Robotics Mini-Proj 2

Yuxin Hu

September 2021

1 Cover Page

Name:	Yuxin Hu
Course Number:	CSCI-4480
Date:	Sep.27.2021

I, Yuxin Hu, certify that the following work is my own and completed in accordance with the academic integrity policy as described in the Robotics I course syllabus.

Link to Github: https://github.com/MoritomoNozomi/Robotics-I-Class-Project

2 Summary

In this project, I implemented the Forward Kinematics, and both the Geometry Inverse Kinematics and the Jacobian Inverse Kinematics functions, based on the example code provided in the lecture. The function of forward and inverse Kinematics were tested and proven correct. And also, the program managed to let the end effector of the robot follow a given path, with both methods.

3 Technical Content

Part 1

Description of the problem:

The arrangement of the path and the robot at zero configuration were given in the instruction. The task is to plot the robot, the path, and its outward normal, and also generate the path length array, λ.

Derivation of the solution:

The code for plotting the robot and path were already implemented in the code example. And the algorithm for the outward normal is:

1) Find the slope by $s_i = p_{i+1} - p_i$
2) Rotate the slope vector by 90 deg clockwise $s_i = R(\frac{pi}{2}) \cdot s_i$
3) Normalize the vector $s_i = norm(s_i)$
4) Plot it with quiver function in Matlab

Results based on simulation:

The result is shown in the figure below.

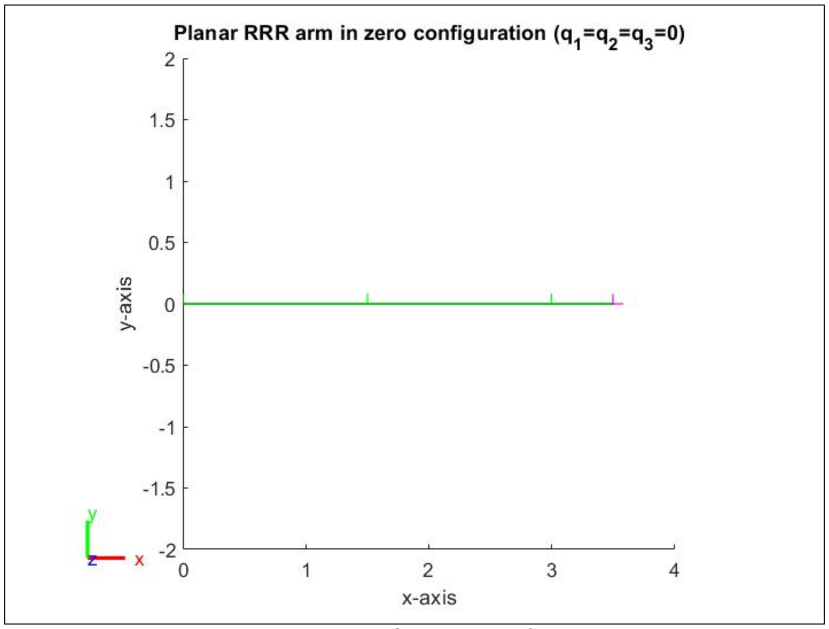


Figure 1: The Zero configuration of the RRR Robot

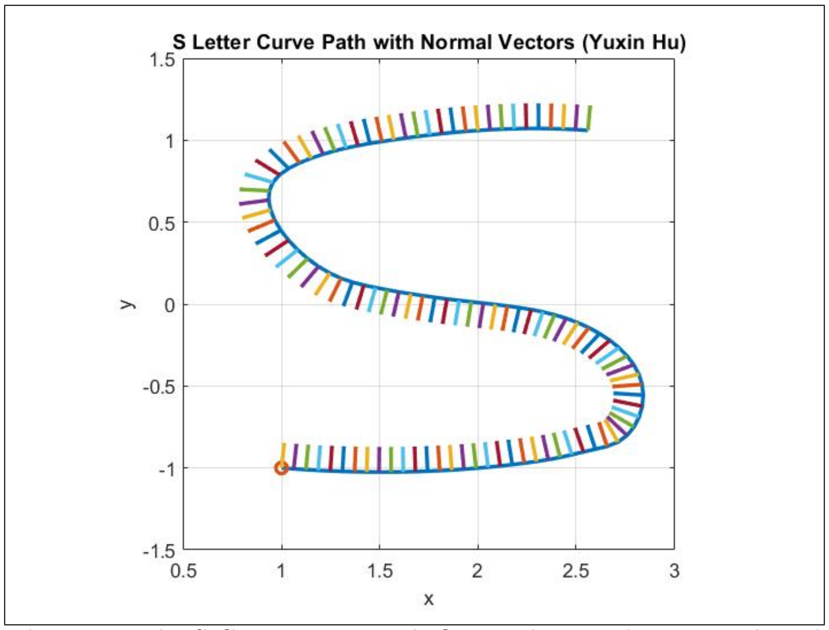


Figure 2: The S Curve Letter with Outward Normal Vectors Plotted

Part 2

Description of the problem:

Write down the forward kinematics of the three-link arm. Explain the algebraic, geo-metric, and iterative solution methods to inverse kinematics.

Derivation of the solution:

Forward Kinematics

$$\begin{aligned}
(\vec{p_{0T}})_0 &= (\vec{p_{01}})_0 + (\vec{p_{12}})_0 + (\vec{p_{23}})_0 + (\vec{p_{3T}})_0 \\
&= \vec{p_{01}} + R_{01}\vec{p_{12}} + R_{01}R_{12}\vec{p_{23}} + R_{01}R_{12}R_{23}\vec{p_{3T}} \\
&= 0 + \begin{bmatrix} cos(q1) & sin(q1) \\ -sin(q1) & cos(q1) \end{bmatrix} \begin{bmatrix} l1 \\ 0 \end{bmatrix} + \begin{bmatrix} cos(q1+q2) & sin(q1+q2) \\ -sin(q1+q2) & cos(q1+q2) \end{bmatrix} \begin{bmatrix} l2 \\ 0 \end{bmatrix} \\
&\quad + \begin{bmatrix} cos(q1+q2+q3) & sin(q1+q2+q3) \\ -sin(q1+q2+q3) & cos(q1+q2+q3) \end{bmatrix} \begin{bmatrix} l3 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} l1 \cdot c_1 + l2 \cdot c_{12} + l3 \cdot c_{123} \\ l1 \cdot s_1 + l2 \cdot s_{12} + l3 \cdot s_{123} \end{bmatrix} \\
q_T &= q_{01} + q_{12} + q_{23}
\end{aligned}$$

Algebraic Inverse Kinematics

Solve the equations:

$$\begin{aligned}
q_T &= q_1 + q_2 + q_3 \\
x_T &= l1 \cdot c_1 + l2 \cdot c_{12} + l3 \cdot c_{123} \\
y_T &= l1 \cdot s_1 + l2 \cdot s_{12} + l3 \cdot s_{123} \\
x_T - l_3 \cdot cos(q_T) &= l_1 \cdot c_1 + l_2 \cdot c_1 2 \\
y_T - l_3 \cdot sin(q_T) &= l_1 \cdot s_1 + l_2 \cdot s_1 2 \\
(x_T - l_3 \cdot cos(q_T))^2 &= {l_1}^2 \cdot c_1^2 + {l_2}^2 \cdot c_{12}^2 + l_1 \cdot l_2 \cdot c_1 \cdot c_{12} \\
(y_T - l_3 \cdot sin(q_T))^2 &= {l_1}^2 \cdot s_1^2 + {l_2}^2 \cdot s_{12}^2 + l_1 \cdot l_2 \cdot s_1 \cdot s_{12} \\
(x_T - l_3 \cdot cos(q_T))^2 + (x_T - l_3 \cdot cos(q_T))^2 &= l_1^2 + l_2^2 + l_1 \cdot l_2 \cdot cos(q_1 + q_2 - q_1) \\
&= l_1^2 + l_2^2 + 2 \cdot l_1 \cdot l_2 \cdot cos(q_2)
\end{aligned}$$

The solution is $q_2 = cos^{-1}(\frac{x_T - l_3 \cdot cos(q_T))^2 + (x_T - l_3 \cdot cos(q_T))^2 - l_1^2 - l_2^2}{2 \cdot l_1 \cdot l_2})$

Then find q_1 by taking q_2 into $(x_T - l_3 \cdot cos(q_T))^2 = {l_1}^2 \cdot c_1^2 + {l_2}^2 \cdot c_{12}^2 + l_1 \cdot l_2 \cdot c_1 \cdot c_{12}$

And find q_3 by $q_T = q_1 + q_2 + q_3$

Geometry Inverse Kinematics

Apply Law of Cosine:

$$
\begin{aligned}
|p_1| &= l1 \\
|p_{12}| &= |p_3 - p_0| \text{ (Distance From p1 to p3)} \\
d &= l2 \\
cos\phi &= \frac{|p_1|^2 + |p_{12}|^2 - d^2}{2|p_1||p_{12}|} \\
&\text{After obtaining } \phi, \\
q_1 &= q \pm \phi \\
&\text{(where } q \text{ is the angle between } \vec{p_{12}} \text{ and x-axis)} \\
p_2 &= \begin{bmatrix} l_1 \cdot cos(q \pm \phi) \\ l_1 \cdot sin(q \pm \phi) \end{bmatrix}
\end{aligned}
$$

Then it is possible to find q_2 by finding the angle between $\vec{p_{12}}$ and $\vec{p_{23}}$.

And then q_3 by finding the angle between $\vec{p_{23}}$ and $\vec{p_{3T}}$

Iterative Inverse Kinematics

The Iterative Kinematics solution does not require $P_{0T} = f(q)$, instead, it looks for a set of q that fulfills $min_q|p_{0T} - f(q)|$.

The algorithm should follow the one shown below:

1) Set up random 3-by-1 matrix q_0, where $-\pi \leq q_{0i} \leq \pi$ for all i in q_0. Note the initial configuration should not be co-linear, or $q_{0i} \neq 0$ for all $i \neq 1$.

2) Using the Forward Kinematics shown in the previous section, to find $J(q)$, the Jacobian Matrix of the RRR robot.

3) Find q_1 by $q_1 = q_0 - \alpha J^T(q_0)(f(q_0) - \chi_d)$, where $f(q_0)$ is $\begin{bmatrix} q_T \\ p_{0T} \end{bmatrix}$ found by the Forward Kinematics of the robot configuration, and $\chi_d = \begin{bmatrix} q_{Td} \\ p_{0T,d} \end{bmatrix}$ is the desired pose.

4) Repeat Step 3 for a reasonable amount of iterations, allowing the solution to converge.

Results based on simulation:

The results are shown in the previous section.

Part 3

Description of the problem:

Write the forward and inverse kinematics routine for the 3-link robot arm.

Derivation of the solution:

Forward Kinematics

In this mini-proj, I implemented the forward kinematics for both the RRR robot and n-link robot configuration.

For RRR robot, the end effector pose is found by $\begin{bmatrix} xT & = & l1 \cdot c1 + l2 \cdot c12 + l3 \cdot c123 \\ yT & = & l1 \cdot s1 + l2 \cdot s12 + l3 \cdot s123 \\ q_T & = & q_1 + q_2 + q_3 \end{bmatrix}$.

And for n-link robot, I used recursive forward kinematics defined as

$$\begin{bmatrix} p_{0T} & = & p_{01} + (\sum_{i=1}^{n} R(\sum_{j=1}^{i} q_j))p_{i,i+1} \\ q_t & = & \sum_{i=1}^{n} q_i \end{bmatrix}.$$

Though they are exactly the same algorithm, I implemented a separate one for RRR robot for clearness.

Geometry Inverse Kinematics

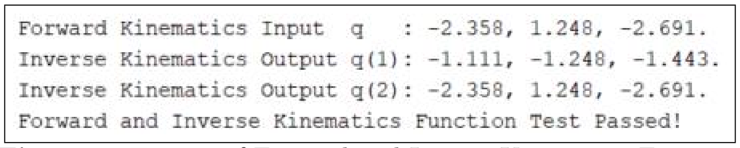
The Geometry Inverse Kinematics is only available for 3-link RRR robot, but not n-link since excessive links bring more Degree-Of-Freedom than needed. The algorithm is exactly the same with the one described in **Part 2**.

As described in the Mini Project instruction, the program generates three $q_i \in [-\pi, \pi]$, then the robot go through forward and inverse kinematics to verify if they are working correctly.

There is one noting point, the computer causes error when processing floating numbers, thus the == operator is not reliable. To compare two floating numbers, use $|p_1 - p_2| \le \theta$ instead, where θ is a relative small gap that could contain the error.

Results based on simulation:

The function worked as expected, and a demo of the forward and inverse kinematics testing is shown below.



```
Forward Kinematics Input  q   : -2.358, 1.248, -2.691.
Inverse Kinematics Output q(1): -1.111, -1.248, -1.443.
Inverse Kinematics Output q(2): -2.358, 1.248, -2.691.
Forward and Inverse Kinematics Function Test Passed!
```

Figure 3: Testing of Forward and Inverse Kinematics Function

Part 4

Description of the problem:

1) Find the desired pose for each point on the letter S path.
2) Find the robot motion that will trace the curve.
3) Visualize the robot motion.

Derivation of the solution:

Since my code was based on the class example code, I decided to use rigidbodytree to visualize the result. The basic algorithm follows:

1) Find the position and normal vector of each point on curve (**Part 1**)

2) For each point, use geometry inverse kinematics to solve for two sets of q (Elbow in and Elbow out)

3) For all q_{sol}, use forward kinematics to plot the robot. (Also record the movie!)

4) Differentiate the q_{sol} to find $\dot{q}_{sol}$ and find the max angular velocity $\dot{q}_{max}$

Results based on simulation:

Two sets of solutions are shown below:

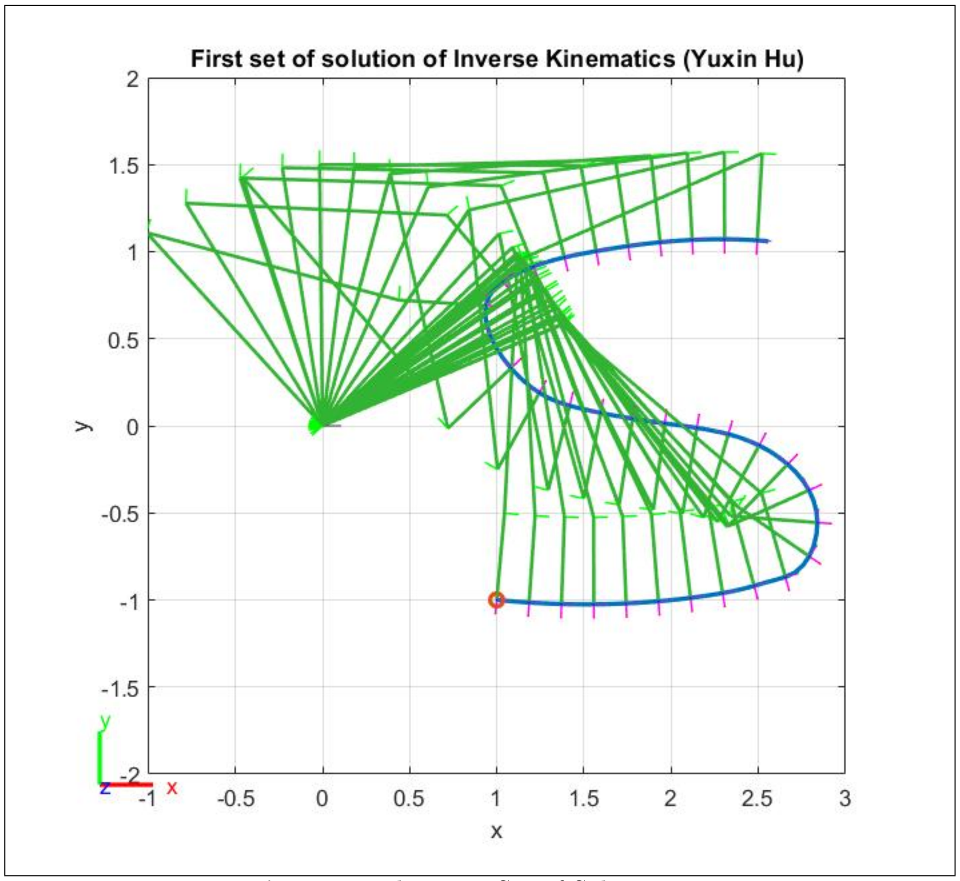


Figure 4: The First Set of Solutions

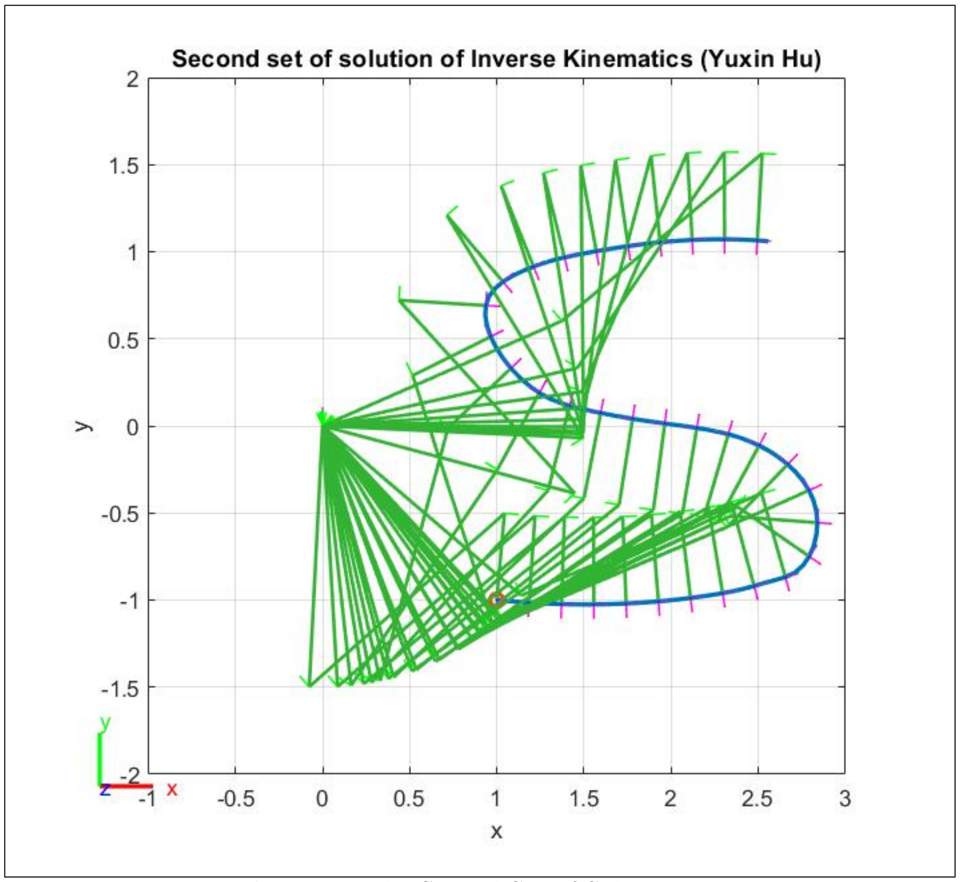
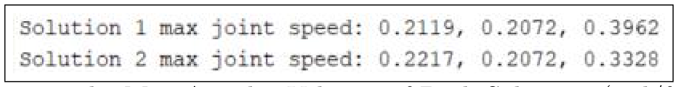


Figure 5: The Second Set of Solutions



```
Solution 1 max joint speed: 0.2119, 0.2072, 0.3962
Solution 2 max joint speed: 0.2217, 0.2072, 0.3328
```

Figure 6: The Max Angular Velocity of Both Solutions (rad/frame)

The Animated results are also available in the Github project folder, named as "$\underline{RRR\_demo1.avi}$" and "$\underline{RRR\_demo2.avi}$"

Part 5

Description of the problem:

Find a solution that gives largest constant path speed $\dot{\lambda}$

Derivation of the solution:

Since the problem was not quite clearly defined, I would interpret the problem as finding a max fps that makes $\dot{q}_{max} = 1rad/s$. The max joint speed from the previous part returns a max joint speed in $rad/frame$.

Thus the $\dot{\lambda}_{max} = fps_{max}(frame/s) = \frac{\dot{q}_{max}(rad/s)}{\dot{q}_{sol,max}(rad/frame)}$.

And the time to complete $t_{sol} = n_{curve}/\dot{\lambda}$.

Assume the distance between points in curve is constant, thus constant fps represents constant $\dot{\lambda}$.

Results based on simulation:

The angular speed is shown in **Figure 6**.

The max angular speed in solution 1 is $0.3962rad/frame$, and that in solution 2 is $0.3328frame/rad$. Thus the second solution is considered a better solution.

The max path speed $\dot{\lambda}_{max} = \frac{\dot{q}_{max}(rad/s)}{\dot{q}_{sol,max}(rad/frame)} = \frac{1(rad/s)}{0.3328(rad/frame)} = 3.005frame/s$.

Thus the total running time will be $t_{sol} = \frac{101frame}{3.005frame/s} = 33.6128s$

To research the relationship between the link length and running time, a few experiment is done. The original configuration is $[1.5, 1.5, 0.5]m$, and the new configuration and the result of max joint speed are shown belwo:

$$l = \begin{bmatrix} 1.5 & 1.5 & 0.1 \\ 1.5 & 1.5 & 0.55 \\ 1.5 & 2 & 0.5 \\ 1.5 & 1.48 & 0.5 \\ 2 & 1.5 & 0.5 \\ 1.48 & 1.5 & 0.5 \end{bmatrix} m \qquad \dot{q}_{sol,max} = \begin{bmatrix} 0.2892 \\ 0.3600 \\ 0.3537 \\ 0.3309 \\ 6.5218 \\ 0.3346 \end{bmatrix} rad/frame$$

The corresponding path speed and running time is:

$$\dot{\lambda}_{max} = \begin{bmatrix} 3.4578 \\ 2.7778 \\ 2.8273 \\ 3.0221 \\ 0.1533 \\ 2.9886 \end{bmatrix} frame/s \qquad t_{sol} = \begin{bmatrix} 29.2092 \\ 36.36 \\ 35.7237 \\ 33.4209 \\ 658.7018 \\ 33.7946 \end{bmatrix} s$$

The result of the tests show that shorter $l3$ and shorter $l2$ will have positive impact, while other modification will have negative impact. Especially longer $l1$ will greatly impact the running time since the joint speed is limited.

P.S.: In the tests, I found that the $l1$ and $l2$ cannot be shortened much since it would be out of reach and yields no solution.

Part 6

Description of the problem:

Implement one of the Jacobian-based robot motion control method.

Derivation of the solution:

The method I chose is the damped least square method. Compared to other methods, the damped least square method could help avoid the oscillation when the error between the end effector and the desired pose is relatively small. The algorithm is already implemented in the class example, so my program is based on the sample program.

Results based on simulation:

The result of the Jacobian Inverse Kinematics on the RRR robot is shown below.

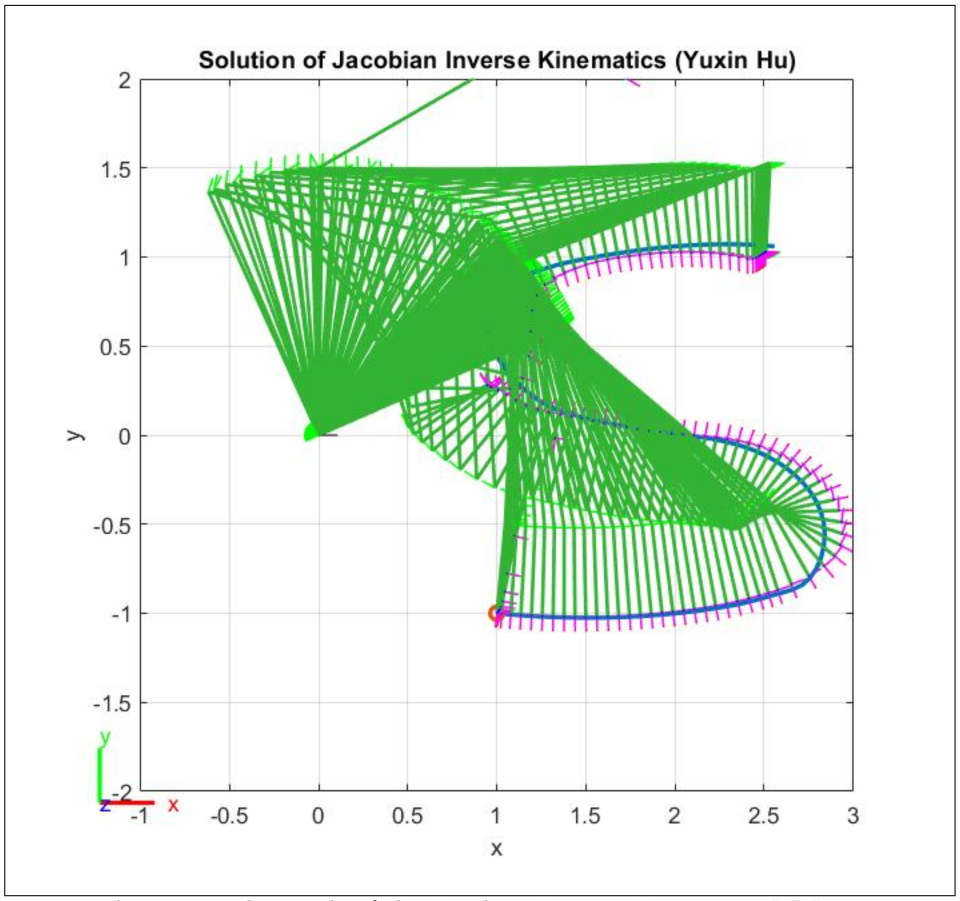


Figure 7: The result of the Jacobian Inverse Kinematics RRR

Since the least square method was used, the end effector pose did not follow the desired pose. The performance is very smooth on the relatively straight segments, but very poor in those curves. On both U-turn the robot failed to trace the path.

The max joint speed is found to be $[0.0243, 0.1000, 0.0414]$ rad/frame.

Using the equations found in **Part 5**, the max path speed
$\dot{\lambda}_{max} = \frac{\dot{q}_{max}(rad/s)}{\dot{q}_{sol,max}(rad/frame)} = \frac{1(rad/s)}{0.1(rad/frame)} = 10 frame/s.$

And the total running time will be $t_{sol} = \frac{101 frame}{10 frame/s} = 10.1s$. Compared to that of the Geometry Inverse Kinematics, which is $33.6128s$, the Jacobian method is almost 3 times faster. Thus it would be a good choice to use Jacobian method rather than Geometry method in the circumstance that speed is more important than pose accuracy.

The Animated result is also available in the Github project folder, named as "*RRR_Jacob_demo.avi*".

Part 7

Description of the problem:

(I am an undergraduate student, but can I get **Extra Credit** for doing the graduate section?)

Implement a 10-link planar arm with each link length 0.35m, that tracks the given S curve.

Derivation of the solution:

Based on the recursive forward kinematics found in **Part 3**, it is easy to extend the program to n-link planar arm robots, for any $n \in \mathbb{N}$.

Results based on simulation:

Based on the class example of n-link robots, the program managed to simulate a 10-link robot and a 100-link robot.
The result of the 10-link robot is shown below:

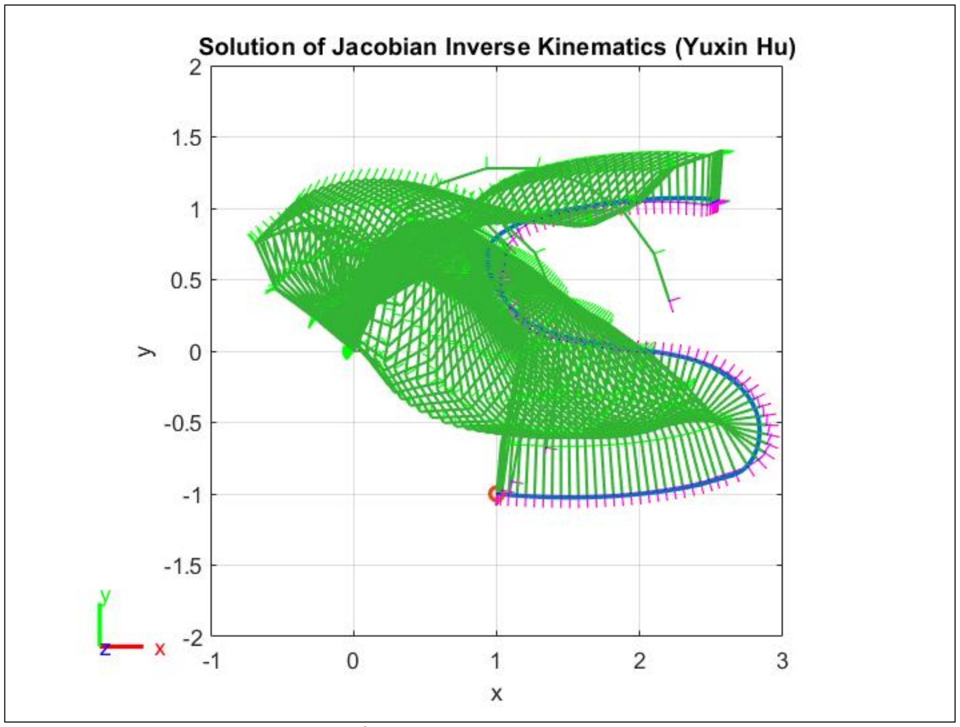


Figure 8: Result of 10-link Jacobian Inverse Kinematics

It is clear to see that the 10-link robot performs way better than the RRR robot in **Part 6**. Though it still has some offset between the end effector and the desired curve, the error is relatively smaller. The max joint speed is reported to be $0.0243 rad/frame$.

$\dot{\lambda}_{max} = \frac{\dot{q}_{max}(rad/s)}{\dot{q}_{sol,max}(rad/frame)} = \frac{1(rad/s)}{0.0243(rad/frame)} = 41.15 frame/s.$

And the total running time will be $t_{sol} = \frac{101 frame}{10 frame/s} = 2.4543s$.
Almost 4 times faster than RRR robot!

I made another simulation of 100-link robots, and the result is shown below:

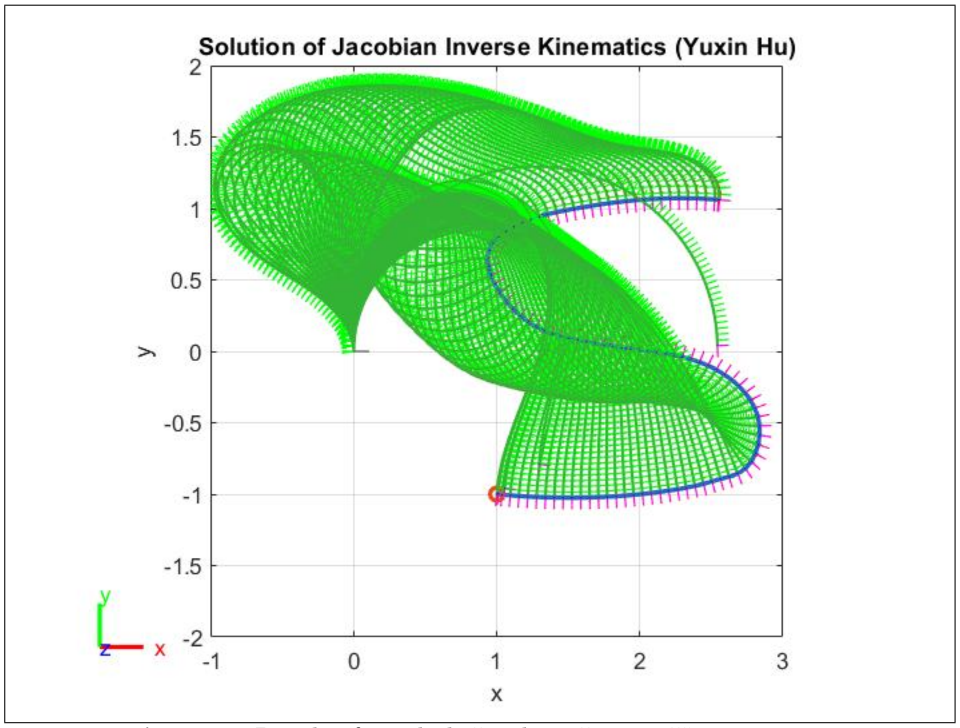


Figure 9: Result of 100-link Jacobian Inverse Kinematics

The simulation result is even better. The pose of the end effector perfectly matched the S letter curve. The max joint speed is reported to be $0.0078 rad/frame$.

$\dot{\lambda}_{max} = \frac{\dot{q}_{max}(rad/s)}{\dot{q}_{sol,max}(rad/frame)} = \frac{1(rad/s)}{0.0078(rad/frame)} = 128.205 frame/s.$

And the total running time will be $t_{sol} = \frac{101 frame}{128.205 frame/s} = 0.7878 s$.

The result is about another 3 times faster than the 10-link robot! Compared to the original RRR arm following geometry inverse kinematics, the 100-link robot is about 42.6 times faster to accomplish the task.

The Animated results are also available in the Github project folder, named as "*10link_Jacob_demo.avi*" and "*100link_Jacob_demo.avi*"

4 Conclusion

In this Mini-Project, I learned the Forward Kinematics to model the robot and three different way of the Inverse Kinematics to control the robot motion. In the Project, I had a better understanding of the meaning of a Jacobian matrix, and how it could serve to find an optimal path to the desired pose. The program worked as expected, which costed me days and nights debugging it.

There is still something that could be improved. In the Jacobian Inverse Kinematics, it is obvious that the RRR robot could not follow the track under constant path speed. If some algorithm of non-constant path speed was introduced, the robot could perform better in speed and accuracy. The robot will perform much better at corners if more time was given. Just like racing cars, that goes at 100% speed on straight lines, while decelerates at corners.