Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology

6.802/6.867 Robot Manipulation

Handout No. 11 Mar 10, 1995

Laboratory Assignment 2

Due March 21, 1995

This assignment is geared towards giving you some practice with inverse kinematics for cartesian interpolation. You should look at `/mit/6.802/labs/lab2/` for the code you will need for this assignment. You should copy all of the files into your own directory so you can edit them.

You'll need to modify the files `lab2.c`, `lab2invkin.c` and `lab2jacob.c`. Look for `/*** YOUR CODE HERE ***/` in six places. These files are linked in with a couple of other files that you do not need to modify. You also have to explain four lines tagged with `/*** EXPLAIN THIS LINE ***/`, two in `lab2.c` and two in `lab2utils.c`.

NB: Even if you cannot get your code to work, you can get credit by doing Problems 2 and 3 (using our executable). If you fail to get your code working, please make sure that you write down your understanding of what the code was supposed to do.

When you run the code, it asks you to specify a start frame and an end frame for the robot. It also asks for the method to be used (inverse kinematics or inverse jacobian) and the number of times to evaluate the inverse kinematics or the Jacobian along the path. It also asks you to specify the number of joint interpolation steps between these evaluations. The idea is that in between the relatively expensive kinematics or Jacobian computations, one does linear joint-space interpolation. This code does not use any max-velocity information, it just evenly divides the segment into a fixed number of steps. The code produces a number of plots indicating errors along the trajectory, the motion of each of the joint angles, etc. These plots can be viewed with gnuplot. Here's a typical interaction:

```
triad[79]cd ~/lab2
triad[80]lab2.demo
Getting D-H parameters from file /mit/6.802/include/puma.600.params.

First, you need to enter the position and orientation of the
endpoints of a linear trajectory to be followed.  The points
describe the configuration of the final frame.  To enter the
orientation, you will be asked for the gripper's x and y vectors.
All points should be expressed in the "base" frame.

Enter start position (x y z): 10 0 -5
Enter the base frame description of the final frame's x axis
(x y z): 1 0 0
Enter the base frame description of the final frame's y axis
(x y z): 0 1 0

Enter end position (x y z): 10 0 5
Enter the base frame description of the final frame's x axis
(x y z): 0 1 0
Enter the base frame description of the final frame's y axis
(x y z): 1 0 0

Select mode of operation: (1) inverse kinematics, (2) jacobian
Enter mode number: 2

Enter number of times to perform the jacobian computations: 10
Enter number of frames between jacobian computations
to use interpolated joint angles: 5


Type some characters and hit <return> to exit: blah

triad[81]gnuplot

G N U P L O T
unix version 3.5

...

Terminal type set to 'x11'
gnuplot> load "showall.gnuplot"
```

A few points to note about this:

- We have provided a (simple) working version for both Sparcs and on Decmips machines. To try things out, `cd` to your directory, and run `lab2.demo`; it should automatically find the executable in the 6.802 locker.

- When specifying axes, you do not have to specify unit vectors, the code will make them unit size, just get them to point where you want. The direction of the x axis is the most important, it is not changed. The y axis you specify does not have to be perpendicular to the x axis, it just has to be in the plane that you want to be the x-y plane.

- We provide a file called `sample.input` you can use like this:
  `athena$ lab2 < sample.input`
  when you don't want to type things in.

- The "end effector" referred to in the plots is NOT drawn – it's a fictional point at (1,1,1) in the final frame; this is used to gauge the effect of the rotation of the gripper.

- Reasonable coordinates for the workpace are within a radius of about 30.

- There are two output files produced, `lab2.desired` and `lab2.actual`. Ignore the joint angles in `lab2.desired`, since they are only one of several correct solutions; but the locations of the end-effector frame ought to be correct. NOTE: **`lab2.desired` is generated by your inverse kinematic routine, so if your routine is wrong, this is wrong, and thus all the error plots are also wrong!!** You should compare what your program produces on `sample.input` with what our version produces. There might be detailed differences of interpolation but yours and ours should be close.

- Here are some examples of useful gnuplot commands:

```
 << To load plots -- either individual graphs, or the collection >>
load "wrist-point.gnuplot"
load "showall.gnuplot"
 << To execute shell commands, eg ls >>
!ls
 << To print hardcopy plots -- they may be queued when gnuplot exits >>
set output "|lpr -Pprintername"
set terminal postscript
replot
 << To show plots onscreen (usually the default) >>
set terminal x11
replot
 << To exit >>
quit
```

Problem 1

There are specific places where you need to provide code (at `/*** YOUR CODE HERE ***/`):

1. In `lab2.c`, within `selectAngles()` you are encouraged (but not required) to provide a better approach to picking which of the inverse kinematic solutions (from the eight provided) to move to next.

2. In `lab2.c`, within `doInverseKin()` you need to provide code to do interpolation between two specified joint angle arrays. You need to produce a fixed number of joint angles sets between the start and the end. You should remember that angles wraparound at 2π and so $0 = 2\pi$. When interpolating, you should move in the shorter direction (increasing or decreasing angle) between the start and the end. The executable we provide you does not do this, so do not emulate its behavior in this regard.

3. In `lab2.c`, within `doJacobian()` you need to also provide code to do the interpolation. But, this requires a different strategy since `jacobianInvKin()` is computing a vector of angular velocities and not start and end joint angle vectors.

4. In `lab2invkin.c`, within `puma_invkin()` you need to fill in two places. The first place solves for joint angle 4 and the second fills in the two wrist solutions. This kinematics solution is modeled after the one in Craig's book. If you read that section, you'll have no difficulty doing this.

5. In `lab2jacob.c`, within `computeJacobian()`, you need to fill in the Jacobian using the cross-product method discussed in class. Note that this function has already computed the origin of the relevant frames in base coordinates. And so, you need to do very little...

6. In `lab2jacob.c`, within `computeJacobian()` you need to specify the desired cartesian velocity vector which gets passed to the linear equation solver (instead of actually computing an inverse Jacobian matrix). Most of what you need is already here, all you need to do is plug the pieces in for the definition of a cartesian velocity vector.

Problem 2

There are four places, two in `lab2.c` and two `lab2utils.c` where you will find lines labeled `/*** EXPLAIN THIS LINE ***/`. We want you to explain **why** these lines are there, not just echo the comment on each line (which generally explain **what** the lines does). We want to make sure you understand the role the lines play in the overall solution. For example, why is such and such a vector converted to base coordinates? What would happen if we didn't? Make sure that you include the line of code that you are explaining before the explanation so that we can tell which one you are referring to.

Problem 3

After everything works, execute the following paths:

1. The sample path we gave you.

2. Start at the zero joint angle initial position and move to the very edge of the workspace with the arm fully outstretched including the wrist at its singularity. You might want to use the forward kinematics function in `lab2invkin.c` to find the frames in terms of joint angles...

Hand in the plots of your interpolated paths with both inverse kinematics and jacobian interpolation and comment on what you found (especially any differences between the methods).

Please hand in the C code files that you modified and an executable. Have the file type of your executable indicate which architecture it was compiled for, e.g. sparc, decmips, etc. So, if you used a sparc, rename your executable `lab2.sparc`. Also, hand in an on-line text file (`README`) with the usual content (what is notable about your project, difficulties, successes, comments, etc.). Hand in hardcopies of the gnuplots, the explanations, and if you need/want to hand in pictures to go with your explanation, also hardcopies of these. Note that all hardcopy is due **in class** on the 21st.