Inverse Kinematics in Character Animation

Daniel Perry

Overview

0.1 Summary

”Kinematics is the study of classical mechanics which describes the motion of ... systems of bodies ... without the consideration of the causes of motion.” [7]. If the mechanical system being observed is parameterized, inverse kinematics is an approach to esimating the parameters to produce specific positions or effects in a mechanical system. [6],[4].

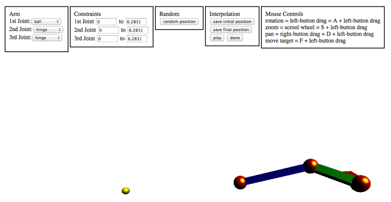
Inverse kinematics has been used extensively in fields like robotics - how to achieve specific effects by a robot or appendage - and character animation - how to achieve specific effects by a computer model.

Character animation is an intersting application of inverse kinematics where specific characters or objects are parameterized in a way to give an artist the ability to pose and move characters with the underlying goal of aesthetics and (often) of simulating real life interaction.

Here I explore one simple example of inverse kinematics as applied to an arm. While the arm is relatively simple model it still provides for a nice variety of problems to solve because of various joints available.

0.2 Contribution

For my project I implemented a simple WebGL visualization of an arm with some sipmle controls along the top, see figure1.



1st Joint: ball
2nd Joint: hinge
3rd Joint: hinge

1st Joint 0 to 6.2831
2nd Joint 0 to 6.2831
3rd Joint 0 to 6.2831

rotation = left-button drag = A + left-button drag
zoom = scrool wheel = S + left-button drag
pan = right-button drag = D + left-button drag
move target = F + left-button drag

Figure 1: User interface to the arm display.

My implementation uses a 3-joint arm where each joint can be one of three types: hinge (single-axis rotational), prismatic (single-axis translational), and ball joints. The interface also provides the option to constrain each of the joints.

Once the joint types and constraints have been set, the user can interact with the arm by moving around a target point. The system is continually solving the inverse kinematics problem for the current goal point.

There is also an interface to setup a replay of the animation. The user sets of an initial pose of the arm, and then a final pose, and the system will generate an animation sequence between the two poses.

I made use of several javascript libraries to help with the development:

1. three.js [2] - was used for most of the 3D graphics and scene organization code, as well as the quaternion and vector data structures.

2. numeric.js [1] - was used for arbitrary size matrix operations (to permit the code to be used on an arm with more linkages).

3. underscore.js [3] - was used for some functional code utilities.

An online demo (with source code) is available [5].

1 Problem solution

To solve the inverse kinematics problem I first find a relationship between the parameterization of the model and the goal position, taking into account the constraints.

For example if the parameterization where each joint i is controled by the parameter θ_i is considered, the two can be related like so:

$J(\Theta)\dot{\Theta} = \dot{Y}$

where $\dot{Y}$ is the change of the end-effector to the goal position.

Once this relationship has been established the instantaneous change in parameters can be solved for,

$\dot{\Theta} = J(\Theta)^{-1}\dot{Y}$

1.1 Jacobian

The Jacobian for this model is fairly straitforward to compute for the rotational joints, where:

$J = \left[A_1 \times \bar{J_1E}, \ldots, A_n \times \bar{J_nE}\right]$

Where A_i is the unit vector of the axis of rotation, $\bar{J_iE}$ is the vector going from the joint location to the end-effector for joint i, and where each column vector $A_i \times \bar{J_iE}$ is the cross product between the two.

For the 1-d rotational joint (hinge joint), this is very straitforward because $A_i = R_i(0,0,1)^T$, where R_i is the rotation applied by previous joints.

For the ball joint computing the axis of rotation was a little more tricky. I compute the axis of rotation for the ball joint like os:

$A_i = V_i \times \bar{J_iG}$

where V_i is the unit vector showing the current local orientation of the segment following the joint, and $\bar{J_iG}$ is the vector from the joint location J_i and the location of the target goal G. Using the cross product computes the axis of rotation that will move the segment towards the goal - in essence choosing the axis of rotation.

In the case of a prismatic joint, the relationship between the end effector and the goal and the joint parameterization is simply a linear projection of $G - E$ onto the axis of translation,

$\dot{X} = ((G-E)^T A_i)A_i$

1.2 Solving using $J^T \approx J^{-1}$

Once the model has been formulated and the Jacobian is constructed the value $\dot{Y}$ is computed, which is essentially the difference between the end effector E and the goal position G (hinted at in the prismatic example above),

$\dot{Y} = G - E$

The result is a linear system $Ax = b$, where $A = J(\Theta)$, $b = \dot{Y}$, $x = \dot{\Theta}$.

The matrix of the system isn't guaranteed to be positive-definite or even square $J(\Theta)$, so solving the system usually involves computing a pseudo-inverse,

$J^{+} = J^T(JJ^T)^{-1}$

The pseudo-inverse can still have singularities [4], and so sometimes it can be beneficial to augment the pseudo-inverse,

$J^{+}_{A} = J^T(JJ^T + \lambda^2 I)^{-1}$.

However, computing the inverse every frame can be expensive (especially for a large configuration space) and involves risking some numerical singularities and the selection of an additinal parameter λ. Another approach is to use the transpose of the Jacobian to approximate the inverse:

$J^{-1} \approx J^T$

While not exact, for a simple model where the dynamics are more of interested this seems to perform fine.

This works because multiplying the the Jacobian transpose is essentially projecting the portion of the Jacobian velocity onto the direction towards the solution. Because this is a poor approximation, the effect is limited by using another step parameter α, so that

$\dot{\Theta} = \alpha J^T \dot{Y}$

While this could be considered yet another parameter to tune, in fact any other instantaneous solution will have to use a step size making it comparable in that regard.

1.3 Results

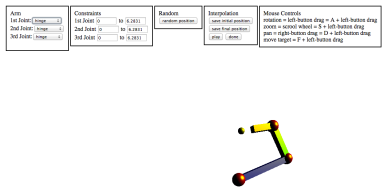
While the current system is capable of $3^3 = 27$ unique arm configurations, here three different arms will be described with screen shots of the system. To see examples in action, please see the accompanying movie demonstration.

1.3.1 Three Hinge Arm

The three-hinge-arm is the most basic arm configuration, where all three joints have one degree of freedom - the angle of rotation, an image of the solution is shown in Figure 2.

As you can see the system is able to find a suitable solution.

The gap between the goal point and the end-effector is most likely caused by the use of the J^T for the solution. However from use of the system, I didn't find the gap to make a big different, as the human visual system makes up for the gap quite easily - once the user gets used to the gap, the interaction becomes pretty natural.



Arm
1st Joint: hinge
2nd Joint: hinge
3rd Joint: hinge

Constraints
1st Joint 0 to 6.2831
2nd Joint 0 to 6.2831
3rd Joint 0 to 6.2831

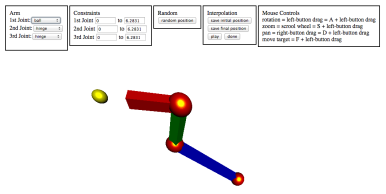
Mouse Controls
rotation = left-button drag = A + left-button drag
zoom = scrool wheel = S + left-button drag
pan = right-button drag = D + left-button drag
move target = F + left-button drag

Figure 2: 3 hinge joint arm

1.3.2 Ball-Hinge-Hinge

Another arm demonstrated here has a ball joint at the base of the arm, with hinge joints at the other two joint locations. This arm has more degrees of freedom because the ball joint can rotate about any axis in 3D.

Figure 3 shows the solution using this configuration. Figure 4 shows the same solution from a slightly different view to make the non-planar solution for obvious.

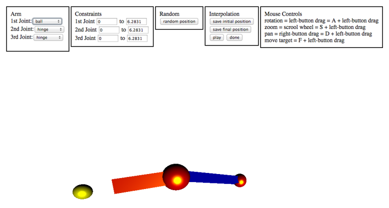


Arm
1st Joint: ball
2nd Joint: hinge
3rd Joint: hinge

Constraints
1st Joint 0 to 6.2831
2nd Joint 0 to 6.2831
3rd Joint 0 to 6.2831

Mouse Controls
rotation = left-button drag = A + left-button drag
zoom = scrool wheel = S + left-button drag
pan = right-button drag = D + left-button drag
move target = F + left-button drag

Figure 3: Arm with joint configuration: Ball, Hinge, Hinge



Arm
1st Joint: ball
2nd Joint: hinge
3rd Joint: hinge

Constraints
1st Joint 0 to 6.2831
2nd Joint 0 to 6.2831
3rd Joint 0 to 6.2831

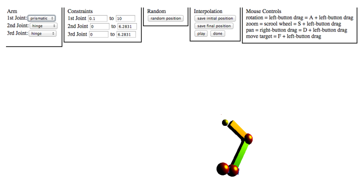
Mouse Controls
rotation = left-button drag = A + left-button drag
zoom = scrool wheel = S + left-button drag
pan = right-button drag = D + left-button drag
move target = F + left-button drag

Figure 4: Ball, Hinge, Hinge from a different view, demonstrating non-planar configuration.

1.3.3 Prismatic-Hinge-Hinge

The final arm shown here has a prismatic joint at the bottom and two hinge joints. The prismatic joint still only has a single degree of freedom, but moves translationally instead of rotationally.

Figure 5 shows one solution using this arm configuration. As you can see the prismatic joint shrinks and expands to move the end-effector closer to the goal position.



Arm
1st Joint: prismatic
2nd Joint: hinge
3rd Joint: hinge

Constraints
1st Joint 0.1 to 10
2nd Joint 0 to 6.2831
3rd Joint 0 to 6.2831

Mouse Controls
rotation = left-button drag = A + left-button drag
zoom = scrool wheel = S + left-button drag
pan = right-button drag = D + left-button drag
move target = F + left-button drag

Figure 5: Arm with joint configuration: Prismatic,Hinge,Hinge.

2 Conclusion

2.1 Summary

This project explored a simple inverse kinematics system of an arm with various joints.

This project demonstrated three different arms and described how constraints can be applied to the system. It also described the interface to construct an animation by selecting two key poses, where the system computes the intervening motion. The project has also demonstrated how a simple transpose of the Jacobian can be used to approximate a solution to the inverse kinematics problem.

While simple, the kinematic system of an arm has an interesting array of challenges, especially in handling a variety of joint types.

It would be interesting to examine more fully how a weighting could be applied to the system or to explore how a more natural arm movements could be attained. It might also be worth exploring some kind of skinning to make the model more realistic.

References

[1] numeric.js, 2014. `http://www.numericjs.com/` [Online; accessed 22-September-2014].

[2] three.js, 2014. `http://threejs.org/` [Online; accessed 22-September-2014].

[3] underscore.js, 2014. [`http://underscorejs.org/` Online; accessed 22-September-2014].

[4] Rick Parent. *Computer animation: algorithms and techniques*. Newnes, 2012.

[5] Daniel Perry. project demo, 2014. [`http://www.cs.utah.edu/~dperry/class/character_animation/ik/project1.html` Online; accessed 22-September-2014].

[6] Wikipedia. Inverse kinematics — wikipedia, the free encyclopedia, 2014. [Online; accessed 22-September-2014].

[7] Wikipedia. Kinematics — wikipedia, the free encyclopedia, 2014. [Online; accessed 22-September-2014].