Numerical Methods Lab 3

November 7, 2010

Objectives

In this lab you will explore the use of non-linear optimization on a problem that occurs in robotics/animation. In particular, you will evaluate Newton's and Broyden's method for non-linear systems of equations in inverse kinematic applications.

Submit with *astep -p lab4 -c c418 your_lab4_assignment.tar.gz*

Background

Given a kinematic chain (a set of joints that are joined by links), forward kinematics is the problem of determining the position of the joints (or only the end-effector) given the joint angles. This problem is a straighforward application of Euclidean transformations.

Inverse kinematics is the inverse problem: determine the joint orientations that satisfy a given a set of constraints (e.g., find the joint orientations that put the end-effector at a specific location). This problem is harder than forward kinematics; the analytical soltion is often complicated or doesn't exist so numerical solutions are required. Furthermore, it is possible that the solution is not unique. There may be no solutions or several solutions.

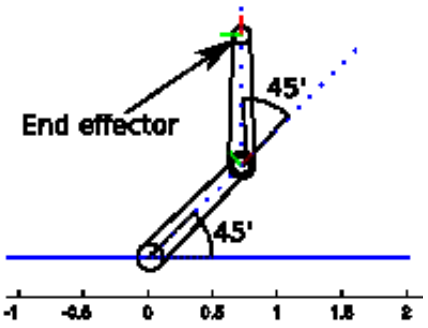

Figure 1: An example of a two link planar manipulator with $\mathbf{l} = [1; 1]$ and $\mathbf{theta} = [\pi/4; \pi/4]$.

Exercises

Part 1

In the first part of this lab you will focus on a 2D planar manipular two links and two rotational degrees of freedom (d.o.f.) (see Fig.). We will parameterize our manipulator with a vector of two link lengths ($\mathbf{l} = [l_1, l_2]^T$) and a vector of angles $\mathbf{theta} = [\theta_1, \theta_2]^T$.

The non-linear equations for positioning the end effector at a point $\mathbf{p} = [x, y]^T$ are

$$f_1(\theta_1, \theta_2) = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) - x \tag{1}$$

$$f_2(\theta_1, \theta_2) = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) - y \tag{2}$$

$$\tag{3}$$

1. (5 marks) Write a function `[pos,J]=evalRobot2D(l,theta)` that returns the position, **pos** (2×1 vector), of the end-effector and the Jacobian, **J**, given the rotation angles **theta** and the fixed link lengths **l**.

2. (5 marks) Write a function `J=fdJacob2D(l,theta,alpha)` that uses central differences to compute the Jacobian using step-size alpha. Evaluate several different values for alpha by comparing the finite difference Jacobian to the analytic jacobian for a variety of values for **theta**.

   - As an example the central difference formula for the first parameter (first column of J) would be (`evalRobot2D(l,theta+[alpha;0])-evalRobot2D(l,theta-[alpha;0]))/(2*alpha)`
   - Are the results close enough to be enough in the optimization?
   - Why would you use this finite-difference approximation instead of the analytic derivative?

3. (10 marks) Write a function `theta=invKin2D(l,theta0,pos,n,mode)` that returns the rotation angles in a 2×1 vector **theta** that put the end-effector at **pos**. Your method should terminate when the constraint is satisfied (use a threshold on the norm of the residual) or when the n iterations have occured. When `mode=0` you should use Newton's method, and `mode=1` you should use Broyden's method

   - For Broyden's method start with the Jacobian from your `evalRobot2D` function.

4. (5 marks) Use a script similar to the eval2D.m to the following to evaluate the effect of **mode**. You can use our plotRobot2D.m function to plot.

   - How does Broyden's compare to Newton?

Part 2

In the second half of this lab you will extend your 2D implementation to 3D. We will be animating two legs (independently) for a human.

Assume that we can parameterize each leg with four degrees of freedom. The location of the end-effector is then the homogenous zero transformed by the a sequence of transformations.

$$f(\theta) = \mathbf{M}_{hip}\mathbf{R}_z(\theta_3)\mathbf{R}_y(\theta_2)\mathbf{R}_x(\theta_1)\mathbf{M}_{knee}\mathbf{R}_x(\theta_4)\mathbf{M}_{foot}[0,0,0,1]^T$$

Where $\mathbf{M}_*$ is a rigid transformation encoding the rest position and orientation of a joint in the space of its parent, and may be different for each leg.

As input, all your functions should take $\mathbf{M} = \{\mathbf{M}_{hip}, \mathbf{M}_{knee}, \mathbf{M}_{foot}\}$ (a cell array) as its first parameter, followed by the joint angles $\mathbf{theta} = [\theta_1, \theta_2, \theta_3, \theta_4]'$.

5. (10 Marks) Write a function `[pos,J]=evalRobot3D(M,theta)` that returns the position and Jacobian for a set of rotation angles.

6. (5 Marks) Write a function `theta=invKin3D(M,theta,pos)` that returns the inverse kinematic solution for **pos**.

   - Should be straighforward to modify your Newton implementation from part 1.

7. (5 marks) Test your function on the data sequences provided in the data directory (walk1.mat,walk2.mat,walk3.mat,...,ju
   Each .mat file contains a **L** and **R** variables are $3 \times t$ matrices containing a time sequence of positional constraints for the left and right legs respectively.

   - Use the humanInterp(drad,angles) function to display the results. The function assumes you have stacked the left and right 4×1 vectors of angles on top of each other into one 8×1 vector (left joint angles as the first four elements). Use `robot3D('new')` to recreate the OpenGL window.

Appendix

Resources

Add c̃340/web_docs/labs/labAssign3/data to your path or copy to a local directory if you want to modify. The first part of the lab:

- eval2D.m: script for you to test your inverse kinematic function in 2D.
- plotRobot2D.m: plotRobot2D(ls,ts) plots the robot with link lengths **ls** and thetas **ts**

The second part:

- The data directory contains animation sequences (walk1,walk2,...).
- human_data.mat Contains the matrix definitions for the human and variables used for animation.
- robot3D.c: *The mex function will work on the lab computers, there is no need to recompile.* Source code for the mex file (to recompile for different matlab). Talk to Neil about recompiling for your own platform (requires POSIX OS so will require some work for Win32).
- humanInterp.m: humanInterp(drad,angles) returns a full animation vector that can be passed to robot3D to draw an interpolated pose for the upper body. `drad` is defined in human_data.mat.

Homogeneous matrix transforms:

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) & 0 \\ 0 & sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R}_y(\theta) = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\theta) & 0 & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R}_z(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 & 0 \\ sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{T}(x,y,z) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The regular rules of calculus apply to matrix products as long as you are aware that you cannot commute elements. The product rule for matrices:

$$\frac{\partial}{\partial\theta}A(\theta)B(\theta) = (\frac{\partial A(\theta)}{\partial\theta})B + A(\frac{\partial B(\theta)}{\partial\theta})$$

Notice that when B is not a function of θ you get:

$$\frac{\partial}{\partial\theta}A(\theta)B = (\frac{\partial A(\theta)}{\partial\theta})B$$