Department Of Computer Science, University of Bath

Inverse Kinematics Coursework

CM50244 Computer animation and games I

Garoe Dorta Perez, Dave Hibbitts, Ieva Kazlauskaite, Richard Shaw
Unit Leader: Prof Phil Willis

November 17, 2014

1 Introduction and Objective

In this project we explore the application of inverse kinematics in three-dimensional space. First, we construct a simple composite object which contains long rigid parts that are connected so as to allow flexibility of movement. The joints connecting the rigid rods are restricted and can either rotate in plane and act like simple hinges. Second, the method of inverse kinematics is explored in order to determine the configuration of the object, i.e. the relevant angles of the joints, when the object is given a task. The method works by planning the motion towards a goal, which is a point in three-dimensional space subject to constraints. Finally, the movement of the object is animated, and a number of non-physical (?) parameters are adjusted to deliver a visually appealing performance.

This report provides a brief account of the objectives of the project and the methods that were explored. Moreover, the issues that were encountered are discussed including problems concerning both the computations and the implementation. We discuss the implementation in Matlab and the computational method that we use for inverse kinematics. We proceed by explaining the operations and execution in OpenGL. The last section of this report includes a review of the performance and the limitations of the method, as well as a discussion of possible further improvements.

2 Related Work

What People do?

3 Procedure

In this section we discuss our approach to solving the problem at hand. We started by considering the corresponding two-dimensional situation where a planar object with a predetermined number of links is reaching towards a point in the plane. Our first attempt at a solution was to use forward kinematics where at each time-step we calculated the subsequent position using the current state and the desired position of the end-point. The first implementation was done in Matlab as we wanted to ensure that the method works as desired before exporting it to the OpenGL framework. In the meantime, we started working on the basic structure in OpenGL, for instance we drew the necessary primitives, such as triangles, squares and lines in two-dimensions, and proceeded by including some appropriate data structures and some primitive graphical user interface.

The next step was to apply inverse kinematics in the case of our two-dimensional problem. We chose to use the Inverse Jacobian technique and implemented the method with both full inverse of the Jacobian matrix as well as the pseudoinverse. The motion of the object was displayed graphically, and we altered the number of limbs in order to test the performance of our method. At this stage the method still had a number of shortcomings even though it was restricted to motion in a plane. Firstly, the motion was not stable when the object was reaching towards a point that was outside the circular region. Secondly, the method favoured the movement of certain limbs without us explicitly specifying the constraints. We also improved the OpenGL GUI by adding mouse capture, and added chain class with bones and joints (why???). In both Matlab and OpenGL the system is described using relative angles, i.e. starting at the origin, each subsequent angle is defined in the coordinate system of the previous one (picture???).

The main questions we faced at this point had to do with local behaviour of each joint and the numerical method used to solve the optimisation problem. The movement of the first few joints (those closest to the origin) appeared significantly more restricted than the movement of the end joints. To solve this issue, we normalise to get global behaviour of the whole system as opposed to local behaviour of each joint. In addition, the simulation naturally slows down as it gets very close to the destination due to the nature of the numerical method which tends to oscillate around the minimum point. This behaviour can be advantageous (especially in robotics) as it results in slow-down motion as it gets close to reaching the goal (e.g. grabbing an object or touching a surface) so as to avoid a severe collision.

We also note that the motion of the system is task-dependent, hence favouring the movement of a number of selected joints is reasonable. For example, if we assume that the object we are modelling is an arm, and the motion is defined as the arm reaching for a nearby object, it is logical to assume that the rotation of the elbow joint will be favoured against the rotation of the shoulder joint, etc. Therefore, we introduce a weight vector that is used to control the importance of the motion of each joint, and added constrains on the angles.

We continued to work on the graphical implementation and improved it in a number of ways. The simulation and rendering were separated into two different threads so that we can analyse the modelling independently from the displaying. What is more, we started displaying the trace left by the tip of the object which makes it easier to track the motion of the object and adds visual appeal.

At this stage the Matlab and the OpenGL implementations were still independent, so we started importing the numerical method from Matlab to OpenGL. The resulting model was two-dimensional, used inverse kinematics, could be adapted to any number of limbs/joints, and had a simple user interface.

Naturally, the next step was to transform the model to the three-dimensional space. We first adapted the graphical framework to three dimensions, i.e. we could rotate the camera, and place a target anywhere in the space, however the object and the motion were still confined to a plane.

Let us now consider the implementation in both Matlab and OpenGL in more detail. The following two sections give a detailed account of the issues encountered during the implementation process, the solution methods we employed and the explanation of why we chose to use the particular approaches.

3.1 Start with Matlab

1. Issues
2. How we solve them
3. Why we use what we use (e.g. Inverse Jacobian)

3.2 Port the code to C++, OpenGL

1. Issues
2. How we solve them

3.3 Exploration

The following properties are to be explored.
1. Some way of indicating where the remote end of the linkage should move to in 3-space.
2. A way to change the physical properties such as rate at which the joints can change and the slow-out and slow-in of the movement of the end of the linkage as it leaves it current position and approaches the target position respectively.
3. Whether to model and thereby vary the flexibility of the rods.

We are allowed to work out something for one joint and explain how we'd apply it to other and so on.

3.4 Summary and Conclusion

What the final thing would look like if we had more time to work on it?