Assignment 1 Report

Yiran Wang

Abstract

I implemented a 5-link kinematic chain “right” arm with an IK algorithm of iterative pseudo-inverse in a 2D plane, with setting the ratio of link lengths and the rotation constraints to each chain to make it look like a real right arm. The demo is implemented using OpenGL and its utility toolkit GLUT and a user interface library GLUI.

User Interface

Fig1. shows the user interface of my demo. The yellow star is the target point the right arm wants to grab. The arm is represented by 5 blue polygons with green envelope. Please use left mouse click to change the position of the target point, which will invoke the arm to start moving towards the target.

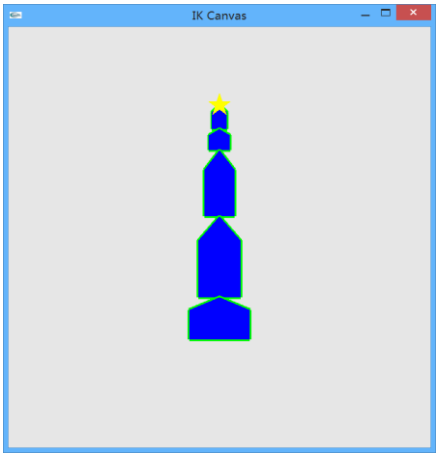


Fig 1. User interface of the demo.

The coordinate system of this canvas is a 2D plane whose x-axis ranges [-3, 4], y-axis also ranges [-3, 4]. From the collar bone, each chain of the arm has the length of {0.7, 13.5, 11, 3.5, 4}, with the center bottom of the collar bone initially located at (0.5, -1.2). I also set the target point initially located at the top center of the finger complex which is (0.5, 2.7) in the coordinate system.
To be clear, the **demo is with all the 3 steps finished and included**, which means once the user click some point, the arm will start “moving” for a while and terminate at the approximately closest point with the rotation constraints of each arm chain which will be discussed later in this report.

Step 1:

To make IK routine running, I used arrays of variables to denote the lengths, widths, positions, and rotation angles of each chain. Every time setting a new target point by clicking the left mouse button, we start the IK iteration using a while loop with at most 10,000 times of iterations to update

the rotation angles of each chain. Within each iteration, the following actions are performed:

1. Calculate the distance between the end effector, which is the top center of the fifth chain, and the target point, denoted by ***D***. If the distance is below the preset threshold value, end the iteration.
2. Else, if the times of iteration is still below 10,000, do the left:
3. Calculate the Jacobian matrix ***J*** analytically: for each arm chain, the entries of ***J*** is the cross product of (0, 0, -1) and (end effector position – center bottom position of that chain). I used (0, 0, -1) here because I set rotating clockwise as the positive.
4. Calculate the $\boldsymbol{J^+}$ Matrix using $J^+ = (J^tJ)^{-1}J^t$.
5. Calculate $\Delta\theta = J^+\Delta E$. I used 1/100*D as ΔE.
6. Update the rotation angles and positions of each arm chain.
7. Go back to 1.

Once this loop finished, the figures on the canvas will be redrawn to show how the current rotate and position of the arm chains.

Step 2:

To make the arm really to "move", I moved out the iteration out of the callback function of the mouse down action to the callback of idle. Instead of calculating out the result, I only reset the target point position and the iteration counter in the callback of mouse down action. And in the callback function of idle, one time of iteration will be down if the distance is still larger than the threshold value and the iteration counter also haven't met the limit. This change will lead to redraw the whole canvas every time we ran the IK iteration. Thus the figures will move much more slightly at one time so as to make the figure look like it is "moving".

What's more, in order to give the best answer we could for the cases that the distance threshold could not be met within 10,000 times of iterations, we will memorize the minimum distance and the responding rotate angles we met during all the iterations and use those values to redraw the arm chains when exiting the iteration loop.

Step 3:

The constraint angles we set to each arm chain were (all in degrees):

1. Collar bone: -15 to 15
2. Upper arm: -100 to 80
3. Forearm: -170 to 0
4. Hand: -90 to 85
5. Finger complex: -180 to 30

I chose these values as an experimental result I did with my own right arm.

To make the IK arm acting under these constraints, I simply did not allow the rotate angle arrays to get out of range every time it gets updated. Take the collar bone as the example, if $\theta + \Delta\theta$ is larger than 15 degrees, I will simply mark θ as 15 degrees, no bigger. The demo demonstrates this method works well.

Discussion

During my playing with the demo, when there is no rotation degree constraints, most of the positions around the canvas is easily reachable since the arm is able to freely rotate to any degree. The only exception would be the physically unreachable points due to the length constraint. The IK arm would drastically vibrates in that case and may end with a state that is fairly far from the target point. Like I mention in section “Step 2”, to make the performance better in this kind of cases, I memorized the best answer and position to display at the end of the iteration. This optimization usually works well as making the whole arm stretched straightly as a line (Fig 2.).

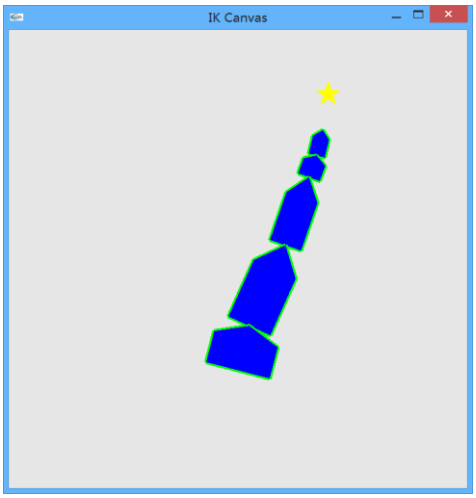
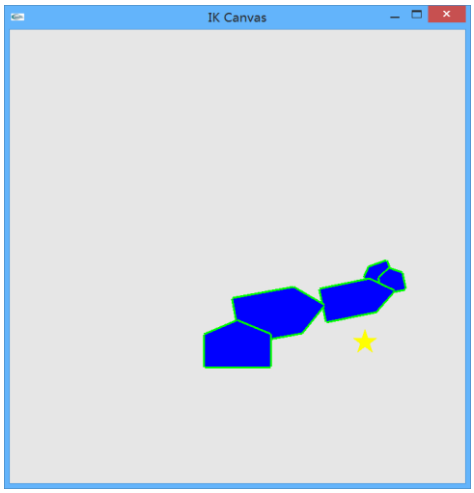
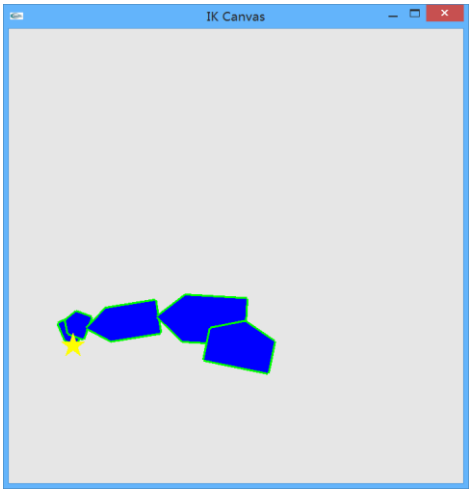


Fig 2. The case of unreachable point due to length.

After I added the rotation angle degree, points at the right half of the canvas become harder to reach, especially the point in the right-bottom quarter of the canvas (Fig 3.). This situation makes sense since the upper arm and the forearm are both unable to bend a lot towards right.



(a) The case of point of unreachable point at right



(b) The case of reachable mirror point

Fig. 3

For the large and small changes, I did not observe interestingly obvious difference on that issue, this may due to that I chose 1/100 times of the distance between end effector and target position as the ΔE which would be a relatively small value under both circumstances.

In a word, this demo basically shows how the pseudo-inverse algorithm work to reach the target point with only one end effector in the arm chain, and it act as expected like a real right arm in a 2D plane.