Neura Robotics Test Task

Shubham Kamble

May 29, 2024

Abstract: This documentation describes the solution of the test task of developing an Iterative Inverse Kinematics algorithm for the Maira robot. The algorithm builds upon the only given Denavit-Hartenberg (DH) parameters to find the nearest possible solution by calculating orientation errors with quaternions, and handling redundancy.

Contents

1 Task Description

1.1 Problem Statement

Given the DH parameters in Table 1, develop an iterative inverse kinematics algorithm that fulfills the following requirements:

- Calculates the forward kinematics using DH parameters.
- Calculates orientation error using quaternions.
- Satisfies joint limits ($\pm$ 180 degrees for all axes, except A2: $\pm$ 120 degrees, and A4: $\pm$ 150 degrees).
- Calculates the Jacobian Matrix.
- Handles redundancy with 2 different methods.
- Calculates the condition number of Jacobian matrix.
- Estimates the success rate of convergence.

Joint i	θ_i (rad)	d_i (m)	α_i (rad)	a_i (m)
1	θ_1	0.438	$-\pi/2$	0
2	θ_2	0	$\pi/2$	0
3	θ_3	0.7	$-\pi/2$	0
4	θ_4	0	$\pi/2$	0
5	θ_5	0.7	$-\pi/2$	0
6	θ_6	0	$\pi/2$	0
7	θ_7	0.115	0	0

Table 1: Denavit-Hartenberg (DH) Parameters of Maira robot

2 Solution

2.1 Kinematics

Kinematics is a fundamental aspect of robotics that deals with the motion of robots without considering the forces that cause this motion. It involves the study of the geometry of motion and the relationships between various parameters such as position, velocity, and acceleration. In robotics, kinematics is crucial for understanding how robots move and interact with their environment, enabling precise control and manipulation.

There are two main types of kinematics in robotics: forward kinematics and inverse kinematics. Forward kinematics focuses on determining the position and orientation of the robot's end-effector (e.g., a robotic hand) based on given joint parameters such as angles or displacements. Inverse kinematics, on the other hand, involves calculating the necessary joint parameters to achieve a specific position and orientation of the end-effector. This is more complex and is critical for applications requiring the robot to navigate through specific paths or handle objects with precision.

Understanding kinematics is vital for designing, programming, and operating robots in various fields. It provides the mathematical foundation for developing control algorithms that enable robots to perform complex tasks accurately and efficiently.

2.2 Quaternions

Quaternions are a number system that extends complex numbers and are used in robotics for representing and computing rotations in three dimensions. A quaternion is typically expressed as:

$$q = w + xi + yj + zk$$

where w, x, y, z are real numbers, and i, j, k are the fundamental quaternion units. Quaternions can also be represented in vector form as:

$$q = (w, \vec{v}) = (w, (x, y, z))$$

where w is the scalar part and $\vec{v} = (x, y, z)$ is the vector part.

One of the key advantages of using quaternions over other representations like Euler angles or Angle-axis representations is that they avoid gimbal lock and provide a more compact and computationally efficient means of representing rotations. To interpret a quaternion, we can understand it as encoding a rotation around an axis. The unit quaternion, which is used for rotations, has a norm (or magnitude) of 1. A unit quaternion q can be written as:

$$q = \cos\left(\frac{\theta}{2}\right) + \sin\left(\frac{\theta}{2}\right)(xi + yj + zk)$$

where θ is the rotation angle, and (x, y, z) is the unit vector along the axis of rotation.

2.3 Transformation using DH Parameters

In robotics, the Denavit-Hartenberg (DH) convention is a systematic method to define the relative transformations between consecutive coordinate frames attached to a robot's joints. This approach simplifies the calculation of the forward kinematics, which involves finding the position and orientation of the end-effector given the joint parameters.

Each joint of a robot is associated with a coordinate frame, and the transformation between two successive frames i and $i-1$ can be described using four DH parameters:

- θ_i: The joint angle, which is the rotation around the z_{i-1} axis.
- d_i: The link offset, which is the translation along the z_{i-1} axis.
- a_i: The link length, which is the translation along the x_i axis.
- α_i: The link twist, which is the rotation around the x_i axis.

The transformation matrix $\mathbf{A}_i$ between two successive frames can be written as:

$$\mathbf{A}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

By multiplying these transformation matrices sequentially from the base to the end-effector, we can find the overall transformation matrix $\mathbf{T}$ that describes the position and orientation of the end-effector relative to the base frame:

$$\mathbf{T} = \mathbf{A}_1 \mathbf{A}_2 \mathbf{A}_3 \cdots \mathbf{A}_n$$

This transformation matrix $\mathbf{T}$ encapsulates the combined rotations and translations necessary to move from the base frame to the end-effector frame, allowing for precise control and positioning of the robot.

2.4 Forward Kinematics

Forward kinematics in robotics involves computing the position and orientation of the robot's end-effector in the operational space, given the joint parameters. Once we have the overall transformation matrix $\mathbf{T}$ from the base frame to the end-effector frame, we can extract both the position and orientation of the end-effector.

The transformation matrix $\mathbf{T}$ is typically of the form:

$$\mathbf{T} = \begin{bmatrix} R & \mathbf{p} \\ 0 & 1 \end{bmatrix}$$

where R is the 3×3 rotation matrix and $\mathbf{p} = [x, y, z]^T$ is the position vector of the end-effector.

The position of the end-effector in the operational space is directly given by the translation part of the transformation matrix:

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

The orientation of the end-effector can be represented using quaternions, which provide a compact and efficient way to handle 3D rotations. To convert the rotation matrix R into a quaternion $q = (w, x, y, z)$, we use the following relations:

$$w = \frac{1}{2}\sqrt{1 + r_{11} + r_{22} + r_{33}}$$

$$x = \frac{sgn(r_{23} - r_{32})}{2}\sqrt{r_{11} - r_{22} - r_{33} + 1}$$

$$y = \frac{sgn(r_{13} - r_{31})}{2}\sqrt{r_{22} - r_{33} - r_{11} + 1}$$

$$z = \frac{sgn(r_{21} - r_{21})}{2}\sqrt{r_{33} - r_{22} - r_{11} + 1}$$

where r_{ij} are the elements of the rotation matrix R.

Thus, the quaternion q representing the orientation of the end-effector is:

$$q = (w, x, y, z)$$

By extracting the position vector $\mathbf{p}$ and converting the rotation matrix R to the quaternion q, we can fully describe the pose of the end-effector in the operational space. This information is crucial for tasks that require precise positioning and orientation, such as robotic manipulation, and navigation.

2.5 Inverse Kinematics

Inverse kinematics (IK) in robotics involves calculating the joint parameters required to position the end-effector at a desired target position and orientation. Given a target pose described by a position vector and a quaternion for orientation, the goal is to determine the joint angles that achieve this pose. Inverse kinematics can be performed in two ways: analytical or numerical. Here, we are going to devise an iterative numerical solver, and for that we need the Jacobian matrix.

The Jacobian matrix $\mathbf{J}$ relates the joint velocities to the end-effector velocities. The Jacobian matrix can be calculated differently depending on whether the joint is prismatic or revolute.

For a prismatic joint i, the Jacobian can be computed as:

$$J_i = \begin{bmatrix} J_{GPi} \\ J_{GOi} \end{bmatrix} = \begin{bmatrix} {}^0\mathbf{e}_{z,i-1,O} \\ \mathbf{0} \end{bmatrix}$$

where

- ${}^0\mathbf{e}_{z,i-1,O}$ is the unit vector along the axis of the prismatic joint in the base frame.
- $\mathbf{0}$ is a zero vector, indicating no rotational component for the prismatic joint.

For a revolute joint i, the Jacobian is given by:

$$J_i = \begin{bmatrix} J_{GPi} \\ J_{GOi} \end{bmatrix} = \begin{bmatrix} {}^0\mathbf{e}_{z,i-1,O} \times ({}^0\mathbf{r}_{E,O} - {}^0\mathbf{r}_{i-1,O}) \\ {}^0\mathbf{e}_{z,i-1,O} \end{bmatrix}$$

where

- ${}^0\mathbf{e}_{z,i-1,O}$ is the unit vector along the axis of the $(i-1)$-th revolute joint in the base frame.
- ${}^0\mathbf{r}_{E,O}$ is the position vector of the end-effector in the base frame.
- ${}^0\mathbf{r}_{i-1,O}$ is the position vector of the $(i-1)$-th joint in the base frame.
- $\times$ denotes the cross product, which computes the linear velocity component due to the rotation of the joint.

Since we have adopted numerical methods, the two widely used ones are:

- Newton's method
- Gradient Descent method

2.5.1 Newton's Method

Newton's method is an iterative technique used to find the roots of a function. In the context of inverse kinematics, it aims to minimize the error between the current end-effector pose and the target pose.

Given a function $\mathbf{f}(\mathbf{q})$ representing the forward kinematics, and **err** representing difference between the current and desired end-effector positions and orientations, Newton's method updates the joint configuration $\mathbf{q}$ as follows:

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \mathbf{J}^{-1}(\mathbf{q}_k)\mathbf{err}_k$$

Here

- $\mathbf{q}_k$ is the joint configuration at iteration k.
- $\mathbf{J}(\mathbf{q}_k)$ is the Jacobian matrix at $\mathbf{q}_k$.
- $\mathbf{err}_k = \mathbf{x}_{E,d} - \mathbf{f}(\mathbf{q}_k)$ is the error function evaluated at $\mathbf{q}_k$.

The Jacobian matrix $\mathbf{J}$ maps joint velocities to end-effector velocities, and its inverse is used to adjust the joint parameters to reduce the error. Newton's method **converges quadratically quickly only near the solution** but requires the computation of the Jacobian and its inverse, which can be computationally expensive and **prone to numerical instability near singularities**.

2.5.2 Gradient Descent Method

The Gradient (Maximum Descent) method is another iterative approach used to minimize a cost function by moving in the direction of the steepest descent. In inverse kinematics, the cost function $E(\mathbf{q})$ typically represents the squared error between the current and desired end-effector poses. The update rule for the Gradient method is given by:

$$\mathbf{q}_{k+1} = \mathbf{q}_k - \alpha \nabla E(\mathbf{q}_k)$$

Here,

- $\mathbf{q}_k$ is the joint configuration at iteration k.

- α is the step size or learning rate.
- $\nabla E(\mathbf{q}_k)$ is the gradient of the cost function evaluated at $\mathbf{q}_k$.

The gradient $\nabla E(\mathbf{q}_k)$ is computed as:

$$\nabla E(\mathbf{q}_k) = -\mathbf{J}^T(\mathbf{q}_k)(\mathbf{x}_{E,d} - \mathbf{f}(\mathbf{q}_k))$$

where $\mathbf{J}^T$ is the transpose of the Jacobian matrix. The Gradient method is simpler to implement than Newton's method and **does not require the computation of the Jacobian inverse**. However, it **may converge more slowly, especially if the step size α is not chosen appropriately**. The good thing is it can be applied to redundant robot.

2.5.3 Problems with Jacobian - Singularity and Redundancy

The Jacobian describes the linear mapping from the joint velocity space to the end-effector/operational velocity space. The Jacobian is, in general, a function of joint space; some configurations in the joint space can lead to rank-deficient Jacobians. These configuratioins are called kinematic singularities. Finding singularities is paramount for the following reasons:

a) Singularities represent configurations at which mobility of the structure is reduced, i.e., it is not possible to impose an arbitrary motion to the end-effector.

b) When the structure is at a singularity, infinite solutions to the inverse kinematics problem may exist.

c) In the neighbourhood of a singularity, small velocities in the operational space may cause large velocities in the joint space.

Singularities can be classified as

- **Boundary Singularities**: this can occur when the manipulator is either outstretched or retracted; where the manipulator is at the boundary of its joint space.
- **Internal Singularities**: this can occur inside the reachable workspace and are generally caused by the alignment of two or more axes of motion,or else by the attainment of particular end-effector congurations. Unlike the above, these singularities constitute a serious problem, as they can be encountered anywhere in the reachable workspace for a planned path in the operational space.

Since, for the Maira robot, the operational space is $r = 6$ and the joint space is $n = 7$, we additionally encounter kinematic redundancy. The range space of $\mathbf{J}$ is the subspace $\mathcal{R}(\mathbf{J})$ in $\mathbb{R}^r$ of the end-effector velocities that can be generated by the joint velocities, in the given manipulator posture. The null space of $\mathbf{J}$ is the subspace $\mathcal{N}(\mathbf{J})$ in $\mathbb{R}^n$ of joint velocities that do not produce any end-effector velocity, in the given manipulator posture. If the Jacobian degenerates at a singularity, the dimension of the range space decreases while the dimension of the null space increases. The existence of a subspace $\mathcal{N}(\mathbf{J}) \neq \emptyset$ for a redundant manipulator allows determination of systematic techniques for handling redundant degrees of freedom. So, in order to handle redundancies we need to handle singularities.

2.5.4 Inverse Kinematics Algorithm with handling redundancy

Thus, an inverse kinematics for a robotic manipulator with joint limits and redundancy can be formulated as a constraint optimization problem. The objective is to find the joint configuration $\mathbf{q}$ that achieves the desired end-effector position and orientation while satisfying joint limits and handling redundancy.

It is important to underline that the inversion of the Jacobian can represent a serious inconvenience not only at a singularity but also in the neighbourhood of a singularity. A more rigorous analysis of the solution features in the neighbourhood of singular con gurations can be developed by resorting to the singular value decomposition (SVD) of matrix $\mathbf{J}$. An alternative solution overcoming the problem of inverting kinematics in the neighbourhood of a singularity is provided by the so-called damped least-squares (DLS) inverse of the Jacobian $\mathbf{J}$.

$$\tilde{\mathbf{J}} = \mathbf{J}^\top (\mathbf{J}\mathbf{J}^\top + \lambda \mathcal{I})^{-1}$$

where λ is a damping factor that renders the inversion better conditioned from a numerical viewpoint and takes care of singularities. Thus, our original Jacobian inverse is replaced by the above.

The damped inverse takes care of singularities but the redundancy is still there. The redundancy is handled using the nullspace projection of the damped inverse given as

$$\mathcal{N} = \mathcal{I} - \tilde{\mathbf{J}}\mathbf{J}$$

where $\mathcal{N}$ will contain the rows that correspond to nullspace and we use this to adjust the change in $\mathbf{q}$ closer to the mean of the joint space as follows:

$$\delta \mathbf{q}_k = \tilde{\mathbf{J}} \cdot \mathbf{err}_k + \alpha \mathcal{N}\Big(\frac{\mathbf{q}_{min} + \mathbf{q}_{max}}{2}\Big)$$

where alpha is the secondary objective gin or weight that forces how closer to stay towards the mean of joint space. Thus, the overall update step is as follows,

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \tilde{\mathbf{J}}(\mathbf{q}_k)\mathbf{err}_k + \alpha \mathcal{N}\Big(\frac{\mathbf{q}_{min} + \mathbf{q}_{max}}{2}\Big)$$

And finally clipping each update step to the joint space yields the following Algorithm 1.

Algorithm 1 Inverse Kinematics with Redundancy Handling

Require: $\mathbf{x}_{target}$: Target position and quaternion orientation
$\mathbf{q}_{seed}$: Initial joint configuration (seed)
$\mathbf{q}_{min}$: Vector of lower joint limits
$\mathbf{q}_{max}$: Vector of upper joint limits
ϵ: Tolerance for convergence
λ: Damping factor
α: Secondary objective gain
maxIterations: maximum number of iterations

Initialize $\mathbf{q} \leftarrow \mathbf{q}_{seed}$
$\lambda \leftarrow 1 \times 10^{-6}$
$\alpha \leftarrow 0.01$
for $i = 1$ to maxIterations **do**
  Compute forward kinematics $\mathbf{T} \leftarrow \mathbf{T}(\mathbf{q})$
  Compute Jacobian $\mathbf{J} \leftarrow \mathbf{J}(\mathbf{q})$
  Compute error $\mathbf{e} \leftarrow \mathbf{x}_{target} - \mathbf{x}(\mathbf{T})$
  if $\|\mathbf{e}\| < \epsilon$ **then**
    return q
  end if
  Compute damped pseudo-inverse $\mathbf{J}_{\text{pinv}} \leftarrow \mathbf{J}^T(\mathbf{J}\mathbf{J}^T + \lambda\mathbf{I})^{-1}$
  Compute null-space projection matrix $\mathbf{N} \leftarrow \mathbf{I} - \mathbf{J}_{\text{pinv}}\mathbf{J}$
  Compute secondary objective:
  $\mathbf{q}_{mid} \leftarrow \frac{\mathbf{q}_{max}+\mathbf{q}_{min}}{2}$
  $\mathbf{q}_{diff} \leftarrow \mathbf{q} - \mathbf{q}_{mid}$
  $\mathbf{q}_{sec} \leftarrow -\alpha\mathbf{q}_{diff}$
  Compute joint update $\Delta\mathbf{q} \leftarrow \mathbf{J}_{\text{pinv}}\mathbf{e} + \mathbf{N}\mathbf{q}_{sec}$
  Update joint angles $\mathbf{q} \leftarrow \mathbf{q} + \Delta\mathbf{q}$
  Enforce joint limits:
  for $j = 1$ to size($\mathbf{q}$) **do**
    if $\mathbf{q}[j] < \mathbf{q}_{min}[j]$ **then**
      $\mathbf{q}[j] \leftarrow \mathbf{q}_{min}[j]$
    end if
    if $\mathbf{q}[j] > \mathbf{q}_{max}[j]$ **then**
      $\mathbf{q}[j] \leftarrow \mathbf{q}_{max}[j]$
    end if
  end for
end for
Print "Max number of iterations reached. Solution found is proximal."
return q