Solving the Inverse Kinematics (IK) Problem using Numerical Methods

Dhyeykumar Thummar[2], Dwip Dalal[3], Pushpendra Pratap Singh[1], R Yeeshu Dhurandhar[1], Shruhrid Banthia[1]

[1] *Electrical Engineering, IIT Gandhinagar*
[2] *Computer Science and Engineering, IIT Gandhinagar*
[3] *Mechanical Engineering, IIT Gandhinagar*

Submitted: 24.04.2022

Abstract
Inverse Kinematics (IK) problem is defined as the problem of determining a set of appropriate joint configurations for which the end effectors move to desired positions as smoothly, rapidly, and as accurately as possible. A few of the iterative methods that solve tries to solve the problem have been described here. Further, we have prepared a MATLAB code that uses one of the methods. We have also simulated the working of the methods in Unity Engine to help better visualize it.

Key words: Inverse Kinematics – Numerical Methods – Jacobian Transpose Method – FABRIK method

1 Introduction

A rigid multibody system is made up of a collection of rigid objects called links that are connected by joints. Inverse kinematics (IK) is commonly used to control the movement of a stiff multibody. It is assumed that specific points on the links, referred to as "end effectors," are allocated "target positions" for IK.

To address the IK problem, we must find joint angle values that will result in a multibody configuration that sets each end effector in its desired location. We want the end effectors to track the target positions and perform exceptionally well even when the target positions are in unreachable positions.

We consider only first-order methods. We assume a multibody has numerous end effectors and various target locations, all of which are provided in real-time online. We want to update the multibody configuration so that the end effectors can dynamically monitor the target positions. We will only look at the "pure" IK issue, which doesn't have any joint limitations or self-collisions. This problem finds its application in robotics, computer animation, ergonomics, and gaming. In computer graphics, articulated figures are a convenient model for humans, animals, or other virtual creatures from films and video games. Most virtual character models are very complex; they are made up of many joints giving a system with many degrees of freedom; thus, it is often challenging to produce a realistic character animation.

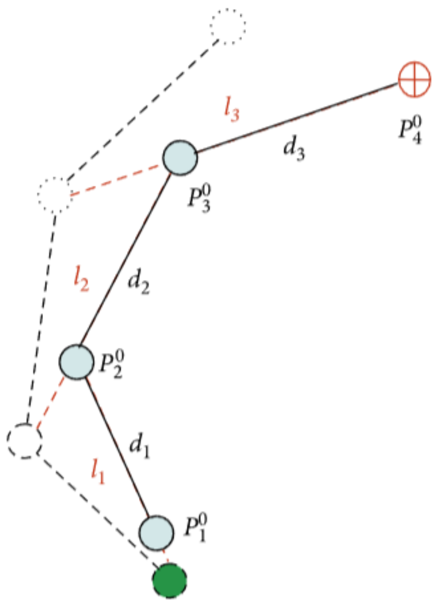


Figure 1. A multi-joint arm.

2 Problem formulation

A multibody is modeled with a set of links connected by joints. Certain points on the links are identified as end effectors.

Subscript 1, 2, . . . , i, . . . , k corresponds to end effectors. k = number of end effectors.
Subscript 1, 2, . . . , j, . . . , n are joints. n = number of joints.

$$\vec{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$

where, $\theta_j = joint angles$
The complete configuration of the multi-body is specified by $\theta_1, ..., \theta_n$.
$s_1, ..., s_k$ are positions of end effectors.

$s_i = f(jointangles)$

$$\vec{s} = \vec{s}(\vec{\theta})$$
$$s_i = s_i(\vec{\theta})$$

$$\vec{s} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_k \end{bmatrix}_{k\times 3}$$

This may be represented as a column vector with $m = 3k$ scalar elements or k scalar entries from R^3.
The multibody will be controlled by specifying target positions for the end effectors.
let t_i =target position of i^{th} end effector.

$$t = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_k \end{bmatrix}$$

where t_i is the target position of the i^{th} end effector.

$$\vec{t_i} = \vec{s_i}(\vec{\theta}) \qquad (1)$$

$\vec{e} = \vec{t} - \vec{s}$ $e_i = t_i - s_i$
where, $e_i = t_i s_i$, the desired change in position of the i^{th} end effector.
Unfortunately, a solution to (1) may not always be available, and there may not be a single (best) solution. Even in well-behaved cases, the solution may not have a closed-form equation.

3 Preliminary approach

Iterative methods can be used to approximate a satisfactory solution. The Jacobian matrix is used to linearly approximate the functions. The Jacobian matrix J is a function of the values and is defined by

$J(\vec{\theta}) = \left(\frac{\partial S_i}{\partial \theta_j}\right)_{i,j}$

Dimensions of the j can be viewed either as a $k \times n$ matrix whose entries are vectors from R^3 , or as $m \times n$ matrix with scalar entries (with $m = 3k$).

The following is the fundamental forward dynamics equation that represents the velocities of the end effectors (using dot notation for first derivatives):

$$\dot{\vec{s}} = J(\vec{\theta})\dot{\vec{\theta}} \qquad (2)$$

The Jacobian provides us an iterative method for solving equation (1). Suppose we have current values for $\theta, \vec{s}$ and $\vec{t}$. From these, the Jacobian $J = J(\theta)$ is computed.

We then seek an update value for the purpose of incrementing the joint angles by :

$$\vec{\theta} := \vec{\theta} + \Delta\vec{\theta} \qquad (3)$$

The change in end effector locations produced by this change in joint angles may be calculated using (2) as follows:

$$\Delta\vec{s} \approx J \cdot \Delta\vec{\theta} \qquad (4)$$

Now, the main task is to find the value of for updating the joint angles. One possible approach is:

$\vec{e} = \Delta\vec{s} = J\Delta\vec{\theta} \Rightarrow \Delta\vec{\theta} = J^{-1} \cdot \vec{e}$

In most cases, this equation cannot be solved uniquely. The Jacobian J may or may not be a square matrix therefore it will be non-invertible. Even if it is invertible, its performance will be low when it is poorly invertible.

3.1 Calculating the Jacobian

If the jth joint is a rotational joint with a single degree of freedom, the joint angle is a single scalar j. Let,

p_j = position of the joint v_j = a unit vector pointing along the current axis of rotation for the joint

The entries in the Jacobian matrix can be found using given three cases: 1. If the i^{th} end effector is affected by the joint

$\frac{\partial \vec{s}i}{\partial \theta j} = \vec{V}j \times (\vec{s}i - \vec{p}_j)$

2. If the i^{th} end effector is not affected by the j^{th} joint

$\frac{\partial \vec{s}i}{\partial \theta j} = 0$

3. If the j^{th} joint is translational.

Then if the i^{th} end effector is affected by the j^{th} joint, we have

$\frac{\partial \vec{s}i}{\partial \theta j} = \vec{V}j$

4 Jacobian transpose approach

The basic idea: use the transpose of J instead of the inverse of J.

$\Delta\theta = \alpha J^T \vec{e}$

for some appropriate scalar .

Now, the transpose of the Jacobian is not the same as the inverse; however, it is possible to justify the use of the transpose in terms of virtual forces.

How to choose the value of :

One reasonable way to choose the value of is to try to minimize the new value of the error vector $\vec{e}$ after the update. For this, we assume that the change in end effector position will be exactly $\alpha J J^T \vec{e}$, and choose so as to make this value as close as possible to $\vec{e}$.

$\alpha = \frac{\langle \vec{e}, JJ^T\vec{e} \rangle}{\langle JJ^T\vec{e}, JJ^T\vec{e} \rangle}$

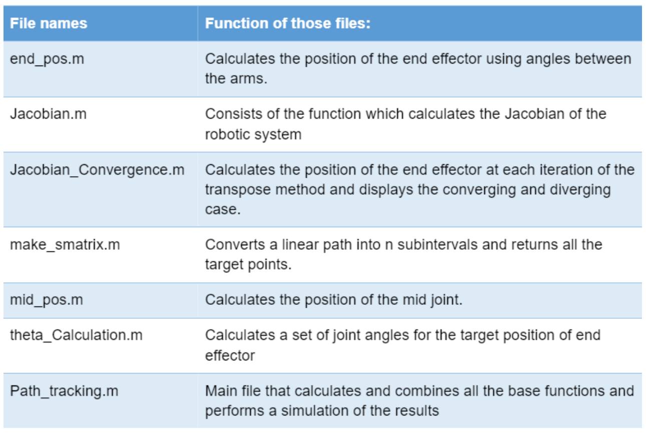
5 Implementation in MATLAB

We have implemented the Jacobian Transpose method in Matlab. This simulation in Matlab attempts to test this method on a virtual three degree of freedom arm with one fixed end, one mid joint, and one end effector. The goal is to simulate the movement of the arm given a start and endpoint. We have used a staggering method to make the arm reach the destination.

In the Path_tracking.m file, we have implemented the simulation. We have divided the linear path into chunks of equal length. Thus we perform interactions of the Jacobian Transpose method for each of these chunks. This results in higher performance when compared to the case without any chunks.

The path taken has been shown using an approximation in the above visualization. We get a better understanding of the Jacobian convergence at each chunk by running the Jacobian_Convergence.m file.

Link to the MATLAB Implementation: github.com/sbthycode/Using-Jacobian-Transpose-method-for-Pathing



File names	Function of those files:
end_pos.m	Calculates the position of the end effector using angles between the arms.
Jacobian.m	Consists of the function which calculates the Jacobian of the robotic system
Jacobian_Convergence.m	Calculates the position of the end effector at each iteration of the transpose method and displays the converging and diverging case.
make_smatrix.m	Converts a linear path into n subintervals and returns all the target points.
mid_pos.m	Calculates the position of the mid joint.
theta_Calculation.m	Calculates a set of joint angles for the target position of end effector
Path_tracking.m	Main file that calculates and combines all the base functions and performs a simulation of the results

Figure 2.

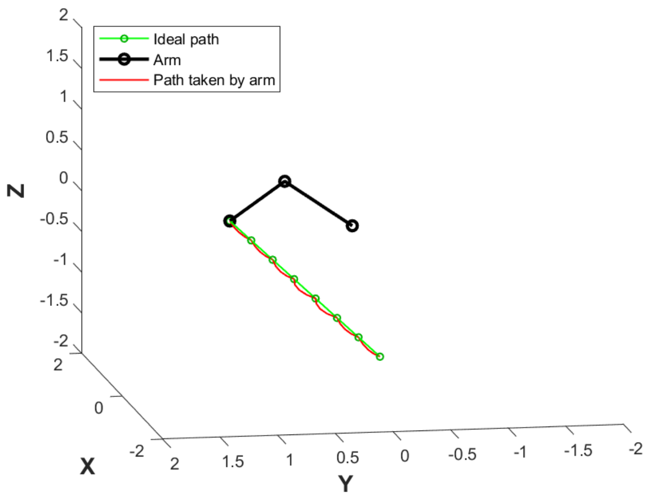


Figure 3. Output from Jacobian$_Convergence.m$

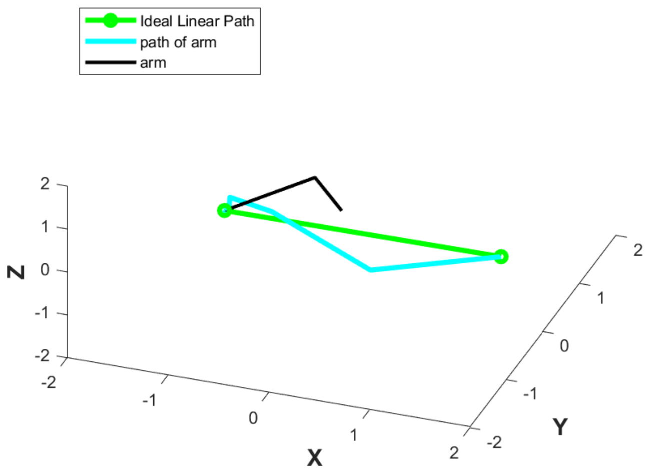
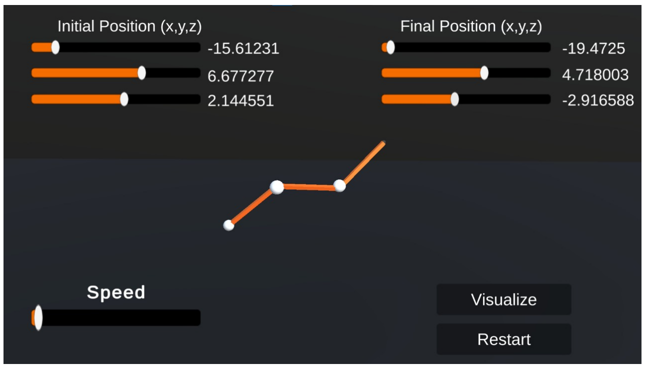


Figure 4. Output from Path$_tracking.m$



Initial Position (x,y,z)
-15.61231
6.677277
2.144551

Final Position (x,y,z)
-19.4725
4.718003
-2.916588

Figure 5. Implementation in Unity.

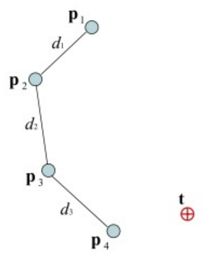


Figure 6. Initial

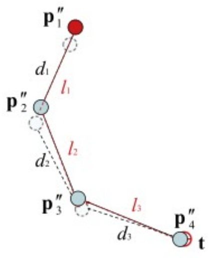


Figure 7. Final

6 Implementation in unity

We have used the FABRIK algorithm to solve the Inverse Kinematics problem in Unity. FABRIK takes it name from Forward and Backward Reaching Inverse Kinematics as it uses forward and backward iterative approach to solve the inverse kinematics problem. Instead of using angle rotation meaning that it has to calculate the rotational angle to matrice, FABRIK treats finding each joint location as a problem of finding a point on a line. Hence, time and computational cost can be saved. In the Jacobian methods, the computational cost is very high. The naive Jacobian approach also suffers from singularity problem and may produce unrealistic movements. FABRIK produces more realistic results than any other method. It is also significantly faster (1000 times) than Jacobian method and thus take less iterations. We have implemented a user friendly interface where one can set the initial coordinates and the final coordinates of the path. Afterwards, user can click on the 'Visualize' button to observe the motion. There is also an option to adjust speed incase the motion feels slow.

Link to the Unity Implementation: pushpendra.itch.io/robot-arm

7 Future Work

As mentioned above we tried different methods for solving the Inverse Kinematics Problem. Apart from these approaches there are various other methods as well : cyclic coordinate descent methods, pseudoinverse methods, Jacobian transpose methods, the Levenberg-Marquardt damped least-squares methods, quasi-Newton and conjugate gradient methods, and neural net and artificial intelligence methods.

Given the chance we would like to try to implement these methods in Unity as well as MATLAB.

8 Acknowledgement

We, as a team, would like to extend our gratitude to all the three instructors of MA202 who gave us this opportunity to work on such an exciting topic. We would also like to thank Prof Tanya Kaushal Srivastava, Prof Satyajit Pramanik, and Prof Uddipta Ghosh for providing guidance and assistance whenever needed.

References

[1] ermudez, Luis. "Overview of Jacobian IK. What Is the Math behind Jacobian IK... | by Luis Bermudez | Unity3DAnimation | Medium." Medium, medium.com, 23 Dec. 2018, https://medium.com/unity3danimation/overview-of-jacobian-ik-a33939639ab2.

[2] . Baerlocher and R. Boilic, Inverse Kinematics Techniques for the Interactive Posture Control of Articulated Figures, PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 2001.

[3] . Baerlocher and R. Boulic, Task-priority formulations for the kinematics control of highly redundant articulated structures, in Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 1, 1998.

[4] . Baillieul, Kinematic programming alternatives for redundant manipulators, in Proc. IEEE International Conference on Robotics and Automation, 1985, pp. 722–728.

[5] . K. Chan and P. D. Lawrence, General inverse kinematics with the error-damped pseudoinverse, in Proc. IEEE International Conference on Robotics and Automation, 1988, pp. 834–839.

[6] . Chiaverini, B. Siciliano, and O. Egeland, Review of damped least-squares inverse kinematics with experiments on an industrial robot manipulator, IEEE Transactions on Control Systems Technology, 2 (1994), pp. 123–134.