INTRODUCTION

Goal

What did we try to do?

We tried to implement inverse kinematics and to get a simulated robot arm to automatically position its joints in order to get its hand to reach a target sphere. Inverse kinematics uses kinematics equations to calculate the necessary joint parameters of a robotic arm to achieve a desired position of its end effector (e.g. a hand).

Who would benefit?

Inverse kinematics is used widely in robotics to control robotic limbs. The application that is more related to computer graphics is its use in animation and the position of joints in 3D models. Inverse kinematics is already fairly robustly implemented in industry applications, so the main benefit is to us, who will hopefully learn the fundamentals of IK.

Previous Work

What related work have other people done? When do previous approaches fail/succeed?

A number of different methods have been employed by people in inverse kinematics. These methods, which will be described and discussed below, include algebraically, cyclic coordinate descent, and the Jacobian.

Algebraically: calculate the joint angles by manually inverting the forward kinematics solution for the end-effector position. For example, the end effector position for a system with 2 DOF can be expressed as $X_{EE} = \left(l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2),\ l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2)\right)$, giving the following expressions for θ_1 and θ_2: $\theta_1 = \dfrac{-(l_2 \sin\theta_2)x + (l_1 + l_2 \cos\theta_2)y}{(l_2 \sin\theta_2)y + (l_1 + l_2 \cos\theta_2)x}$, $\theta_2 = \cos^{-1} \dfrac{x^2 + y^2 - l_1^2 - l_2^2}{2 l_1 l_2}$. It is easy to see that this is really only practical for cases where the DOFs are limited to 2 or 3, as it gets prohibitively complicated past that. Additionally, the solution does not always exist, and if it does, it is not always unique nor continuous.

Cyclic coordinate descent: Minimization of an objective function $E(\theta) = \left(P - X(\theta)\right)^2$ is applied to each joint separately, with passes going from the most distant segment to the base segment. $E(\theta)$ is the error, P is the goal position of the end effector, and $X(\theta)$ is the current position of the end effector as a function of the joint angles. A number of passes are made to find the global minimum of the error equation. This method works fine for simple structures, but is not as effective for complicated ones. Also, when the change that must be done is near the base, a lot of wasted passes occur, which may slow down computation. Finally, the result is more like a chain than rubber, which may not be what is desired for figure animation. A few things this method excels at though are that it is free of singularities (where a function takes an infinite value), it does not include matrix inversion, and in most cases, only a few passes would be required, so computation is reasonably fast.

Jacobian matrix: The Jacobian is defined as the multidimensional extension to the differentiation of a single variable. Essentially, it shows how changes in the various joint angles affect the Cartesian position of the end effector via the formula $J\Delta\theta = \vec{e}$, where $\vec{e}$ represents the desired change in the position of end effector and $\Delta\theta$ is the amount to update the joint angles at each joint. The variable we are most interested in is $\Delta\theta$, so the equation is transformed to $\Delta\theta = J^{-1}\vec{e}$, and the problem is reduced to finding the Jacobian and then inverting it.

Entries in the Jacobian are actually quite easy to calculate. If the jth joint is a rotational joint with 1 DOF, the joint angle is simple the scalar θ_j. Now let p_j be the position of the joint, and v_j be the unit vector pointing along the axis of rotation (which will always be (0,0,1) in our 2D implementation), and s be the position of the end effector. The corresponding entry in the Jacobian is $\frac{\partial s}{\partial \theta_j} = v_j \times \left(s - p_j\right)$. Finding the inverse is the difficult part. First, the Jacobian matrix's dimension grows with the degrees of freedom, as does the cost of calculating the inverse. Furthermore, a matrix that is rectangular (such as any Jacobian matrix that does not have 6 DOF in 3D space) is not invertible. To combat this, two approximations, the transpose and the pseudo inverse are used instead.

The idea behind using the transpose is that, if the changes in angles calculated using the transpose are sufficiently damped (by multiplying by a factor $\alpha > 0$), the result will reasonably approximate those of the inverse.

The pseudo inverse, $J^{\dagger} = J^T\left(JJ^T\right)^{-1}$, is convenient in that it is defined for all matrices, even ones that are not square or not of full row rank, which would have made them non-invertible. Additionally, the pseudo inverse gives the best possible solution to the equation $J\Delta\theta = \vec{e}$ in terms of least squares. However, the pseudo inverse does have stability problems in the neighborhoods of singularities, which means that if the target position is not reachable or the arm is pulled taut, small changes in the target position will cause really big fluctuations in joint angles, which does not look natural.

Approach

What approach did we try? Under what circumstances do we think it should work well? Why do we think it should work well under those circumstances?

To get the benefits of easy calculability and convenience provided by the pseudo inverse, but to also reduce the effects of singularities, we used the damped least squares method. Instead of directly finding the solution to $J\Delta\theta = \vec{e}$, this method tries to find $\Delta\theta$ such that it minimizes the equation $\left\|J\Delta\theta - \vec{e}\right\|^2 + \lambda^2\left\|\Delta\theta\right\|^2$, where λ is a non-zero damping coefficient. This is equivalent to solving $\Delta\theta = J^T\left(JJ^T + \lambda^2 I\right)^{-1}\vec{e}$. Through trial and error, our λ ended up needing to be 10 to achieve smoothness. By using this method, our program works well for any number of joints and still behaves in the vicinity of singularities, making it look smooth for essentially any target position.

METHODOLOGY

What pieces had to be implemented to execute my approach?

For each piece…

Were there several possible implementations?

If there were several possibilities, what were the advantages/disadvantages of each?

Which implementation(s) did we do? Why?

What did we implement?

What didn't we implement? Why not?

- Results
  - How did we measure success?
  - What experiments did we execute?
  - What do my results indicate?
- Discussion
  - Overall, is the approach we took promising?
  - What different approach or variant of this approach is better?
  - What follow-up work should be done next?
  - What did we learn by doing this project?
- Conclusion