Inverse Kinematics Programming Assignment

CS 448D: Character Animation

Due: Wednesday, April 29th 11:59PM

1 Logistics

In this programming assignment, you will implement a simple inverse kinematics solver to compute poses for an articulated skeleton. The assignment is intended to be completed in Matlab. Matlab is installed on all of the "Myth" computers in the teaching lab in the basement of Gates, room B08. If you are unfamiliar with Matlab, you may want to glance at some of the tutorials available here: `http://www.mathworks.com/academia/student_center/tutorials/launchpad.html`.

You should begin by obtaining the starting code from the course website.

To submit the assignment, e-mail all relevant files to **cs448d-spr0809-staff@lists.stanford.edu**. Specifically, the following files should be included in your submission: InitialPoseHuman.m, Jacobian.m, ComputeIK.m, InterpolateFrame.m, EulerToQuat.m, QuatToEuler.m, QuatSlerp.m, README. You are welcome to play around with modifying the other *.m files, but please do not submit them. We should be able to run your solution simply by replacing the default InitialPoseHuman.m, Jacobian.m, ComputeIK.m, InterpolateFrame.m, EulerToQuat.m, QuatToEuler.m and QuatSlerp.m files with your implementation.

This assignment is meant to be completed individually.

2 Introduction

In this assignment, you will implement the iterative IK solver described in class using the Matlab linear system solver, and then implement spherical quaternion interpolation to interpolate between keyframes computed with IK. Specifically, the following components must be completed:

1. ComputeIK.m - this function should implement an iterative IK algorithm that uses the Jacobian to compute a pose satisfying specified end effector constraints.
2. Jacobian.m - this function should compute the Jacobian given the state of the system.
3. InitialPoseHuman.m - this function should set the initial pose of the human character in a way that allows the system to converge to a good optimum.
4. InterpolateFrame.m - this function should smoothly interpolate between two sets of parameters, using a spherical quaternion interpolation to interpolate the ball and socket joint of the right hip.
5. EulerToQuat.m, QuatToEuler.m, QuatSlerp.m - these utility functions should be implemented to convert between Euler angles and quaternions and perform spherical interpolation.

6. README - this file should contain answers to questions in Section 9.

3 The Iterative IK Algorithm

In order to implement the basic iterative IK algorithm, you will need to fill in the missing code in ComputeIK.m and Jacobian.m. Jacobian.m contains the framework for a function to compute the Jacobian, and the comments in ComputeIK.m contain some hints for useful functions you may need to call in your iterative algorithm. You are not required to implement good convergence checks (it is sufficient to just have the algorithm run for a certain sufficiently large number of iterations), but it may be a good idea in order to get the animation test running fast enough.

4 Setting Initial Pose

In order to pass TestMedium.m, you may need to modify the starting values of θ in your algorithm. Do not tailor these values to the specific tests, but rather select values that improve the algorithm's chances of convergence. You will be asked to justify your choices in the README file.

5 Keyframe Interpolation

In the next component, you will improve the keyframe interpolation in InterpolateFrame.m. To do this, you will first need to implement EulerToQuat.m, QuatToEuler.m, and QuatSlerp.m to perform the necessary quaternion operations. Remember to always interpolate across the smaller angle. The formula for generating a quaternion $q_1 + iq_2 + jq_3 + kq_4$ from Euler angles α, β, γ is:

$$q_1 = cos(\alpha/2)cos(\beta/2)cos(\gamma/2) + sin(\alpha/2)sin(\beta/2)sin(\gamma/2)$$

$$q_2 = sin(\alpha/2)cos(\beta/2)cos(\gamma/2) - cos(\alpha/2)sin(\beta/2)sin(\gamma/2)$$

$$q_3 = cos(\alpha/2)sin(\beta/2)cos(\gamma/2) + sin(\alpha/2)cos(\beta/2)sin(\gamma/2)$$

$$q_4 = cos(\alpha/2)cos(\beta/2)sin(\gamma/2) - sin(\alpha/2)sin(\beta/2)cos(\gamma/2)$$

And the formula for generating Euler angles from a quaternion is:

$$\alpha = \text{arctan2}\left(\frac{2(q_1q_2 + q_3q_4)}{1 - 2(q_2^2 + q_3^2)}\right)$$

$$\beta = \arcsin(2(q_1q_3 - q_4q_2))$$

$$\gamma = \text{arctan2}\left(\frac{2(q_1q_4 + q_2q_3)}{1 - 2(q_3^2 + q_4^2)}\right)$$

With these three functions implemented, you should modify InterpolateFrame.m to use spherical interpolation for the right hip, which consists of joints 17, 18, and 19. To do this, convert the two keyframes for the joint into quaternions, interpolate between them, and convert back to parameter angles. This is somewhat inefficient, but is sufficient for this assignment.

You will also need to modify the interpolation so that the angular velocity of all joints is 0 at each keyframe. This can be accomplished with Hermite interpolation as discussed in lecture 2.

6 Evaluation

Your assignment will be evaluated on how well it can find an IK solution for each of the three tests. A correct solution will converge to a pose that very nearly satisfies the end effector constraints for each of the tests.

In addition, TestAnimation.m should run fast enough to render the walking character on the "Myth" machines at a reasonable pace. There is no hard requirement, but if the animation is taking more than a few minutes to complete, your implementation may not be correct.

Finally, the interpolation test should interpolate between the three keyframes, slowing down at each keyframe and interpolating the right hip smoothly through the shortest path, as is the case with spherical quaternion interpolation.

7 Useful Matlab Tips

Some useful Matlab tips you may need to use in this assignment:

- $x = A\backslash b$ - this command solves the linear system $Ax = b$, even when A is not a square matrix.
- $v = A(:)$ - this command converts a $N \times M$ matrix A into a NM-dimensional vector v which consists of the concatenation of the columns of A.
- If you would like to rotate your view in the plot viewer to see the skeleton from different angles, go to "View" and select "Camera Toolbar." To rotate the camera, select the leftmost tool in the new camera toolbar and click and drag inside the plot.

8 The Skeleton

While your IK algorithm should be general enough to handle any skeleton, you may want to look more closely at the skeleton structure for determining a good initial pose. This assignment uses a simplified human skeleton with 20 degrees of freedom to represent the character. Each joint is a rotational joint that rotates about either the x, y, or z axis. The root, shoulders, and hips actually consist of three joints, each rotating about a different axis. Some of these joints have a link length of 0 and therefore appear to occupy the same physical space. No rotational constraints are placed on any of the joints. Figure 1 shows the skeleton in the default initial position, with each joint and its axis of rotation v_i labeled. You may also refer to HumanSkeleton.m for a list of all joints and their indices.

9 Questions

You should include a plain text file entitled "README" with your submission that contains answers to the following questions:

9.1 Question 1

Justify your choice of starting pose in InitialPoseHuman.m. Why did you choose this specific pose? What problems did you find with the default initial pose? Why does your solution solve these problems?

9.2 Question 2

What other unexpected problems did you encounter with your IK implementation? How did you solve these problems?

10 Code Overview

- ComputeIK.m - this file should contain an implementation of an iterative IK algorithm. You will need to fill in the missing code to complete this function.
- ComputeMatrices.m - this function accepts the parameters of the current state and generates complete transformation matrices for each joint. You will not need to modify this function.
- ComputeRelation.m - this function accepts skeleton parameters and generates the matrix C, which describes which joints influence which other joints. You will not need to modify this function.
- DrawSkeleton.m - this function draws the skeleton and end effector constraints in a specified configuration. The end effector constraints are shown as red circles. You will not need to modify this function.
- ForwardKinematics.m - this function computes the forward kinematics of a skeleton given the current configuration, producing joint positions p, world-space joint axes v, and end effector positions s. You will not need to modify this function.
- HumanSkeleton.m - this function creates the human skeleton. See Section 8 for details on the specific skeleton used in this assignment. You will not need to modify this function.
- Rotation.m - this function returns a rotation matrix given an axis and an angle. You will not need to modify this function.
- InitialPoseHuman.m - this function sets the initial pose for the skeleton. You will need to implement this function to set an initial pose that is general enough to solve all of the poses.
- Jacobian.m - this function computes a Jacobian given joint positions p, joint axes v, end effector positions s, relations matrix C, and the numbers of joints and end effectors (N and K respectively). You will need to implement this function.
- InterpolateFrame.m - this function interpolates between the parameters of two keyframes, specified by $theta1$ and $theta2$, according to parameter u. You will need to modify this function to interpolate with a smooth cubic and use spherical quaternion interpolation for the right hip.
- QuatToEuler.m - this function converts a quaternion into three Euler angles. You will need to implement this function.
- EulerToQuat.m - this function converts three Euler angles into a quaternion. You will need to implement this function.

- QuatSlerp.m - this function performs spherical interpolation between quaternions $q1$ and $q2$ according to parameter u. You will need to implement this function.
- TestEasy.m - this function uses your IK code to test an easy pose.
- TestMedium.m - this functino uses your IK code to test a more difficult pose. If implemented correctly, your IK code should satisfy all of the constraints - the end effectors should overlap exactly with the red circles.
- TestAnimation.m - this function uses your IK code to animate a walking character. If implemented correctly, your IK code should cause the character to walk in a believable manner.
- TestInterpolation.m - this function uses your IK code to animate a kicking character using keyframing. If both the IK and interpolation portion are implemented correctly, the character should swing his right leg in two smooth motions.

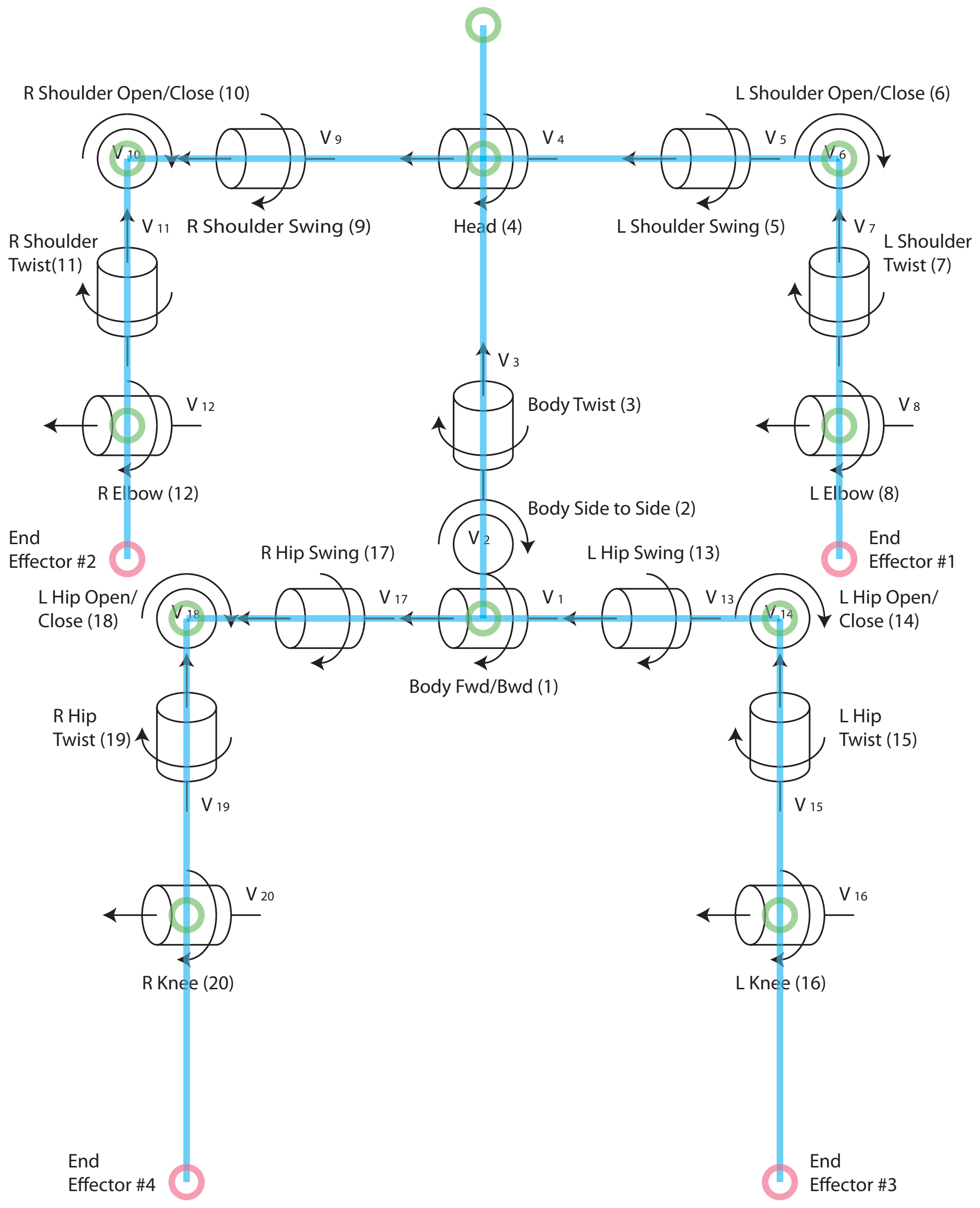

Figure 1: Schematic illustration of the skeleton used in this assignment. The axes and pivots of all joints are labeled.