CS184 Final Project Report

Stephen Tu
University of California, Berkeley
stephen_tu@berkeley.edu

1. PROBLEM STATEMENT

The final project was to build an inverse kinematics solver for an arbitrary linked robot, and then to provide a system for doing keyframe animations with it. For the project, I limited the scope to only handling tree structure robots, so each joint could only have at most one parent. This assumption was useful in simplifying the computation of the Jacobian matrix.

2. DEVIATIONS FROM INITIAL PROPOSAL

In my initial proposal, one of the methods I proposed to implement was *selectively* damped least squares (SDLS) presented in [4]. I chose not to do this however because on further inspection, the SDLS method was just a refinement on the damped least squares method (which I did implement) that simply picked the damping factor appropriately instead of having it set manually. Since I was already getting decent results by setting the damping factor manually, I figured that SDLS would not provide much of an advantage (definitely not noticeable visually) to this project so I decided to focus on the constraint problem and animations instead, both which I did not initially propose.

3. DESCRIPTION

The program is interacted with via a GUI. In this GUI, the user uses his mouse to click and drag effectors around the screen. The program automatically solves the inverse kinematics in real time, updating the robot continuously to match the movement of effectors. The user is allowed to drag effectors into configurations which the robot cannot possibly achieve. In this case, a best fit solution is computed which attempts to minimize the error. The root node of the robot can also be moved similarly to how effectors are moved.

The user can adjust the camera settings by using the same set of shortcut keys used in Assignment 5 (rotations, translations, scales). The coordinate axes are drawn in the lower right hand corner of the screen and updated accordingly as the user manipulates the camera.

The user can also add frames to an animation buffer by pressing the **a** key, which takes the current configuration of the robot and appends its state to a buffer. Furthermore, the user can right click on the GUI to display a pop-up menu exposing options relating to animation, such as clearing the animation buffer, toggling the animation on/off, and changing the playback rate of the animation. If the user turns the animation on, each frame in the animation buffer is displayed back to the user. This animation loops continuously until the user turns off the animation.

Independent from the animation system, the user can also save a current robot configuration by pressing the **m** key and restore the robot to the saved configuration by pressing the **r** key.

4. SUMMARY OF FEATURES

These are the inverse kinematics methods which I implemented for solving the problem:

- Pseudoinverse
- Damped least squares
- Generalized Jacobian transpose
- Gradient descent

Here is a summary of the implemented features:

- Rotation joints (1-DOF)
- Ball and socket joints (3-DOF)
- Translation joints (1-DOF)
- Positional constraint enforcement
- Keyframe animation system

5. PROJECT WEBSITE AND RESULTS

To see the program in action and the results of my implementation, please visit the final project webpage which shows a variety of videos [1] of the program in action.

A white-paper I wrote [9] detailing the technical aspects of this project is also linked to on the website. It goes into a fair amount of detail outlining the various algorithms which I implemented in this project.

6. CITATIONS

In this section I list the sources used to implement this project.

The project was written in C++ using OpenGL, STL, and the Armadillo linear algebra library [8].

The ideas for the pseudoinverse and damped least squares method were provided by [3] and [6]. The ideas for the generalized Jacobian transpose methods came from [10]. The ideas for formulating the problem as a non-linear optimization problem came from [5] and Robert Carroll.

A lot of OpenGL tricks which I used came from [2].

7. CHALLENGES AND SOLUTIONS

One of my first challenges was figuring out how to represent ball and socket joints (3-DOF). At first I tried to use Euler angles, but this turned out to perform quite poorly because of the co-dependencies between the angles. As a result, solutions would often shoot off in the wrong direction and it looked really bad. To solve this issue, I instead used a single angle to represent rotation about an arbitrary axis suggested by [7]. To compute the axis of rotation at any given point, I simply looked at the position of the effector and the position of the goal; the rotation axis was the axis which caused the effector to move directly towards the goal. Mathematically, if p is the position of the effector, z the position of the joint, and r the position of the goal, then the rotation axis v is:

$$\hat{v} = \frac{(p-z) \times (r-p)}{||(p-z) \times (r-p)||} \tag{1}$$

where $|| \cdot ||$ is the L_2 norm of a vector. Even though this complicated the implementation a bit since now I had to keep track of the last rotation axis used, the results were much better than Euler angles.

Another challenge I had was trying to enforce inequality constraints on the inverse kinematics solution using lagrange multipliers. These were tricky because they required the minimization of the Lagrangian function $L(\mathbf{x})$, but since the Lagrangian is not a residual error function anymore, simple techniques like Gauss-Netwon or Levenberg-Marquardt were no longer applicable. I was able to implement gradient descent to do the non-linear minimization, but this resulted in very poor convergence behaviors. If I had more time I would have tried to implement other non-linear optimization methods such as BFGS or other quasi-Netwon methods.

Finally, the last notable challenge was in tuning all the various parameters to achieve good performance. These include things like tolerances for when to stop iterating, the damping factor in the damped least squares method, etc. In the end I just ended up tuning everything with trial and error; there were no theoretical justifications for the constants which I chose.

8. CONCLUSION

This report outlines the system that I implemented and some of the challenges that I faced. Please look at [9] for the technical aspects of this project which I covered. Also please do not forget to checkout the [1] for videos of the system in action.

9. REFERENCES

[1] Cs184 final project website. `http://inst.eecs.berkeley.edu/~cs184-cv/finalproj`.

[2] Neon helium opengl tutorials. `http://nehe.gamedev.net`.

[3] S. R. Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods.

[4] S. R. Buss and J.-S. Kim. Selectively damped least squares for inverse kinematics. *In Journal of Graphics Tools*, 10(3):37–49, 2005.

[5] M. P. Engell-Norregard, S. M. Niebe, and M. B. Bonding. Inverse kinematics with constraints. 2007.

[6] N. Joubert. Numerical methods for inverse kinematics.

[7] S. Rotenberg. Inverse kinematics. Lecture slides for CSE169 at UCSD, 2005.

[8] C. Sanderson. Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments. *NICTA Technical Report*, 2010.

[9] S. Tu. Inverse kinematics techniques. `http://inst.eecs.berkeley.edu/~cs184-cv/finalproj/ik-tech-report.pdf`.

[10] C. Welman. Inverse kinematics and geometric constraints for articulated figure manipulation. 1989.