How to constrain a particle without losing your mind.

Andrew Witkin
15-860 Spring 94

We've gone through the basic constraint force derivation in class, but things can get a little messy when you try to derive the formulae for nontrivial constraints. We'll run through the whole process here for a stupid constraint, but we'll do it in a way that scales up to more complicated ones. We assume you've followed the discussion in class and gone through the previous handouts.

The first part of the process only has to be done once, because it doesn't depend on the particular constraint at all, only on the general form $C(x(t), y(t), t) = 0$. To avoid clutter, we'll use subscripts to denote differentiation, so C_x is dC/dx, C_{xy} is $d^2C/dxdy$, etc. We'll use C_t to denote C's direct derivative w.r.t. time, and $\dot{C}$ to denote the total time derivative, direct and indirect. To take a time derivative we compulsively apply the chain rule and the product rule, making sure not to forget any dependencies.

$$\dot{C} = C_x\dot{x} + C_y\dot{y} + C_t$$

and differentiating again,

$$\ddot{C} = C_{xx}\dot{x}^2 + 2C_{xt}\dot{x} + C_x\ddot{x} + C_{yy}\dot{y}^2 + 2C_{yt}\dot{y} + C_y\ddot{y} + 2C_{xy}\dot{x}\dot{y} + C_{tt}.$$

We get constraint forces into the picture by using the fact that $\ddot{x} = f_x + \lambda C_x$ and $\ddot{y} = f_y + \lambda C_y$, where $[f_x, f_y]$ is the external force, giving

$$\ddot{C} = C_{xx}\dot{x}^2 + 2C_{xt}\dot{x} + C_x(f_x + \lambda C_x) + C_{yy}\dot{y}^2 + 2C_{yt}\dot{y} + C_y(f_y + \lambda C_y) + 2C_{xy}\dot{x}\dot{y} + C_{tt}.$$

Next we solve $\ddot{C} = \alpha C + \beta\dot{C}$ for λ (where α and β are the feedback constants) getting

$$\lambda = \frac{1}{C_x^2 + C_y^2}(\alpha C + \beta\dot{C} - C_{xx}\dot{x}^2 - 2C_{xt}\dot{x} - C_x f_x - C_{yy}\dot{y}^2 - 2C_{yt}\dot{y} - C_y f_y - 2C_{xy}\dot{x}\dot{y} + C_{tt}). \quad (1)$$

Once we have λ, we simply calculate the constraint force $[\lambda C_x, \lambda C_y]$, and add it in to the applied force.

The formula we just derived is a template for any constraint. To implement a particular constraint, we need to derive expressions for all the derivatives that appear in equation 1—C_x, C_y, C_{xt}, etc. Having done so, it might seem that the next step is to substitute all those expressions into equation 1, but that's usually a bad idea, because you'll create a huge horrible mess which you then have to try to simplify and code up without flipping signs or dropping terms.

You can save yourself a *lot* of grief by doing the substitution *numerically*, at runtime. Here's how: first, write code that evaluates λ according to equation 1 given numerical values for everything. This is a universal code snippet that you can use for every constraint. Then, write code that evaluates each derivative, returning a numerical value. By plugging these pieces of code together, you can evaluate λ without ever seeing the giant horrible expression.

Now let's do a dumb constraint: let $C = ax^2 - y + bxt$, which constrains the particle to stay on a moving parabola. Here's everything we ever need to know about C to implement the constraint:

$$\begin{aligned} C &= ax^2 - y + bxt \\ C_x &= 2ax + bt \\ C_y &= -1 \\ C_t &= bx \\ C_{xx} &= 2a \\ C_{xy} &= 0 \\ C_{xt} &= b \\ C_{yt} &= 0 \\ C_{yy} &= 0 \\ C_{tt} &= 0. \end{aligned}$$

Pretty simple, huh? Less simple for complicated constraints, but still managable.

As for implementation, in C++ you'd define a base constraint class with a virtual method for each member of the above list. To implement a new constraint, all you'd need to do is define a derived class with the appropriate method definitions (if the default methods return zero, you'd only need to override the non-zero ones.) The base class would be responsible for invoking these methods to calculate λ and apply the constraint force, based on equation 1.