the other end of the spline and sweeping in the other direction, but this only fixed the error in a few cases. In the case of both the inlet and outlet being horizontal (I tested it in the x-direction), the non-specific sweep error comes up regardless of which end I sweep from.

Many forums online suggested converting the spline to a polyline, so I have done this and this manages to extrude a tube which looks identical to the tube originally swept over the spline. However, the ends of the tube turn out are not as tangent as they were in the spline form, which I would need to fix (Figure 7). I brought back my idea of appending the straight part to the end, so the final code combines the spline tangency property with an extra straight portion (Figure 8).

The final design can take a variety of forms, one of which is depicted in Figure 9.

2.3 Future Work

While the flexible tubing should work for most cases, it is not designed to handle cases in which the inlet and outlet have the same y-value. This is simply because of the equation I've used (y=mx+b). There is no solution to a line with an 'infinite' slope, so the function cannot be used on pieces of tubing which have a constant (i.e. run directly in the y-z plane). Although ideally I could make a separate solveblock case to cover for this (i.e., determine that $|\Delta y| > |\Delta x|$ and then calculate a slope of dx/dy=0 and use an equation $x(t) = m \cdot y(t) + b$), both solveblocks would need to run simultaneously and one would output an error. Unfortunately, any error will not allow the code to run, so this is something that may need to be addressed in the future.

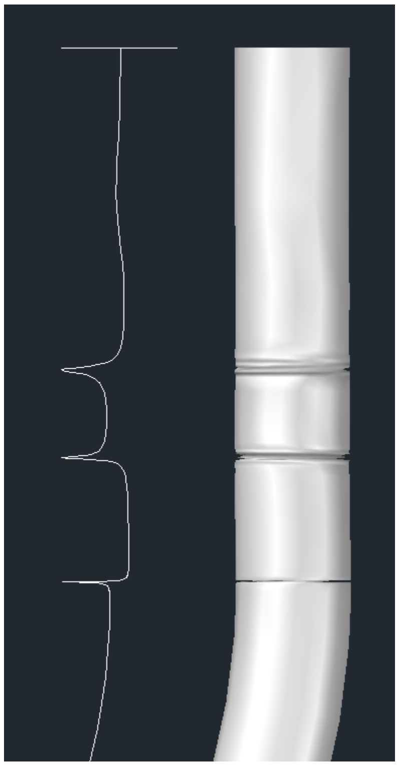


Figure 6: Spline with straight portion added

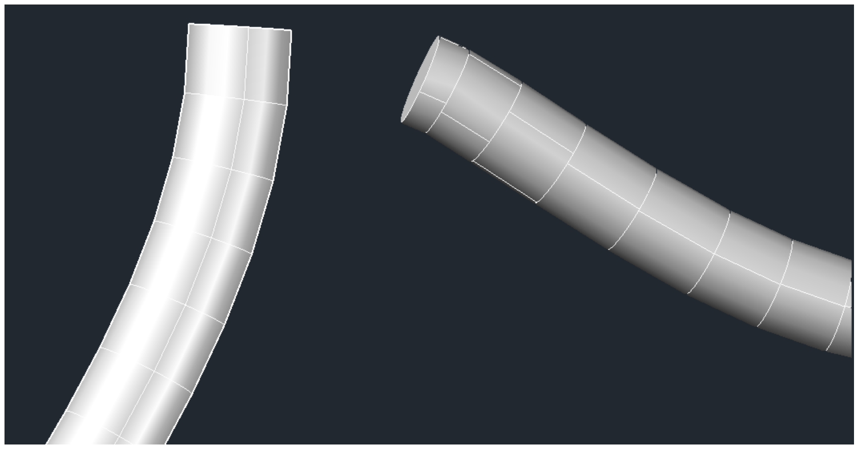


Figure 7: Spline with end tangency added

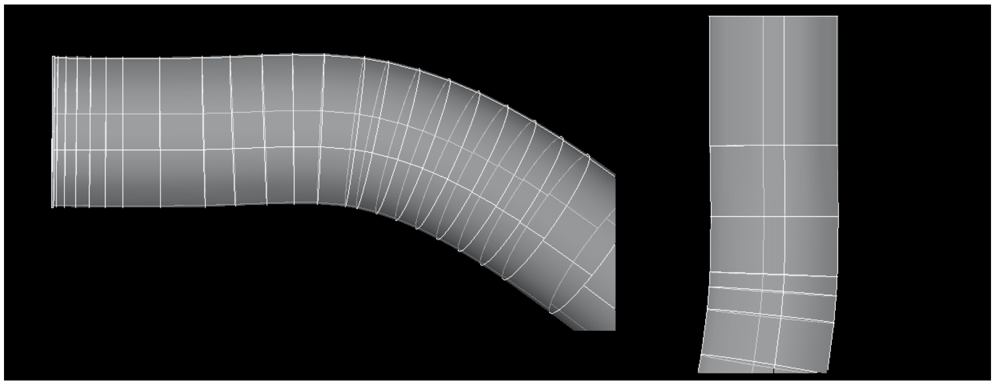


Figure 8: Spline with end tangency and straight portion added

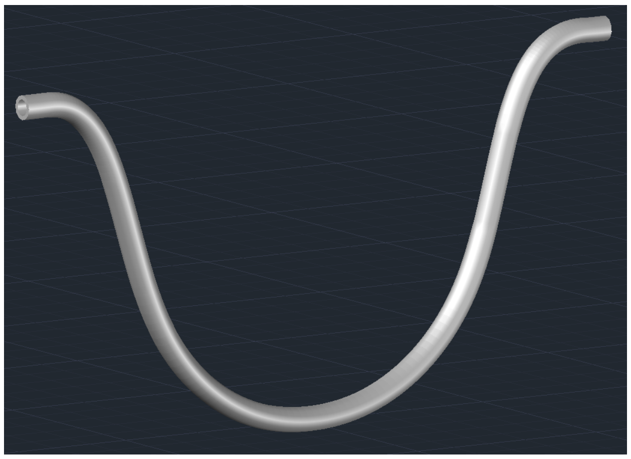


Figure 9: Example of final design