Inverse Kinematics and Optimization

Robert Bridson

October 7, 2011

1 Problems with Forward Kinematics

Putting what we have seen so far together, we have the makings of a basic keyframe animation system: we can rig a model with a skeleton, pose it at keyframes using Forward Kinematics, interpolate the motion through time using motion curve splines, skin it at every frame, and set up cameras for the final rendering.

However, the FK approach in particular isn't always so convenient. A classic example is provided by the problem of *footskate* in walking characters. When a foot is in contact with the ground while walking we expect it to stay fixed in place due to friction; if it erroneously moves in a visible way we might interpret the artifact, namely footskate, as being the actual result of a slippery or spongy surface. It can be very distracting.

When animating a walking character with FK one node in the tree has to be the root, and in particular at least one of the feet will *not* be the root, since there can only be one root; for simplicity we'll assume the hips are the root as usual. The animator directly specifies the motion of the root in the world with motion curves, but the motion of other nodes in the FK tree is a more complicated function of the motion of higher nodes in the tree and joint parameters.

To avoid footskate in FK, the animator has a tedious task of carefully adjusting hip, knee and ankle joint angles in response to the motion of the root so that a foot planted on the ground stays fixed (i.e. the complicated composition of the root's motion curves and the animated joint angles along the leg balances out to a constant).

Related problems happen with other types of contact, such as hands grasping objects like doorknobs. Simply switching the root temporarily to the foot (or hand, or whatever) to make maintaining a solid contact easy for the animator isn't possible when more than one contact is present—for example, a character standing on two feet opening a door with one hand while

keeping their other hand on a railing.

2 Inverse Kinematics

Figuring out the joint angles needed to maintain a contact essentially amounts to solving a set of equations—which is perfectly suited for automatic solution by the computer instead of manual artist intervention. This is the process of *Inverse Kinematics* (IK): automatically determining joint parameters in a skeleton to satisfy constraints on the output. Whereas FK computes the frames of nodes in the tree from the joint parameters, IK is the inverse, computing the joint parameters from requirements on the frames of nodes in the tree.

IK is useful both as a specific animation technique—we can designate that for a time interval a certain condition must hold true and require the computer to calculate the necessary joint parameters for that duration—and for interactive control over posing—with IK it's possible for an artist to directly drag a character's hand, say, to where it should be for a keyframe pose without having to tediously adjust all the individual joints along the way instead.

IK has a rich history in robotics, where a typical problem might be what parameters to send to joint servo motors to place the end of a robotic arm in a desired position and orientation. From this connection we get terminology such as *end-effector* for a node at the leaves of the FK tree which we wish to control by IK, such as the tool at the end of a robotic arm.

There are two main approaches to IK: special analytical solutions, and numerical methods for approximately finding a solution. Both have their place.

2.1 Analytical Solutions

In some standard scenarios, it's possible to analytically derive what joint angles must be to achieve the desired result.

For example, take a look at a typical limb restricted to two dimensions for simplicity. At the base of the limb (the hip or shoulder) there is a joint with one angle parameter θ, then a segment (upper leg or arm) of length a extending to another joint (knee or elbow) with angle parameter ϕ, then another segment (lower leg or forearm) of length b ending at the end-effector (ankle or wrist) we wish to control.

Relative to the base, the coordinates of the intermediate joint are

$$\big(a\cos\theta,\ a\sin\theta\big). \tag{1}$$

Taking into account the θ rotation of the first segment's frame, the coordinates of the end-effector are

$$\big(a\cos\theta,\ a\sin\theta\big) + \big(b\cos(\theta+\phi),\ b\sin(\theta+\phi)\big). \tag{2}$$

It should be noted this assumes that at rest, when $\theta = \phi = 0$, the limb extends straight along the positive x-axis of the base coordinate system.

Given a desired position (x, y) for the end-effector in the base coordinate system, the IK problem amounts to solving two equations,

$$\begin{aligned} x &= a\cos\theta + b\cos(\theta+\phi) \\ y &= a\sin\theta + b\sin(\theta+\phi), \end{aligned} \tag{3}$$

for the two unknown joint parameters θ and ϕ. Unfortunately, this system is heavily nonlinear, so it's not trivial to solve.

The clever route is to first figure out ϕ, matching the distance L between the base and desired position,

$$L^2 = x^2 + y^2, \tag{4}$$

with what the cosine law tells us the actual distance between the base and the end-effector must be:

$$a^2 + b^2 - 2ab\cos(\pi - \phi). \tag{5}$$

Noting $\cos(\pi - \phi) = \cos\phi$, we can solve this to get

$$\phi = \cos^{-1}\left(\frac{L^2 - a^2 - b^2}{2ab}\right), \tag{6}$$

if it's possible. Obviously if $L > a + b$ no solution can exist (the target is too far for the limb to reach even when fully out-stretched with $\phi = 0$); likewise if $L < |a - b|$ a solution is impossible (the target is too close for the mismatch in segment lengths, even when the limb is completely tucked back in).

If you're thinking carefully about your trigonometry, you'll notice there are actually two different angles for ϕ that provide the same cosine: the solution is not unique. Perhaps a physical constraint, such as the elbow or knee only being allowed to bend in one direction, can resolve the ambiguity.

We can then use the sine law to determine the angle ψ between the first limb segment and the line between the base and the target,

$$\frac{\sin\phi}{L} = \frac{\sin(\pi - \psi)}{b}, \tag{7}$$

giving that angle as

$$\psi = \sin^{-1}\left(\frac{b}{L}\sin\phi\right), \tag{8}$$

where we also used the identity $\sin(\pi - \psi) = \sin\psi$.

Finally, the sum $\theta + \psi$ is the angle pointing from the base to the target, which we can compute as

$$\tan(\theta + \psi) = \frac{y}{x}, \tag{9}$$

whence we get

$$\theta = \tan^{-1}\frac{y}{x} - \psi. \tag{10}$$

Obviously this formula would need to be modified slightly when $x = 0$.

2.2 Well-Posedness

The analytical approach is great, once the solution is fully worked out as above, but for general animation software that can support arbitrary skeletons there are several big problems in extending this:

- How can the program figure out a solution strategy for an arbitrary type of skeleton?
- How can we handle nastier constraints than simply placing an end-effector at a particular position, such as avoiding self-intersections in the skeleton?
- How do we deal with non-unique solutions?
- What do we do if there is no solution?

Although the capabilities of symbolic computer algebra shouldn't be underestimated, seeing as we only need numerical solutions accurate to within visual tolerances, there is a better way forward using techniques from scientific computing which will help with the first two problems.

But first let's deal with an even more fundamental problem: what to do about the non-unique or non-existant solutions. The concept of a *well-posed problem* from applied math is needed here.

A well-posed problem is one where we can mathematically guarantee that a single unique solution exists, and that it is stable to perturbations—meaning the solution only changes a small amount if we change the data of the problem a small amount. A problem that isn't well-posed is called *ill-posed*, and probably is pointless to try to "solve": if no solution exists, you can't solve it; if many solutions exist you don't know which to pick; if the solution can change wildly under tiny perturbations of the input data, and you don't know if you measured the input data exactly, then you can have no confidence that you've found a valid solution.

Often an ill-posed problem can be easily changed into a well-posed problem by adding or relaxing requirements. For example, finding x so that $x^2 = y$ for a given real number input y is ill-posed because there are usually two solutions. We can fix that by requiring that the solution x be non-negative—then there's at most one solution. However, we are still ill-posed because for negative y there's no non-negative solution (only complex-valued solutions). If we add the restriction that $y \geq 0$, then we finally have a well-posed problem—you can verify that $x = \sqrt{y}$ is the unique solution, and it is a continuous function of the input data y.

Another common strategy for making problems well-posed is to phrase them as *optimization* problems, looking for the best possible "solution" even if it doesn't perfectly solve the original problem. For example, solving a real 2×1 linear system for an unknown x,

$$\begin{pmatrix} a \\ b \end{pmatrix} (x) = \begin{pmatrix} c \\ d \end{pmatrix}, \tag{11}$$

is ill-posed. If $c/a \neq d/b$, or $a = 0$ but $c \neq 0$, or $b = 0$ but $d \neq 0$, there is no solution. Rather than add requirements on the data, we could instead change the problem to look for the optimal choice of x that comes as close as possible to satisfying both equations, say in terms of the Euclidean norm:

$$\min_x \left\| \begin{pmatrix} a \\ b \end{pmatrix} (x) - \begin{pmatrix} c \\ d \end{pmatrix} \right\|^2. \tag{12}$$

Remembering that the Euclidean norm squared of a vector (u, v) is just the sum of the squares of the entries,

$$\|(u, v)\|^2 = u^2 + v^2, \tag{13}$$

this sort of minimization problem is often called *least-squares*. If there was an exact solution to the original problem, $x = c/a = d/b$, it's still the unique solution to the least-squares problem—it succeeds in taking the Euclidean norm all the way down to zero. As an exercise, you can work out that the solution in the general case is

$$x = \frac{ac + bd}{a^2 + b^2}. \tag{14}$$

Unfortunately, we're still not quite well-posed: what if $a = b = 0$? Obviously our solution formula breaks down; any x is as good as any other, i.e. there are infinitely many solutions.

We can finally fix the problem by saying, for example, we want the minimum-norm x that minimizes the least-squares problem—so if $a = b = 0$, the unique solution is $x = 0$. Alternatively we can *regularize* the problem, changing it slightly, to avoid the breakdown in an easier way. Introduce a tiny parameter $\epsilon > 0$, and instead pose:

$$\min_x \left\| \begin{pmatrix} a \\ b \end{pmatrix} (x) - \begin{pmatrix} c \\ d \end{pmatrix} \right\|^2 + \epsilon \|x\|^2. \tag{15}$$

This is at last a well-posed problem, which has the unique solution

$$x = \frac{ac + bd}{a^2 + b^2 + \epsilon} \tag{16}$$

which does vary continuously with the input data, no matter what. It balances staying close to both equations when they are well defined with keeping a small norm answer x when the equations vanish.

This last example is a bit contrived—we rarely have exactly two linear equations with one variable—but is the perfect model for what we're about to do with more nonlinear equations and more variables.