ME594

Numerical Methods in Mechanical Engineering

Fall 2021

The Inverse Kinematics of a 4-DOF RRRR Manipulator in 3D Space via Newton's Method: A Comparison to the Geometric Approach

---

12th December, 2021

"I pledge that I have abided by the graduate student code of Academic Integrity."

The paper has been prepared by:

Aldrin D. Padua

ABSTRACT

In robotics, inverse kinematics is the process of computing the joint variables of a manipulator (also called a robotic arm) based on the desired position in space and the orientation of its endpoint. In practice, this process is solved by either the geometric or numerical approach. This paper explores one type of numerical approach, namely the Newton's method, to solve the inverse kinematics of a four degrees of freedom (4-DOF) four-revolute-joint (RRRR) manipulator in 3D space. This numerical method utilizes a single system of equations (i.e., a generalized solution), solved through Denavit-Hartenberg convention, that contains all sets of possible manipulator configurations. That contrasts with the geometric approach which requires a different set of equations for every type of configuration. The iterations involve solving for the four values of the four revolute-joint variables with only three known data which are the x, y, and z coordinates of the endpoint, and computations are carried out in matrix format. Hence, for the inverse of the non-square Jacobian matrix, the Moore-Penrose inverse (pseudoinverse) is utilized. The numerical method proves to be able to adjust to different configurations using only one set of equations, solely varying the output by the initial guesses fed into the iterative loop. To contain all sets of configurations in one set of solutions is a very promising characteristic as it eliminates the need for tedious derivation of different sets of solutions as in the case of the geometric approach. However, compared to the geometric approach with an average runtime of around 1.7ms, the average computation speed for the numerical method is still relatively slower (average runtime time ≈ 27ms) which translates to more power and memory consumption as evidenced by the large iteration count for every run. The main cause of this proves to be the unmodeled criterion for the selection process of the initial guesses which remains arbitrary. Although, the runtime of the numerical method is still considered fast when referenced to the average human response time to visual and auditory stimuli which is 180ms. They are not directly comparable to each other as the human response time includes the actuation of the task itself, while that of the measured average runtime in this paper encompasses only the function's elapsed time excluding the actuation time of the manipulator itself. However, since the measured average function runtime is kept in the range of 27ms for the numerical approach, that means a large chunk of the manipulator's total response time (if the actuation is included) can be maximized to compete with that of the human's because the time spent for calculation was kept minimal.

INTRODUCTION

A robotic arm or manipulator is composed of joints, links, and end-effector. Joints are the components connecting endpoints of a manipulator's link. They can either be revolute (rotational) or prismatic (translational). In this paper, only revolute types are used. Links are the rigid branches directly connected to joints. Lastly, the end-effector is the component at the end of the last link which actuates the task of robot.

In robotics, a robot's degree of freedom (DOF) is equivalent to the number of its joints. The degree of freedom represents the direction by which each robot's link can freely move. The 4-DOF RRRR (4-Revolute) manipulator that is examined in this paper is as shown in below figure.

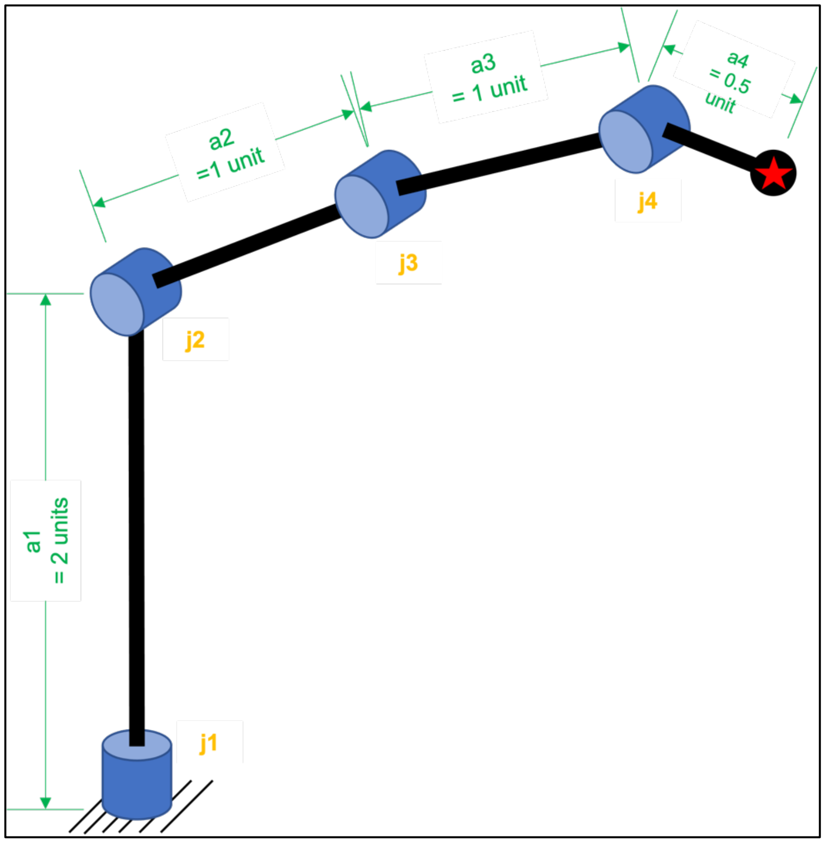

Figure 1. The image shows the actual subject (4-DOF RRRR Manipulator) of this paper. There is no end-effector but rather, just an endpoint. The length of each link is treated as constant.

The components j1, j2, j3, and j4 are the revolute joints while a1, a2, a3, and a4 are the links. The point marked with a red star is not the end-effector. This paper is concerned in demonstrating the use of numerical method in solving the inverse kinematics and its merits and demerits compared to the geometric approach. Thus, a manipulator without an end-effector will suffice. For clarity in the context of this work, the point with the red star will be referred to as the manipulator's endpoint, or simply endpoint.

The process of computing the desired position in space of the endpoint based on its joint variable values and orientation is called the *forward kinematics*. That is assuming that the joint variables are known for every desired final position. However, in real world application, that is often not the case. In practice, the known data points are the final position coordinates while the configuration parameters or the joint variable values are the unknowns to be solved. The process of solving for the joint variables and orientation based on the endpoint position coordinates in space is called the *inverse kinematics*.

The objective of this paper is to demonstrate how the inverse kinematics of the given 4-DOF RRRR Manipulator is solved through a numerical method (i.e., Newton's method). Another is to compare the numerical approach against a geometric counterpart to highlight the strong and weak points of each method and determine what kind of tasks are they more suitable.

Since robots, in general, are built to replace human effort, it will be very fitting to correlate the function runtime of the implemented methods with the average human response time when performing tasks. Also, the function runtime will be correlated to the manipulator's motion in terms of power and time consumption to evaluate the efficiency and optimization capability of the numerical method against the geometric approach. Other characteristics to be compared are the adaptability of each method to different configurations, and how the solution for each is derived. Lastly, this paper will provide grounds in determining whether one of the two presented approaches can be regarded as a general method to solve all inverse kinematics problems.

THEORY AND NUMERICAL METHOD

Forward and Inverse Kinematics

The inverse kinematics is commonly solved through geometric approach for robotic arms with 3 DOFs or less. With only 3 DOFs at maximum, the geometry is simple to analyze, thus, allowing the computation of the joint variable values in terms of the endpoint position coordinates relatively easy. Function runtime of the geometric approach is very fast compared to numerical method since calculations are directly carried out without iterations. However, one limitation for the geometric approach is that it is very tedious to compute for manipulators with more than 3 DOFs as in the case of this paper. Another is that one solution only corresponds to one type of configuration. If there are several types of configurations, then all others must be computed separately and integrated one by one into the robot's algorithm. To make this clear, refer to the hypothetical 2-DOF planar manipulator below (for demonstration only; not the actual subject of this paper).

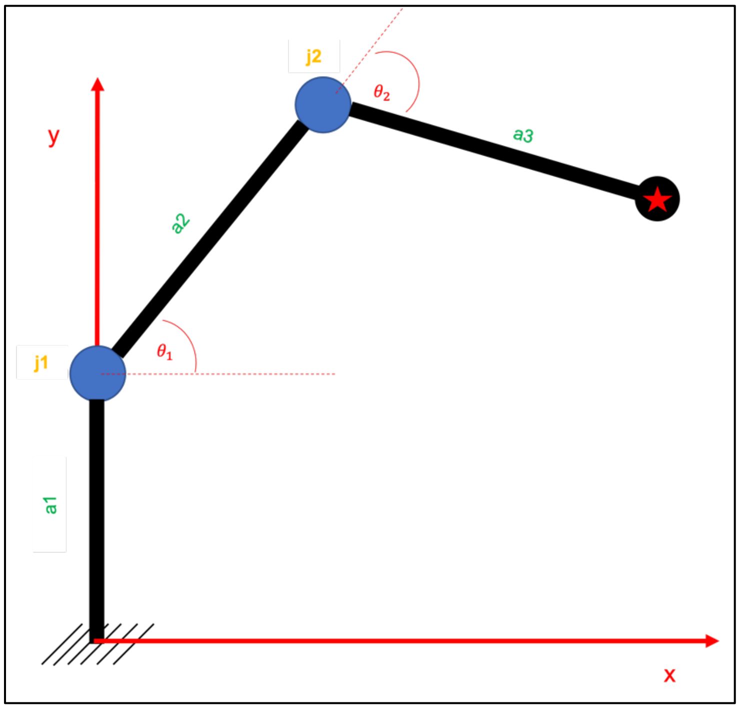

Figure 2. A hypothetical 2-DOF Planar Manipulator in elbow-up configuration illustrates a simple geometry where the endpoint position in space can be derived by plain trigonometry. (This manipulator is not the subject of this paper and is just used to demonstrate the concept of the geometric approach.)

In the above figure, the equations for the joint variables, θ_1 and θ_2, can be easily derived by trigonometry, expressing the unknown values of theta in terms of the known position values, x and y. Notice that the second joint, j2, is in a position such that links a2 and a3 appear to be on a concave down form. This is one type of configuration called *elbow-up*. By inspection, one can infer that by rotating j1 below its x-axis and positioning j2 such that links a2 and a3 appear to be concaving upward, the same endpoint position can still be reached as shown in the next figure.

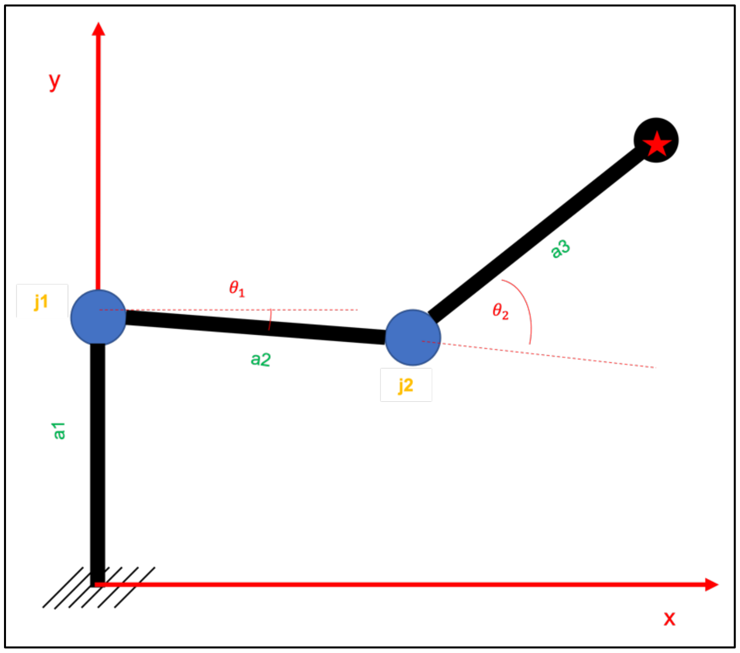

Figure 3. The same endpoint position can be achieved by a different configuration as illustrated by the same hypothetical 2-DOF Planar Manipulator in elbow-down configuration. (This manipulator is not the subject of this paper and is just used to demonstrate the concept of the geometric approach.)

Above figure shows another type of configuration called *elbow-down*.

If only the equations for the elbow-up configuration are integrated into the algorithm which makes the robot move, then at any arbitrary endpoint position in space, the manipulator will take the form of the elbow-up configuration as shown in Figure 4. This is a proven behavior for geometrically derived solutions.

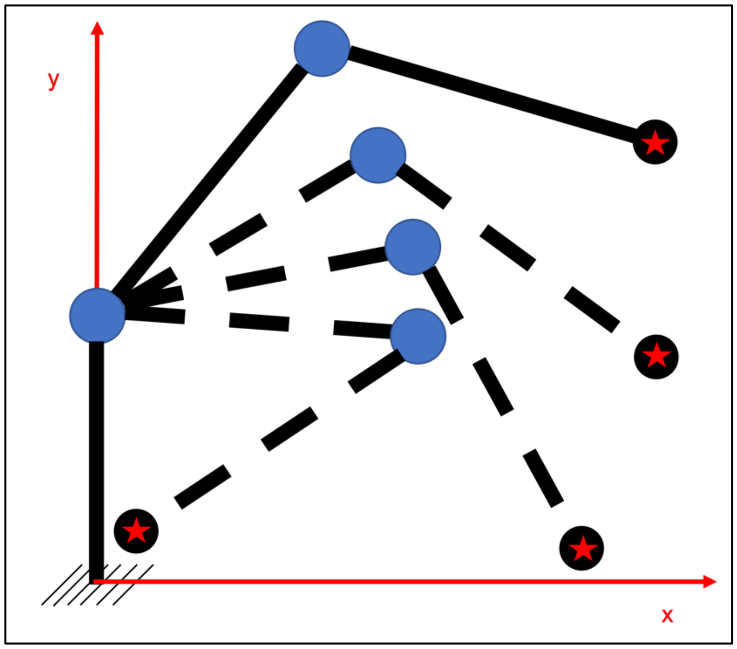

Figure 4. The manipulator is expected to always be in an elbow-up configuration no matter what position in space it reaches. This happens when the equation used by the algorithm is derived using elbow-up configuration and no other equations are fed. (This manipulator is not the subject of this paper and is just used to demonstrate the concept of the geometric approach.)

There are cases when one desires to make the robot move and switch into the elbow-down configuration, that is, to minimize the displacement or to avoid an obstacle in the workspace. Then in such situations, the equations for the elbow down configuration must be solved separately and integrated into the algorithm. Furthermore, another algorithm must be developed for the criterion which will make the robot dynamically determine and choose which configuration to take since technically, there is now more than one solution for the combination of joint variables required to achieve any arbitrary position. This is acceptable for a simple manipulator such as the one illustrated in Figures 2 through 4. But for more complex systems (i.e., manipulators with more than 3-DOFs) where there are many configuration combinations to choose from, the solution set count multiplies and the computation gets more tedious in two dimensions, much more in 3D space.

In this paper, the use of a single set of solutions (one system of linear equations) to solve the inverse kinematics of the 4-DOF RRRR manipulator in 3D is explored. The system of linear equations relating the x, y, and z coordinates of the endpoint to the joint variables θ_1, θ_2, θ_3, and θ_4 are obtained by forward kinematics via Denavit-Hartenberg convention, or simply *DH* convention. With only a single system of linear equations, different configuration types are shown to be attainable using a numerical approach, namely the Newton's method.

The Denavit-Hartenberg (DH) Convention

To solve the inverse kinematics, the general solution obtained through the forward kinematics is still required. This general solution will be the main part of the iterative method that will be used to solve for the joint variables, thetas. The use of trigonometric approach to obtain the forward kinematics solution proved to be tedious especially when dealing with three-dimensional space. The DH convention simplifies this process by establishing rules on how to define reference frames for each joint. This helps in obtaining the final homogenous transformation matrix of each link in a more organized and standardized fashion. The mentioned final homogeneous transformation matrix contains the generalized solution that is required by the numerical method.

DH convention starts by mapping a reference frame in each manipulator's link endpoints. The following is a summarized version of the convention, taken into the context of this paper:

1. The z-axis is always the joint's axis of rotation.
2. The orientation of the first frame, also called the base frame or zeroth frame, is purely arbitrary.
3. Let i be from 1 to n where n is the number of links. For every i-th frame, the x_i and x_{i-1} axes must both lie in a plane perpendicular to z_{i-1} axis. Direction is arbitrary.
4. The direction of the y-axis is determined using the right-hand rule.

   Right-hand rule: Position the thumb, index finger, and middle finger in a manner where they are all perpendicular to each other. Align the thumb with the z-axis and the index finger with the x-axis. The direction of the middle finger is the direction of the positive y-axis.

The mapped figure is as shown below. (Note: The manipulator was rotated in a manner where all the links were either aligned or exactly perpendicular to each other. This was not a necessary step and was just done to clearly visualize the rules in defining the reference frames.)

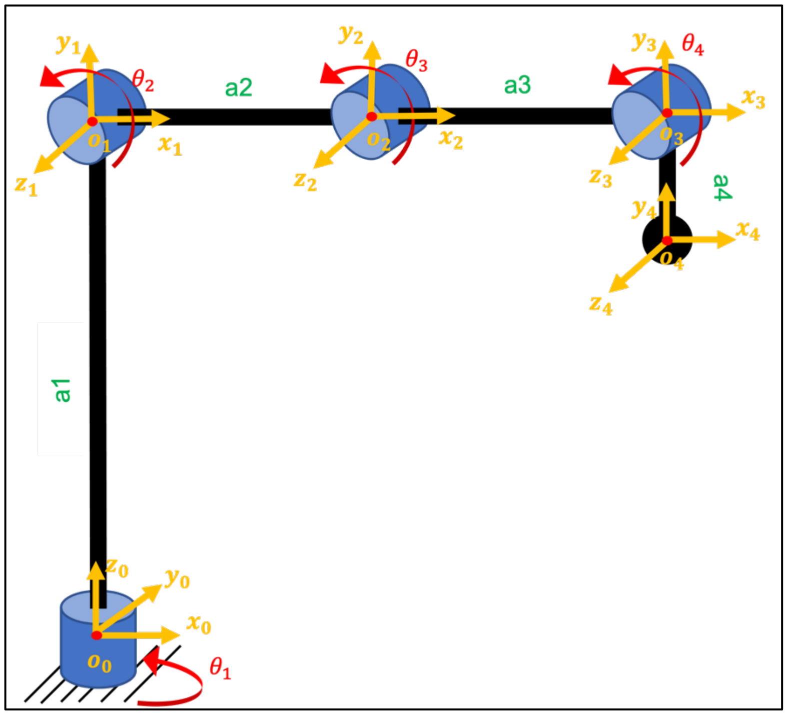

Figure 5. The 4-DOF RRRR Manipulator (actual subject) is mapped with reference frames following the DH convention. The zeroth frame marked as by o_0 is arbitrarily positioned. Re-positioning the links in parallel or perpendicular to each other when mapping the reference frames is not necessary and is only done to give better visualization to the DH rules.

To derive the homogenous transformation matrix for each manipulator link, the construction of the DH table is necessary. The DH table contains four parameters namely,

$a = \ distance\ between\ z_i\ and\ z_{i-1}\ \ axes\ measured\ along\ the\ x_i\ axis$
$\alpha = \ the\ angle\ by\ which\ z_{i-1}$ axis need to rotate to mimic the position of z_i axis
$d = \ the\ displacement\ between\ x_{i-1}\ and\ x_i\ \ axes\ measured\ along\ the\ z_{i-1}\ axis$
$\theta = \ the\ angle\ between\ x_{i-1}\ and\ x_i\ \ axes, or\ simply\ the\ angle\ of\ rotation\ of\ joint\ j_i$

If z_{i-1} and z_i axes are aligned, angle α is zero. Otherwise, the positive sense of the angle α of every i-th frame is determined by another right-hand rule.

Right-hand rule: Stick the thumb out. Curve the other fingers towards their natural direction. Align the thumb with the x_i axis. The direction of the curved fingers denotes the positive sense of angle α.

Following the above convention, the DH table for the 4-DOF RRR manipulator was derived as follows:

Table 1. This is the DH Table of the 4-DOF RRRR Manipulator derived through the mapped schematic in Figure 5. The parameters were obtained by looking at each link one at a time from a1 to a4. The forward endpoint of a link (considering down-to-up and left-to-right as the directions) corresponds to the i-th frame while the backward endpoint is the (i-1)th-frame.

Link	$\boldsymbol{a_i}$	$\boldsymbol{\alpha_i}$	$\boldsymbol{d_i}$	$\boldsymbol{\theta_i}$
1	0	90°	a1	θ_1
2	a2	0	0	θ_2
3	a3	0	0	θ_3
4	a4	0	0	θ_4

Each row in the DH table represents a homogeneous matrix, A_i, for each link that is equivalent to the product of four basic transformations as follows:

$$A_i = Rot_{z,\,\theta_i} \cdot Trans_{z,\,d_i} \cdot Trans_{x,\,a_i} \cdot Rot_{x,\,\alpha_i}$$

where $Rot_{z,\,\theta_i}$ is the rotation about the z-axis by θ_i, $Trans_{z,\,d_i}$ is the translation along z-axis by d_i, $Trans_{x,\,a_i}$ is the translation along the x-axis by a_i, and $Rot_{x,\,\alpha_i}$ is the rotation about the x-axis by α_i. The homogenous transformation for each link is then generalized by the following matrix:

$$A_i = \begin{bmatrix} R & d \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where R is the rotation matrix, and d is the displacement vector. Symbols c and s represent the cosine and the sine functions, respectively.

Using the above generalized transformation matrix, the homogeneous matrix for each link i was derived as follows:

$$A_1 = \begin{bmatrix} c_{\theta_1} & 0 & s_{\theta_1} & 0 \\ s_{\theta_1} & 0 & -c_{\theta_1} & 0 \\ 0 & 1 & 0 & a1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_3 = \begin{bmatrix} c_{\theta_3} & -s_{\theta_3} & 0 & a3c_{\theta_3} \\ s_{\theta_3} & c_{\theta_3} & 0 & a3s_{\theta_3} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} c_{\theta_2} & -s_{\theta_2} & 0 & a2c_{\theta_2} \\ s_{\theta_2} & c_{\theta_2} & 0 & a2s_{\theta_2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_4 = \begin{bmatrix} c_{\theta_4} & -s_{\theta_4} & 0 & a4c_{\theta_4} \\ s_{\theta_4} & c_{\theta_4} & 0 & a4s_{\theta_4} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Chronologically multiplying all the above homogeneous matrices produced the final homogeneous transformation of the last frame with respect to the zeroth frame, denoted by A_4^0, that contains the system of linear equations required to solve the inverse kinematics numerically. The final homogenous solution was derived as follows:

$$A_4^0 = \begin{bmatrix} c_{\theta_1}c_{\theta_2+\theta_3+\theta_4} & -c_{\theta_1}s_{\theta_2+\theta_3+\theta_4} & s_{\theta_1} & c_{\theta_1}(a_2c_{\theta_2} + a_3c_{\theta_2+\theta_3} + a_4c_{\theta_2+\theta_3+\theta_4}) \\ s_{\theta_1}c_{\theta_2+\theta_3+\theta_4} & -s_{\theta_1}s_{\theta_2+\theta_3+\theta_4} & -c_{\theta_1} & s_{\theta_1}(a_2c_{\theta_2} + a_3c_{\theta_2+\theta_3} + a_4c_{\theta_2+\theta_3+\theta_4}) \\ s_{\theta_2+\theta_3+\theta_4} & c_{\theta_2+\theta_3+\theta_4} & 0 & a_1 + a_2s_{\theta_2} + a_3s_{\theta_2+\theta_3} + a_4s_{\theta_2+\theta_3+\theta_4} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Extracting the displacement vector from the final homogeneous transformation matrix, the general solution to the position of the manipulator's endpoint was obtained.

$$d_4^0 = X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} c_{\theta_1}\left(a_2c_{\theta_2} + a_3c_{\theta_2+\theta_3} + a_4c_{\theta_2+\theta_3+\theta_4}\right) \\ s_{\theta_1}\left(a_2c_{\theta_2} + a_3c_{\theta_2+\theta_3} + a_4c_{\theta_2+\theta_3+\theta_4}\right) \\ a_1 + a_2s_{\theta_2} + a_3s_{\theta_2+\theta_3} + a_4s_{\theta_2+\theta_3+\theta_4} \end{bmatrix} \qquad (eq. 1)$$

By intuition, the system of linear equations shown is regarded as undetermined. That means, there are more unknown values than available equations. This further implies that the matrix of partial derivatives, also called the Jacobian, is a non-square matrix that has a rank equal to the row dimension.

Given above constraints, since the inverse of the Jacobian was necessary for the iterative method to work, and that the regular inverse was mathematically incalculable for a non-square matrix, the Moore-Penrose pseudoinverse was used instead.

The Jacobian Matrix

The Jacobian is a very pivotal quantity in the analysis and control of robot motion, trajectory planning, and even in the determination of manipulator's singularities; topics that are out of this work's scope. The Jacobian matrix discretizes and correlates the role of the angular velocity induced by the rotation of each joint in relation to the total velocity experienced by the manipulator's endpoint as it moves to the desired position. Mathematically speaking, the Jacobian matrix is composed of the partial derivatives of the position equations with respect to each of the joint variables.

Using the previously obtained generalized solution to the endpoint position, the Jacobian matrix was derived as follows:

$$J = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

$$= \begin{bmatrix} -s_{\theta_1}\left(a_2c_{\theta_2} + a_3c_{\theta_2+\theta_3} + a_4c_{\theta_2+\theta_3+\theta_4}\right) & -c_{\theta_1}\left(a_2s_{\theta_2} + a_3s_{\theta_2+\theta_3} + a_4s_{\theta_2+\theta_3+\theta_4}\right) & -c_{\theta_1}\left(a_3s_{\theta_2+\theta_3} + a_4s_{\theta_2+\theta_3+\theta_4}\right) & -c_{\theta_1}a_4s_{\theta_2+\theta_3+\theta_4} \\ -c_{\theta_1}\left(a_2c_{\theta_2} + a_3c_{\theta_2+\theta_3} + a_4c_{\theta_2+\theta_3+\theta_4}\right) & -s_{\theta_1}\left(a_2s_{\theta_2} + a_3s_{\theta_2+\theta_3} + a_4s_{\theta_2+\theta_3+\theta_4}\right) & -s_{\theta_1}\left(a_3s_{\theta_2+\theta_3} + a_4s_{\theta_2+\theta_3+\theta_4}\right) & -s_{\theta_1}a_4s_{\theta_2+\theta_3+\theta_4} \\ 0 & a_2c_{\theta_2} + a_3c_{\theta_2+\theta_3} + a_4c_{\theta_2+\theta_3+\theta_4} & a_3c_{\theta_2+\theta_3} + a_4c_{\theta_2+\theta_3+\theta_4} & a_4c_{\theta_2+\theta_3+\theta_4} \end{bmatrix}$$

(eq. 2)

where the columns of J represent the angular velocities $\theta_1{}'$, $\theta_2{}'$, $\theta_3{}'$, and $\theta_4{}'$, respectively.

The Moore-Penrose Inverse for Undetermined System of Linear Equations

In MATLAB, there exists a function called *pinv()* which performs the operation of the Moore-Penrose Inverse, or simply called the pseudoinverse. Although this function is the one directly used in the code implementing the numerical method for this paper, it is important to know what's going on in the background.

The pseudoinverse is the general definition of what is known as the regular inverse for square matrices. While the regular inverse does not exist for matrices other than square ones, the pseudoinverse does. Thus, for any matrix A, there exists a pseudoinverse, $A^{\dagger}$(A-dagger), whose uniqueness satisfies and can be proven by the following four properties known as the Penrose conditions:

1. $AA^{\dagger}A = A$
2. $A^{\dagger}AA^{\dagger} = A^{\dagger}$
3. $A^{\dagger}A = (A^{\dagger}A)^T$
4. $AA^{\dagger} = (AA^{\dagger})^T$

The pseudoinverse is considered the generalized form of the regular inverse because it is exactly equal to the regular inverse for full-rank square matrices. The pseudoinverse is generally computed via Singular Value Decomposition (SVD), but it has two simplified forms which are applicable for non-square matrices in full rank. Consider a matrix A with dimensions $m \times n$, where m is the number of rows and n is the number of columns. Then, the two simplified forms of the pseudoinverse for full-rank matrices are defined as:

1. If $m < n$ and the rank is equal to m (full row rank)

$$A^{\dagger} = A^T(AA^T)^{-1}$$

2. If $m > n$ and the rank is equal to n (full column rank)

$$A^{\dagger} = (A^TA)^{-1}A^T$$

The pseudoinverse that the numerical method of this paper implements is of the *form no. 1* as it satisfies the case for undetermined system of linear equations where $m < n$. Below is a proof that *form no. 1* satisfies all four of Penrose conditions.

Condition 1: $AA^{\dagger}A = A$

$$\begin{aligned}&\textit{via LHS manipulation}\\&A(A^T(AA^T)^{-1})A = A\\&(AA^T)(AA^T)^{-1}A = A\\&A = A\end{aligned}$$

Condition 2: $A^{\dagger}AA^{\dagger} = A^{\dagger}$

$$\begin{aligned}&\textit{via LHS manipulation}\\&(A^T(AA^T)^{-1})A(A^T(AA^T)^{-1}) = A^{\dagger}\\&(A^T(AA^T)^{-1})(AA^T)(AA^T)^{-1} = A^{\dagger}\\&(A^T(AA^T)^{-1}) = A^{\dagger}\\&A^{\dagger} = A^{\dagger}\end{aligned}$$

Condition 3: $A^{\dagger}A = (A^{\dagger}A)^T$

$$\begin{aligned}
&via\ RHS\ manipulation \\
&A^{\dagger}A = (A^{\dagger}A)^T \\
&A^{\dagger}A = A^T[A^T(AA^T)^{-1}]^T \\
&A^{\dagger}A = A^T[(AA^T)^{-1}]^T A \\
&A^{\dagger}A = A^T[(AA^T)^T]^{-1} A \\
&A^{\dagger}A = A^T(AA^T)^{-1} A \\
&A^{\dagger}A = A^{\dagger}A
\end{aligned}$$

Condition 4: $AA^{\dagger} = (AA^{\dagger})^T$

$$\begin{aligned}
&via\ manipulation\ of\ both\ sides \\
&A(A^T(AA^T)^{-1}) = (A(A^T(AA^T)^{-1}))^T \\
&(AA^T)(AA^T)^{-1} = ((AA^T)(AA^T)^{-1})^T \\
&I = (I)^T \\
&I = I
\end{aligned}$$

For undetermined systems, there cannot be a unique solution. In the context of this paper, since there are four unknowns and only three available equations, any three combinations of any three unknowns may solve the system while the other one remains a free (arbitrary) variable. That means that for every arbitrary value of the free variable, there is one solution, thus, the 'infinitely many solutions' issue. However, by utilizing the pseudoinverse, the best one is picked out of the infinite solution count; that is, the least norm solution.

Algebraically speaking, consider the equation $Ax = b$. For overdetermined systems, where there are more equations than unknowns, the problem turns into finding that one solution that minimizes the residual the most to best satisfy the condition $\|x - A^{\dagger}b\| \approx 0$.

On the other hand, for underdetermined systems, the problem turns into finding that one solution that minimizes the value $\|x\| \approx \|-A^{\dagger}b\|$,because unlike the case of overdetermined systems, b here is fixed. As such, the problem turns into optimization problem where the desired value of x is the one that has the shortest length. Since x is a vector, then the length is represented by the Euclidean norm, hence, the term "least norm solution."

The Newton's Method

The inverse kinematics for the 4-DOF RRRR Manipulator in 3D space was solved using the theories and equations detailed in the previous sections, in conjunction with the Newton's method.

The iteration equation used in the numerical method was defined as

$$q_{i+1} = q_i - J(q_i)^{\dagger} * F \qquad (eq. 3)$$

where q_i is the vector containing the values of joint variables (θ_1 , θ_2 , θ_3 , & θ_4), $J(q_i)^{\dagger}$ is the pseudoinverse of the non-square Jacobian matrix of q_i, F is the function vector derived by equating the vector of position functions to zero, and q_{i+1} is the approximate next value of q_i.

The convergence criterion used to converge to the desired solution was the absolute error defined as

$$\|q_{i+1} - q_i\| < 10^{-6}.$$

To assess the Newton's method in solving the inverse kinematics, two endpoint positions were considered for the manipulator's simulations. For each endpoint position, 3 different sets of initial guesses were tested for q_i. Hence, there was a total of 6 simulation runs. For each run, the iteration count, iteration runtime, and absolute error were recorded, and the manipulator's configuration simulated and plotted.

The MATLAB function code for the algorithm of the Newton's method was written under *InvKin_Newton.m* file. The code was also attached in the *Appendix (Part I)* along with the MATLAB script, *driver.m (Appendix (Part III))*, that runs the simulation of the whole process and plots the manipulator's configuration.

The Geometric Approach

As previously discussed, going with the geometric approach involves computing separate set of equations for different configuration types. That is the case when one desires to vary the geometric form of the manipulator with respect to the endpoint position for whatever desired purpose it may serve.

For the sake of comparison and to assess the strong and weak points of the numerical method versus the geometric approach, the results of both methods were compared. As such, one of the many sets of geometric solutions to the 4-DOF RRRR Manipulator was derived and its algorithm coded, as detailed in the *InvKin_Geometric.m* MATLAB function file which was also included in the *Appendix (Part II)*.

The derived geometric set of solutions was purposefully constrained in the following conditions:

1. The joint variable θ_4 negates the total angle formed by the sum of the other two joint variables θ_2 & θ_3, such that the link a4 is always parallel to the x-axis of the zeroth-frame or the base frame.
2. Joint variable θ_3 always rotate clockwise with respect to x_2 axis.

The set of solutions for the joint variables of the above configuration were derived as follows:

$$\theta_1 = tan^{-1}\left(\frac{y_e}{x_e}\right) \quad (eq.4)$$

$$\theta_2 = cos^{-1}\left(\frac{(\sqrt{{x_e}^2 + {y_e}^2} - a4)^2 + (z_e - a1)^2 + a2^2 - a3^2}{2 \times a2 \times \sqrt{(\sqrt{{x_e}^2 + {y_e}^2} - a4)^2 + (z_e - a1)^2}}\right) \times tan^{-1}\left(\frac{z_e - a1}{\sqrt{{x_e}^2 + {y_e}^2} - a4}\right) \quad (eq.5)$$

$$\theta_3 = -\left[\pi - cos^{-1}\left(\frac{a2^2 + a3^2 - \left(\sqrt{{x_e}^2 + {y_e}^2} - a4\right)^2 + (z_e - a1)^2}{2 \times a2 \times a3}\right)\right] \quad (eq.6)$$

$$\theta_4 = -(\theta_2 + \theta_3) \quad (eq.7)$$

Since all solutions were directly expressed in terms of the known endpoint coordinates, the iteration count and the absolute error values are both not applicable to this approach. Hence, the value that was recorded aside from the calculated joint variables, was only the function runtime.

The full derivation of the geometric set of solutions can be found in the *Appendix (Part IV)*.

RESULTS AND DISCUSSION

I. Numerical Approach (Newton's Method)

Global Constants:

$a1 = 2\, units$

$a2 = a3 = 1\, unit$

$a4 = 0.5\, unit$

Convergence criterion, $tol = 10^{-6}$

Note:

The theta values in the joint variable vectors, q_{guess} and q_f, are defined in degrees.

Legend:

$P = endpoint\ position$

$q_{guess} = initial\ guess\ for\ the\ q_i\ of\ the\ iteration$

$iter_{count} = total\ iterations\ count\ to\ arrive\ at\ the\ solution\ q_f$

$abs_{err} = absolute\ error\ comapring\ q_f to\ its\ previous\ value\ before\ convergence$

$runtime = the\ elapsed\ time\ by\ the\ numerical\ iteration$

==============================Test 1================================

$$\boldsymbol{Test\ Constant}: P = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1.5 \\ -1 \\ 2.5 \end{bmatrix}$$

Simulation 1

Input:

$$\mathrm{q_{guess}} = \begin{bmatrix} -90 \\ -60 \\ -30 \\ 0 \end{bmatrix}$$

Output:

$$q_f = \begin{bmatrix} -33.6901 \\ 64.9475 \\ -69.7874 \\ -35.1833 \end{bmatrix}$$

$iter_{count} = 842$

$abs_{err} = 9.9711e-07$

$runtime = 27.545\ ms$

Plot:

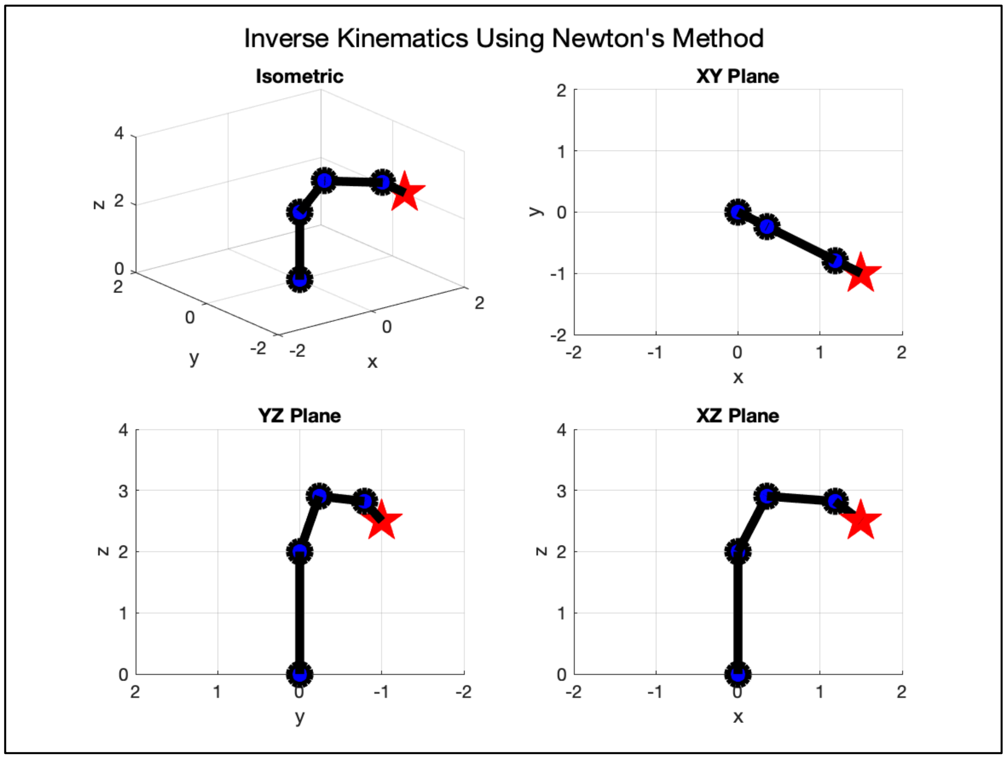

Figure 6. (Test 1 - Simulation 1) Joints 2 through 4, and joint 3 through the endpoint are all in elbow-up configurations. Without explicitly defining the elbow-up configuration as the geometric approach does, the numerical approach converges to this solution.

Simulation 2

Input:

$$\mathrm{q_{guess}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Output:

$$q_f = \begin{bmatrix} -33.6900 \\ -31.7057 \\ 64.2213 \\ 44.9195 \end{bmatrix}$$
$$iter_{count} = 774$$
$$abs_{err} = 9.9132e-07$$
$$runtime = 20.684\ ms$$

Plot:

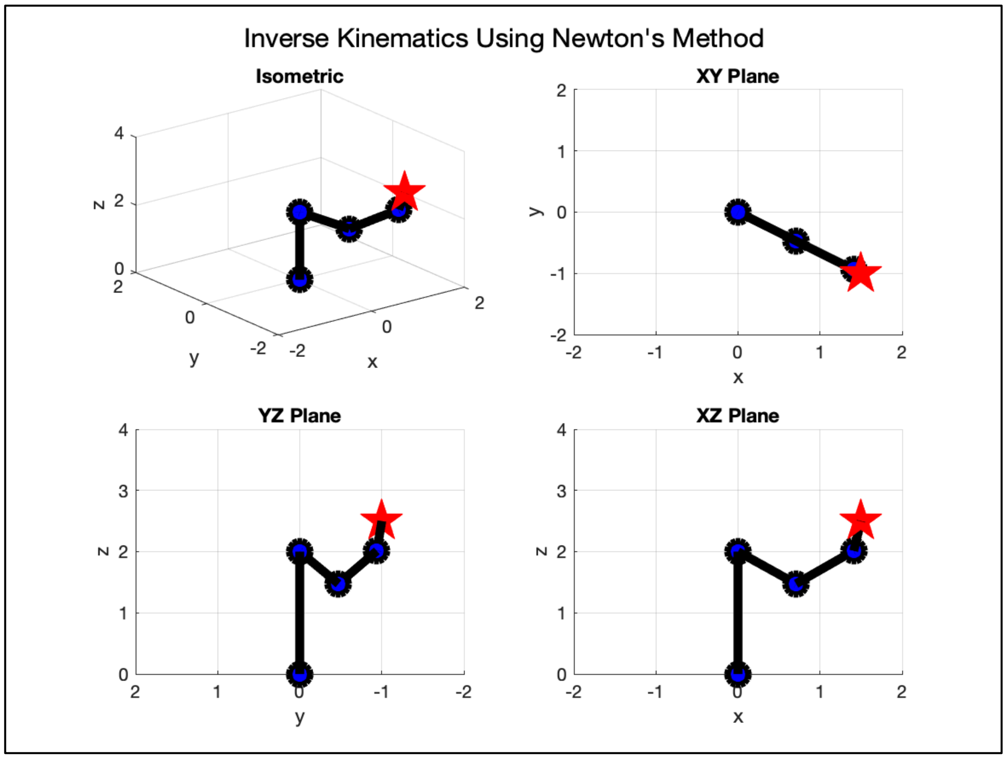

Figure 7. (Test 1 - Simulation 2) Joints 2 through 4, and joint 3 through the endpoint are all in elbow-down configurations. Without explicitly defining the elbow-down configuration, the numerical approach can shift and converge to this solution just by changing the initial guesses.

Simulation 3

Input:

$$\mathrm{q_{guess}} = \begin{bmatrix} -30 \\ 0 \\ -30 \\ 60 \end{bmatrix}$$

Output:

$$q_f = \begin{bmatrix} -33.6901 \\ 16.7392 \\ -32.6567 \\ 119.3921 \end{bmatrix}$$
$$iter_{count} = 765$$
$$abs_{err} = 9.8534e-07$$
$$runtime = 25.916\, ms$$

Plot:

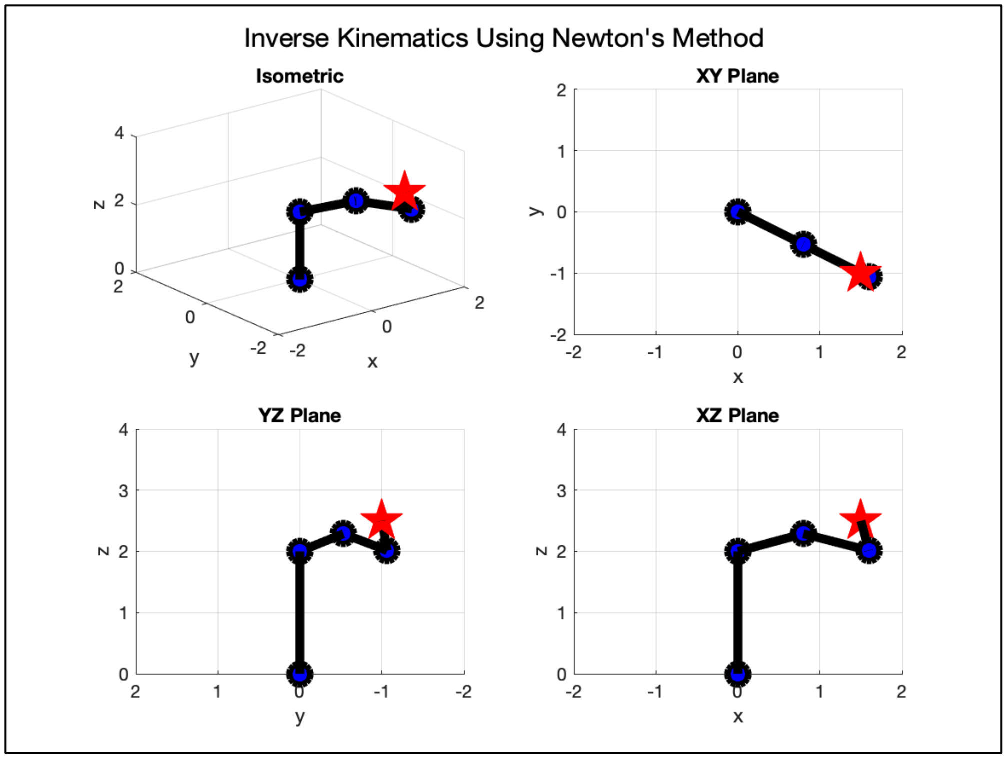

Figure 8. (Test 1 - Simulation 3) Joints 2 through 4 is in elbow-up configuration, while joint 3 through the endpoint is in elbow-down configuration. The numerical approach can mix different configuration types without the need to use separate equations in the computation.

The results in Test 1 clearly illustrate the ability of the numerical method to solve the inverse kinematics of the 4-DOF RRRR manipulator. Observing Figures 6-8, the endpoint position is constant, but the configuration types are different for each simulation. Without changing the generalized set of solutions (obtained via DH convention) in the algorithm, the numerical method can change the configuration type of the manipulator by solely varying the initial guess for each run. In Simulation 1, joints 2-4, and joint 3-endpoint are all in elbow-up configurations, while in Simulation 2, joints 2-4, and joint 3-endpoint are all in elbow-down configurations. Another configuration type is achieved in Simulation 3. These all resulted by purely varying the initial guess for the function while all other parameters are kept constant. This proves that the numerical method encompasses all the configuration types as opposed to the geometric approach which requires different sets of solutions for each.

==============================Test 2================================

$$\boldsymbol{Test\ Constant}: P = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -1.8 \\ 0.5 \\ 1.5 \end{bmatrix}$$

Simulation 1

Input:

$$q_{\text{guess}} = \begin{bmatrix} -90 \\ -60 \\ -30 \\ 0 \end{bmatrix}$$

Output:

$$q_f = \begin{bmatrix} 164.4759 \\ 32.8290 \\ -69.3625 \\ -26.8058 \end{bmatrix}$$
$$iter_{count} = 820$$
$$abs_{err} = 9.8647e-07$$
$$runtime = 25.977\ ms$$

Plot:

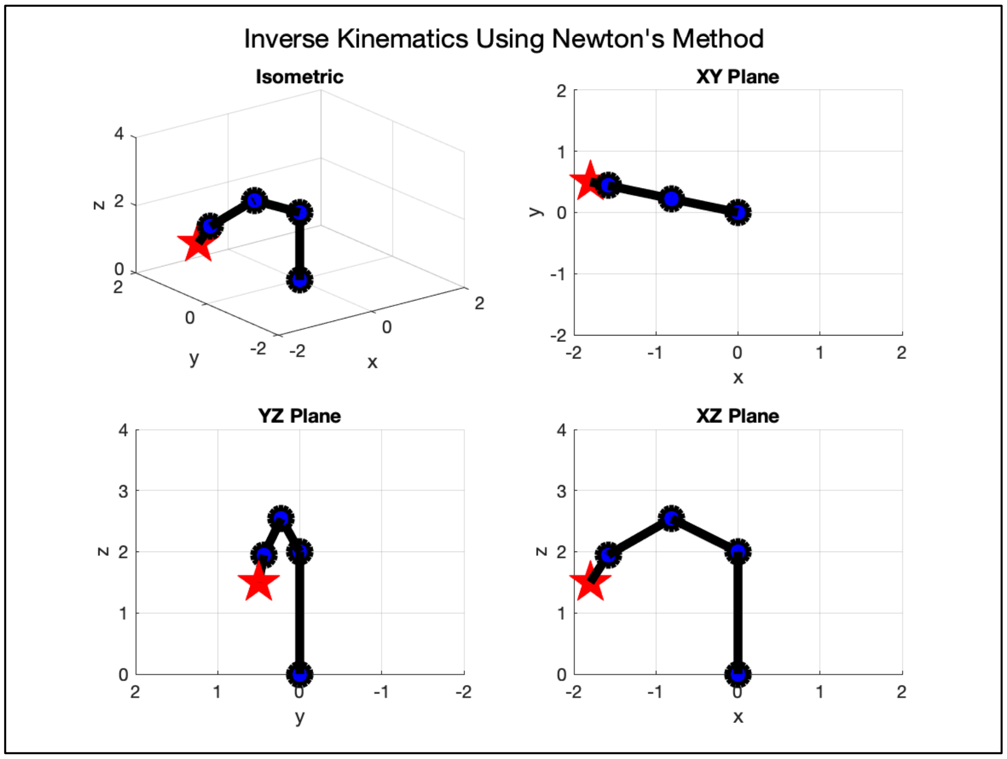

Figure 9. (Test 2 - Simulation 1) Joints 2 through 4, and joint 3 through the endpoint are all in elbow-up configurations. Using a different endpoint position but the same initial guess as in Test 1 – Simulation 1, the numerical approach converges to the same configuration type.

Simulation 2

Input:

$$q_{\text{guess}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Output:

$$q_f = \begin{bmatrix} 164.4759 \\ 31.1339 \\ -64.4487 \\ -36.0110 \end{bmatrix}$$
$$iter_{count} = 888$$
$$abs_{err} = 9.9122e-07$$
$$runtime = 26.555\ ms$$

Plot:

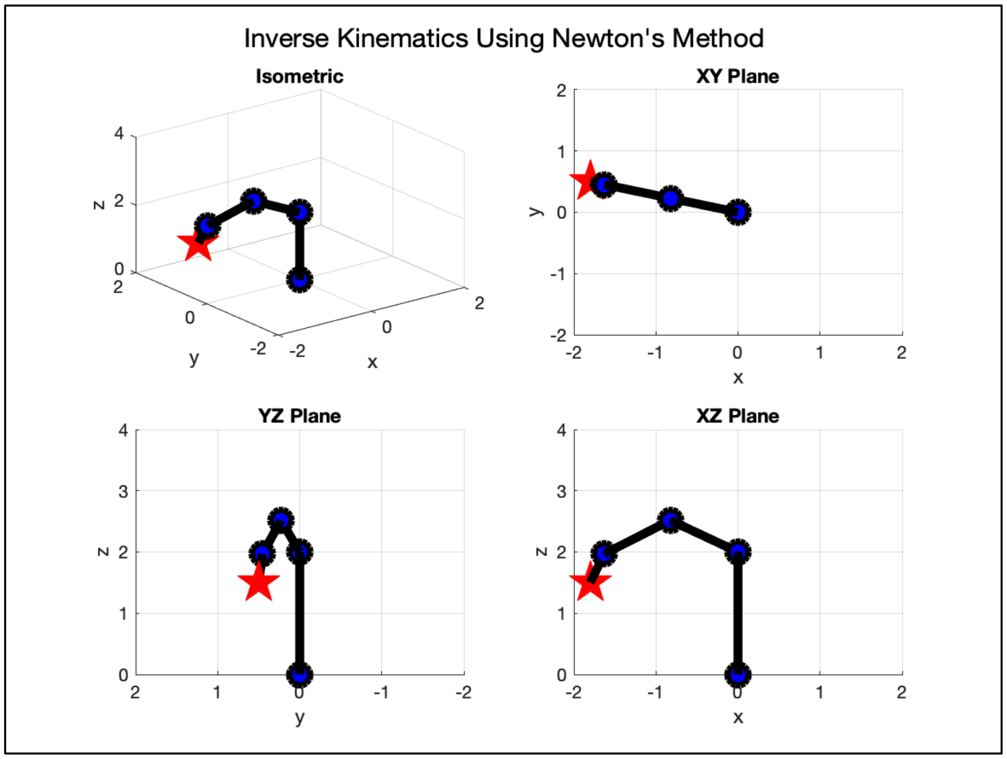

Figure 10. (Test 2 - Simulation 2) Joints 2 through 4, and joint 3 through the endpoint are all in elbow-up configurations. Using a different endpoint position but the same initial guess as in Test 1 – Simulation 2, the numerical approach does not converge to the same configuration type.

Simulation 3

Input:

$$\mathrm{q_{guess}} = \begin{bmatrix} -30 \\ 0 \\ -30 \\ 60 \end{bmatrix}$$

Output:

$$q_f = \begin{bmatrix} 164.4759 \\ -10.5806 \\ -38.7526 \\ 111.4938 \end{bmatrix}$$
$$iter_{count} = 906$$
$$abs_{err} = 9.8339e-07$$
$$runtime = 33.244\ ms$$

Plot:

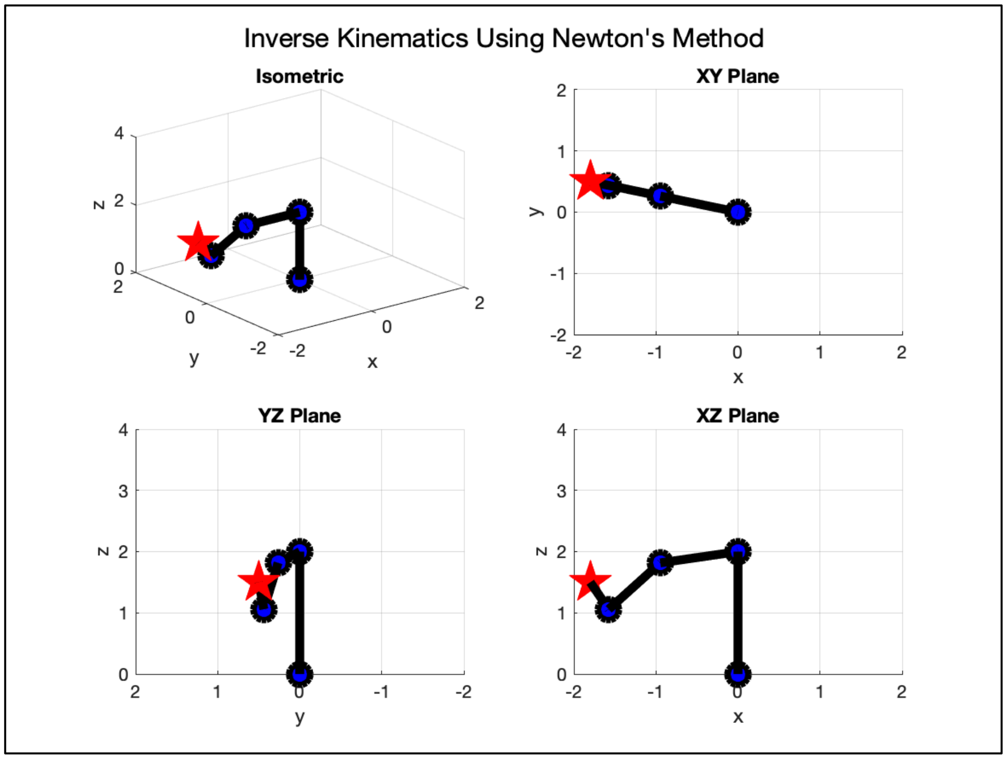

Figure 11. (Test 2 - Simulation 3) Joints 2 through 4 is in elbow-up configuration, while joint 3 through the endpoint is in elbow-down. Using a different endpoint position but the same initial guess as in Test 1 – Simulation 3, the numerical approach converges to the same configuration type.

In Test 2, the endpoint position is changed to another value, but is kept constant for its three simulations. Although the initial guess for one simulation in Test 2 is the same with its counterpart simulation in Test 1, the iteration count of the former and the latter are in nowhere near nor at least predictable compared to each other. Take Test 2 – Simulation 2 and Test 1 – Simulation 2 for example. The iteration counts are 888 and 774, respectively. The initial guess for both is the same, only the endpoint positions are different. This illustrates one of the downsides of using a numerical method: the unpredictability of iteration count which translates to the unpredictability of the function runtime. This means that, it doesn't matter if the initial guess is kept constant, if the endpoint position varies (or vice versa), the number of iterations is not guaranteed to be constant.

II. Geometric Approach (For Comparison)

Global Constants:

$a1 = 2\ units$

$a2 = a3 = 1\ unit$

$a4 = 0.5\ unit$

Note:

The theta values in the joint variable vector, q_f, are defined in degrees.

Legend:

$P = endpoint's\ position$

$runtime = the\ elapsed\ time\ by\ the\ geometric\ solution$

==============================Test 3================================

$$P = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1.5 \\ -1 \\ 2.5 \end{bmatrix}$$

Output:

$$q_f = \begin{bmatrix} -33.6901 \\ 66.7526 \\ -91.5121 \\ 24.7594 \end{bmatrix}$$

$runtime = 3.005\ ms$

Plot:

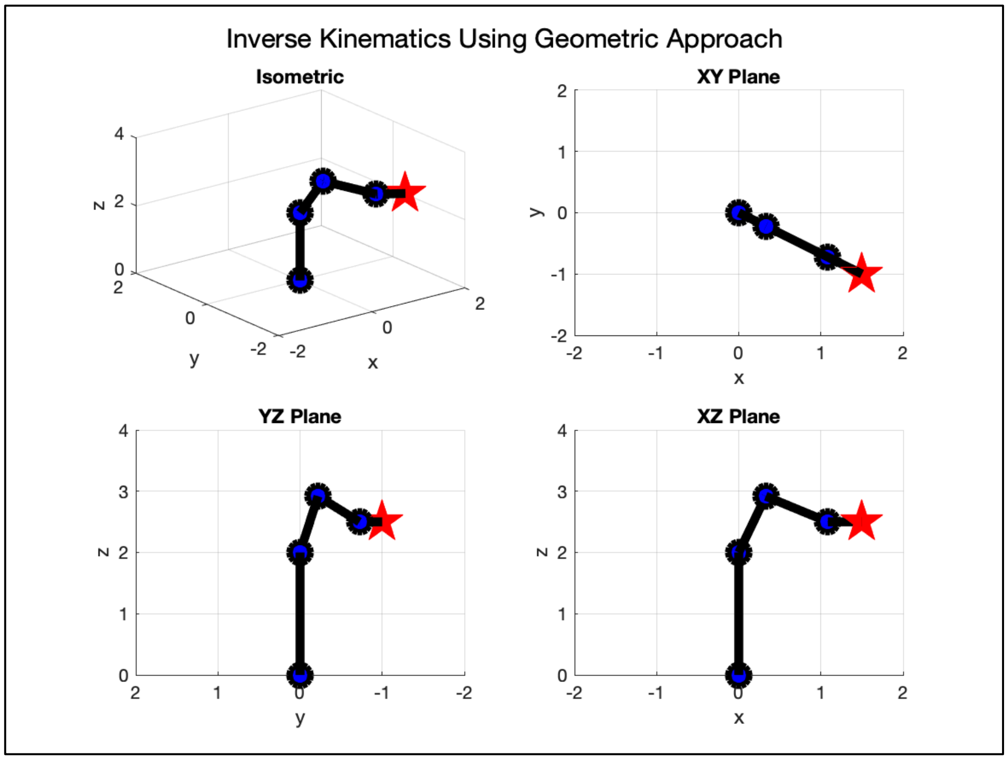

Figure 12. (Test 3) Joints 2 through 4 is in elbow-up configuration, while link a4 is parallel to the base x-axis. This configuration type is expected since the geometric set of solutions were derived based on this form as defined by the constraints discussed in the previous section.

In Test 3, it is important to observe the geometric form (configuration type) of the manipulator. Joints 2-4 are in elbow up configuration while a4 is parallel to the ground (base x-axis). This is absolutely expected since when the geometric set of solutions where derived, the manipulator is positioned in this manner. That means, the relationship of the joint variables with each other is pre-determined and is fixed with respect to the solution set. It is also noteworthy that the function runtime was greatly reduced by around 8 times if compared with any of the runtimes that resulted using the numerical approach in Test 1. This is also expected because the geometric approach directly relates the endpoint position values to each of the joint variables, thus, the computations are direct, and no iterations are needed.

===============================Test 4=================================

$$P = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -1.8 \\ 0.5 \\ 1.5 \end{bmatrix}$$

Output:

$$q_f = \begin{bmatrix} 164.4759 \\ 23.1785 \\ -86.5072 \\ 63.3287 \end{bmatrix}$$

$$runtime = 0.406\ ms$$

Plot:

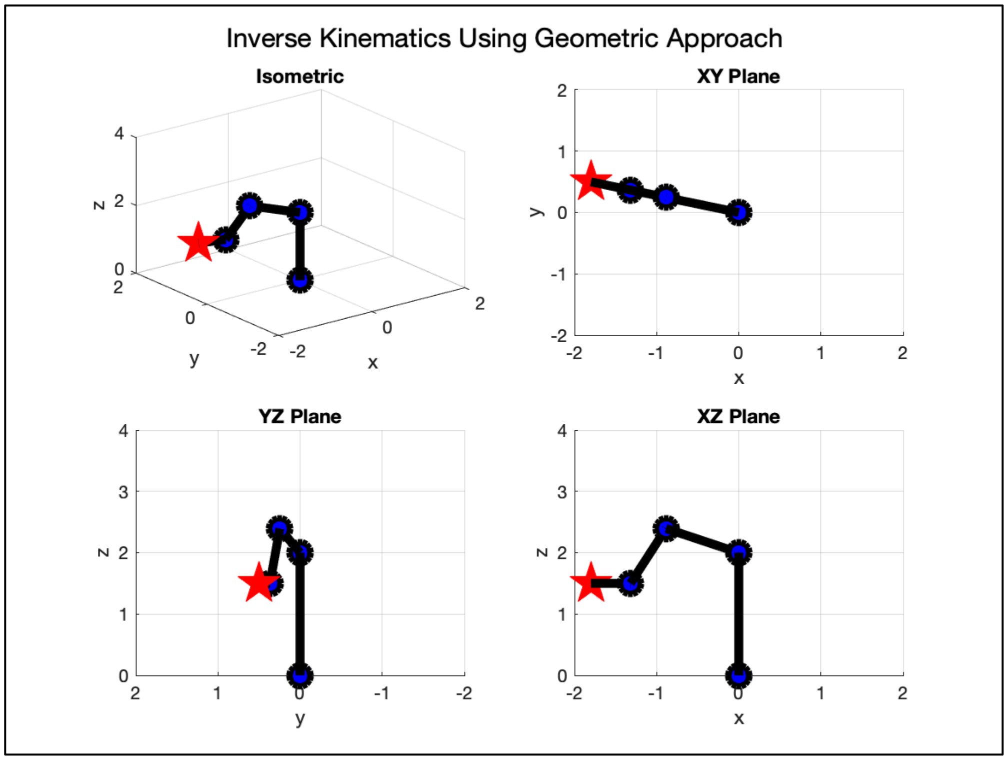

Figure 13. (Test 4) Joints 2 through 4 is in elbow-up configuration, while link a4 remains parallel to the base x-axis. The configuration type is the same as with Test 3. This illustrates the characteristic of the geometric approach which preserves the geometric form of the manipulator regardless of the endpoint position in space.

In Test 4, the pre-determined and fixed relationship of the joint variables with each other is confirmed, that is, when endpoint position is changed, the configuration type of the manipulator is preserved. This condition is always true if the same geometric solution set is used in the algorithm. In Figure 13, joints 2-4 are still in elbow up, and link a4 remains parallel to base x-axis. Also, the runtime remains to be relatively low compared to any of the runtimes that resulted in Test 2.

III. Tabulated Results

Table 2. This table summarizes the results of the two-part test (three simulations each) of the numerical approach averaging the values for the iteration runtime and iteration count.

Test	Simulation	Endpoint Position P	Initial Guess q_{guess} (in degrees)	Output				
				Computed Joint Variables q_f (in degrees)	Norm	Function Runtime (in ms)	Iteration Count $iter_{count}$	Absolute Error abs_{error}
1	1	[1.5;-1,2.5]	[-90;-60;-30;0]	[-33.6901;64.9475;-69.7874;-35.1833]	107.06	27.545	842	9.97E-07
	2		[0;0;0;0]	[-33.6900;-31.7057;64.2213;44.9195]	91.01	20.684	774	9.91E-07

	3		[-30;0;-30;60]	[-33.6901;16.7392;-32.6567;119.3921]	129.37	25.916	765	9.85E-07
2	1	[-1.8;0.5;1.5]	[-90;-60;-30;0]	[164.4759;32.8290;-69.3625;-26.8058]	183.47	25.977	820	9.86E-07
	2		[0;0;0;0]	[164.4759;31.1339;-64.4487;-36.0110]	182.95	26.555	888	9.91E-07
	3		[-30;0;-30;60]	[164.4759;-10.5806;-38.7526;111.4938]	202.72	33.244	906	9.83E-07
					Average	**26.654**	**833**	

Table 3. This table summarizes the results of the two-part test of the numerical approach averaging the values for the function runtime.

Test	Endpoint Position P		Output	
		Computed Joint Variables q_f (in degrees)	Norm	Function Runtime (in ms)
3	[1.5;-1,2.5]	[-33.6901;66.7526;-91.5121;24.7594]	120.74	3.005
4	[-1.8;0.5;1.5]	[164.4759;23.1785;-86.5072;-63.3287]	197.70	0.406
			Average	**1.706**

The average function runtime of the numerical approach, presented in Table 2, is almost 16 times the average function runtime of the geometric approach, presented in Table 3. As previously mentioned, this is expected since geometric approach carry out computations directly without the need to iterate multiple times. What this means for a computer is that less memory is used during computations, thus less power is consumed; that's for the calculations of the algorithm alone. But can the same be inferred about the total power that will be used by the manipulator to physically position itself to the endpoint?

Comparing the *Norm* columns of the data in Test 1 and that of Test 3, it can be observed that the magnitude in Test 3 is higher than in Test 1 - Simulations 1 & 2 and slightly lower than Test 1 – Simulation 3. The norm represents the degree by how much the manipulator needs to move or rotate its joints in the physical world. That translates into the power consumption of the manipulator and the time spent for its motion. The geometric approach can then be said to invoke a motion trajectory with power and time consumption that cannot be adjusted or optimized for every reachable point in space. The numerical approach, however, provides an optimized power and time consumption by providing option with the least norm. Tests 1 & 3, and Tests 2 & 4 are both reaching the same point in space, respectively. Notice that if the initial guesses in Simulations 2 of Tests 1 & 2 are fed into the algorithm, then the norm of the output q_f can be maintained at a minimum with respect to the point in space they are trying to reach. That will be more efficient if compared to their respective counterpart norms in Tests 3 & 4 where the geometric approach was used. Then, it can be said that the power and time consumed by the iterations of the numerical approach is compensated by the least norm solution it provides to optimize a supposed manipulator's motion in space.

Moreover, looking at the bigger picture, the numerical approach still performs great with an average function runtime of 26.654ms. Since any manipulator's role is to replace human effort, it is just right to relate the performance of the robot to that of a human in terms of the function runtime obtained in the tests.

One study found that the average human's response time to visual and auditory stimuli is around 247.60ms and 228.01ms, respectively. Since the human's response time was measured including the time when the tasks were actuated, it is not directly comparable to the figures in the above tables because the function runtime is only a portion of the manipulator's total response time (i.e., actuation time is excluded). However, the function runtime signifies the beginning of the robot's actuation. By correlation, if the goal of the manipulator's total response time is hypothetically set equal to that of a human, then, one can say that the numerical approach performed well by minimizing the function runtime to a very small portion, still leaving a large portion of the target response time for the robot's actuation.

If a method's advantage is solely based on how fast it can compute the output, then the geometric approach is clearly much preferable. Given the Newton's method's iteration count unpredictability, it may not be desirable for time-sensitive applications. This unpredictability is caused by the unmodeled criterion in selecting the initial guess which, until now, remains arbitrary; it's also the reason for the term 'guess.'

It is true that there are tasks where the computation time plays the most vital role such as in continuous path applications like welding and painting. In such tasks, the manipulator's configuration is usually fixed because the paths are pre-defined. Since the manipulator follows continuous motion, the algorithm must be able to compute the next position very fast because one delay may cause issues like overly heated and deformed metal surfaces or an unevenly distributed paint. For these applications, geometric methods are preferable. On the other hand, there are also tasks where a tiny fluctuation in delay does not invoke a significant effect like in heavy object lifting and inventory sorting applications. For those, geometric method or numerical method can either be used, but the latter should be more preferred given its ability to adapt to different kind of configurations without the need for separate set of solutions; that is on top of its optimizing capability to provide the least norm solution. That makes the manipulator more efficient and flexible in reaching and moving objects at different directions.

The ultimate deciding factor on which approach to use will be the geometry of the manipulator. If the number of DOFs is large (more than 3), then the geometric solution sets will be very hard to derive without bounding the function in a closed form (such as what was done for this paper's geometric solution set; refer to *Appendix (Part IV)*), which also explains why the relationship of the joint variables with each other are pre-determined and fixed for every geometric solution set. Defining constraints does not guarantee an easy derivation, and it limits the configuration of the manipulator to one type. For the above reasons, the numerical approach is preferable in manipulators with higher DOFs since it eliminates the need for complex derivations, and its generalized set of solutions is relatively easy to obtain following the DH convention.

CONCLUSION

The numerical approach implemented via Newton's method can solve the inverse kinematics of the 4-DOF RRRR manipulator while at the same time providing flexibility to adapt to different configuration types with just one generalized set of solutions obtained through DH convention. Compared to the geometric approach with an average function runtime of 1.706ms, the numerical method is relatively slower as represented by its average iteration count of 833 and average function runtime of 26.654ms. However, the geometric approach does not provide the same flexibility as the numerical approach and has been proven to preserve the configuration type from which its solution set was derived. If another configuration type is desired, then the geometric approach calls for the need to obtain another set of solutions which is very tedious especially for manipulators with high number of DOFs.

In terms of power and time consumption, the geometric approach spends less power and time for the output computation than the numerical method. However, the numerical method provides the least norm solution which in turn optimizes the power and time consumed by the manipulator during its supposed motion, compensating the shortcoming of the numerical method's computation part.

Correlating to the average human response times for visual and auditory stimuli of around 247.60ms and 228.01ms, respectively, the numerical method can be said to have still performed well by keeping the computation time minimal (less than 30ms range), leaving a large chunk of the target robot's response time for the actuation part.

For continuous motion tasks like welding and painting where time is of utmost importance and configurations are fixed, if the manipulator has relatively small number of DOF, the geometric approach is preferable because it optimizes the delay of output computation. The unpredictability of the function runtime for numerical method caused by its unmodeled criterion for the initial guess makes it ill-suited for such applications. However, for tasks like heavy object lifting and inventory sorting where motion efficiency and flexibility is required to reach more spaces and is not significantly affected by small fluctuation in delay, numerical method is preferable.

When the number of DOFs is large (more than 3), the derivation of geometric solution sets is very tedious without defining constraints. However, by defining constraints, the manipulator is stuck with one configuration type per geometric solution set. In such case where analyzing the manipulator's geometry is difficult, the use of numerical method is preferred because it only uses one generalized set of solutions whose derivation is greatly simplified by the DH convention.

REFERENCES

1. M.W. Spong, S. Hutchinson, & M. Vidyasagar, Robot Modeling and Control, *John Wiley & Sons, Inc.*, Hoboken, NJ, pg. 1-26, 73-118, 119-121 (2006)
2. Ross MacAusland, MATH 420: Advanced Topics in Linear Algebra, *University of Puget Sound*, Tacoma, WA, pg. 1-4 (2014)
3. A. Jain, R. Bansal, A. Kumar, & KD Singh (2015). A comparative study of visual and auditory reaction times on the basis of gender and physical activity levels of medical first year students, *International Journal of Applied and Basic Medical Research*, *5*(2), 124–127. https://doi.org/10.4103/2229-516X.157168

APPENDIX

I. MATLAB Function Code for the Numerical Approach: *InvKin_Newton.m*

```
function [qf, iter, abs_err] = InvKin_Newton(P, q_guess, a,tol)
%INVKIN_NEWTON Inverse Kinematics of 4-DoF 4-Revolute Robotic Arm
%   Computes for the configuration of the revolute joints given the final
%   position of the arm's end point via Newton's Method.
%   Input:
%       P   = vector containing the coordinates [x y z] of the arm's
%           desired end-point position
%       q_guess = initial guess for the joint variable values, thetas, in
%           in degrees
%       a   = vector of link lengths [a1; a2; a3; a4]
%       tol = convergence criterion
%   Output:
%       qf = vector of joint variables (thetas in degrees)
%       iter = number of iterations it took to arrive at the solution qf

% initializing iteration output count
iter = 0;

% Storing link lengths into separate variables for easier inline coding
a1 = a(1);
a2 = a(2);
a3 = a(3);
a4 = a(4);

% ERROR CHECK
b = (P(3) - a1);
c = (a2+a3+a4);
if b^2 > c^2
   iter = 0;
   abs_err = 0;
   qf = 'Geometrically unreachable space.'; % error message
   return
else
   a = sqrt(c^2 - b^2);
   a_des = sqrt(P(1)^2 + P(2)^2);
   if a_des > a
      iter = 0;
      abs_err = 0;
      qf = 'Outside reachable workspace.'; % error message
      return
   end
end

% Formula for function matrix
```

```
X = @(q) [cosd(q(1))*(a2*cosd(q(2))+a3*cosd(q(2)+q(3))+a4*cosd(q(2)+q(3)+q(4))); ...
     sind(q(1))*(a2*cosd(q(2))+a3*cosd(q(2)+q(3))+a4*cosd(q(2)+q(3)+q(4))); ...
     a1+a2*sind(q(2))+a3*sind(q(2)+q(3))+a4*sind(q(2)+q(3)+q(4))];

% Formula for jacobian matrix
J     =     @(q)     [-sind(q(1))*(a2*cosd(q(2))+a3*cosd(q(2)+q(3))+a4*cosd(q(2)+q(3)+q(4)))     -
cosd(q(1))*(a2*sind(q(2))+a3*sind(q(2)+q(3))+a4*sind(q(2)+q(3)+q(4)))                             -
cosd(q(1))*(a3*sind(q(2)+q(3))+a4*sind(q(2)+q(3)+q(4))) -a4*cosd(q(1))*sind(q(2)+q(3)+q(4));...
     cosd(q(1))*(a2*cosd(q(2))+a3*cosd(q(2)+q(3))+a4*cosd(q(2)+q(3)+q(4)))                        -
sind(q(1))*(a2*sind(q(2))+a3*sind(q(2)+q(3))+a4*sind(q(2)+q(3)+q(4)))                             -
sind(q(1))*(a3*sind(q(2)+q(3))+a4*sind(q(2)+q(3)+q(4))) -a4*sind(q(1))*sind(q(2)+q(3)+q(4));...
     0                                    a2*cosd(q(2))+a3*cosd(q(2)+q(3))+a4*cosd(q(2)+q(3)+q(4))
a3*cosd(q(2)+q(3))+a4*cosd(q(2)+q(3)+q(4)) a4*cosd(q(2)+q(3)+q(4))];

% NEWTON's METHOD loop
qtemp = q_guess;
while true
  qnext = qtemp - pinv(J(qtemp))*(X(qtemp)-P);
  iter = iter+1;

  % NORMALIZER: keep the angles equal or less than one full rev (360deg)
  % since the joints are unconstrained
  for i = 1:length(qnext)
    if abs(qnext(i)) > 360
      qnext = qnext./360;
      break;
    end
  end

  if norm(qnext-qtemp) < tol
    abs_err = norm(qnext-qtemp);
    qf = qnext;
    break;
  else
    qtemp = qnext;
  end

end

% OPTIMIZER: if absolute value of the angle is greater than 180deg, output
% the opposite direction (to save time by travesing shorter distance)
for i = 1:length(qf)
  if qf(i) < -180
    qf(i) = qf(i) + 360;
  elseif qf(i) > 180
    qf(i) = qf(i) - 360;
  end
end
```

II. MATLAB Function Code for the Geometric Approach: *InvKin_Geometric.m*

```
function qf = InvKin_Geometric(P, a)
%INVKIN_GEOMETRIC Inverse Kinematics of 4-DoF 4-Revolute Robotic Arm
%   Computes for the configuration of the revolute joints given the final
%   position of the arm's end point via geometric approach.
%   Constraints: theta4 negates the total angle formed by theta2 and theta
%   3 to position the last link in parallel to the base x-axis.
%   Input:
%       P   = vector containing the coordinates [x y z] of the arm's
%             desired end-point position
%       a   = vector of link lengths [a1; a2; a3; a4]
%   Output:
%       qf = vector of joint variables (thetas in degrees)

% Storing link lengths into separate variables for easier inline coding
a1 = a(1);
a2 = a(2);
a3 = a(3);
a4 = a(4);

% Storing endpoint positions into separate variables for easier coding
xe = P(1);
ye = P(2);
ze = P(3);

% ERROR CHECK
b = (P(3) - a1);
c = (a2+a3+a4);
if b^2 > c^2
    qf = 'Geometrically unreachable space.'; % error message
    return
else
    a = sqrt(c^2 - b^2);
    a_des = sqrt(P(1)^2 + P(2)^2);
    if a_des > a
        qf = 'Outside reachable workspace.'; % error message
        return
    end
end

% solving for theta1
theta1 = atan2(ye,xe);

% solving for theta3
r = sqrt(xe^2 + ye^2);
r2 = r - a4;
z2 = ze - a1;
```

```
h = sqrt(r2^2 + z2^2);
beta = acos((a2^2+a3^2-h^2)/(2*a2*a3));
theta3 = -(pi - beta);

% solving for theta2
gamma = atan2(z2,r2);
alpha = acos((h^2+a2^2-a3^2)/(2*a2*h));
theta2 = alpha + gamma;

% solving for theta4
theta4 = -(theta2 + theta3);

qf = [theta1; theta2; theta3; theta4].*180/pi;
```

III. MATLAB Driver Script for the Manipulator Simulation: *driver.m*

```
clc
format short

% desired endpoint position in 3D [x y z]
P = [-1.8; 0.5; 1.5];

% column vector of initial guess [theta1; theta2; theta3; theta4]
% in degrees
q_guess = [-30; 0; -30; 60];

% link lengths [a1 a2 a3 a4]
a = [2 1 1 0.5];

% storing link lengths into separate variables for easier inline coding
a1 = a(1);
a2 = a(2);
a3 = a(3);
a4 = a(4);

% convergence criterion
tol = 10^-6;

% Inverse Kinematics figure titles
fig_title = {'Inverse Kinematics Using Newton''s Method', 'Inverse Kinematics Using Geometric Approach'};

for j = 1:2
  disp('==================================================')
  disp(newline)
  disp(strcat(fig_title{j},':'))
  disp(newline)
  % Inverse Kinematics & runtime measurement of the numerical method
  if j == 1
    tic
    [qf, iter_count, abs_err] = InvKin_Newton(P,q_guess,a,tol);
    toc
  else
    tic
    qf = InvKin_Geometric(P,a);
    toc
  end

  % Error Handling
  % if qf is numeric, plot the output
  % else, output the error message
  if isnumeric(qf)
    qf
```

```
if j == 1
   iter_count
   abs_err
end

% defining each robotic arm's link endpoints
origin = [0 0 0];
link1 = [0 0 a1];
link2 = [a2*cosd(qf(1))*cosd(qf(2)) a2*sind(qf(1))*cosd(qf(2)) ...
   a2*sind(qf(2))+a1];
link3 = [cosd(qf(1))*(a2*cosd(qf(2))+a3*cosd(qf(2)+qf(3))) ...
   sind(qf(1))*(a2*cosd(qf(2))+a3*cosd(qf(2)+qf(3))) ...
   a1+a2*sind(qf(2))+a3*sind(qf(2)+qf(3))];
link4 = [cosd(qf(1))*(a2*cosd(qf(2))+a3*cosd(qf(2)+qf(3))+a4*cosd(qf(2)+qf(3)+qf(4))) ...
   sind(qf(1))*(a2*cosd(qf(2))+a3*cosd(qf(2)+qf(3))+a4*cosd(qf(2)+qf(3)+qf(4))) ...
   a1+a2*sind(qf(2))+a3*sind(qf(2)+qf(3))+a4*sind(qf(2)+qf(3)+qf(4))];

% storing the endpoints into one array
arm = {origin link1 link2 link3 link4};

figure(j)
% plotting the robotic arm
subplot(2,2,1)
for i = 2:length(arm)
   if i == length(arm)
      plot3([arm{i-1}(1) arm{i}(1)],[arm{i-1}(2) arm{i}(2)], ...
         [arm{i-1}(3) arm{i}(3)], '-k', 'LineWidth', 5)
      hold on
      plot3(arm{i}(1), arm{i}(2), arm{i}(3), 'pr', 'Markersize', 25, 'MarkerFaceColor', 'red')
      hold on
   else
      plot3([arm{i-1}(1) arm{i}(1)],[arm{i-1}(2) arm{i}(2)], ...
         [arm{i-1}(3) arm{i}(3)], '-ok', 'LineWidth', 5, 'MarkerSize', 12, 'MarkerFaceColor', 'blue')
      hold on
   end

end
title('Isometric')
hold off
grid on
xlabel('x')
ylabel('y')
zlabel('z')
xlim([-2 2])
ylim([-2 2])
zlim([0 4])

subplot(2,2,2)
```

```
for i = 2:length(arm)
   if i == length(arm)
      plot3([arm{i-1}(1) arm{i}(1)],[arm{i-1}(2) arm{i}(2)], ...
         [arm{i-1}(3) arm{i}(3)], '-k', 'LineWidth', 5)
      hold on
      plot3(arm{i}(1), arm{i}(2), arm{i}(3), 'pr', 'Markersize', 25, 'MarkerFaceColor', 'red')
      hold on
   else
      plot3([arm{i-1}(1) arm{i}(1)],[arm{i-1}(2) arm{i}(2)], ...
         [arm{i-1}(3) arm{i}(3)], '-ok', 'LineWidth', 5, 'MarkerSize', 12, 'MarkerFaceColor', 'blue')
      hold on
   end

end
title('XY Plane')
hold off
grid on
xlabel('x')
ylabel('y')
zlabel('z')
xlim([-2 2])
ylim([-2 2])
zlim([0 4])
view(0,90)

subplot(2,2,3)
for i = 2:length(arm)
   if i == length(arm)
      plot3([arm{i-1}(1) arm{i}(1)],[arm{i-1}(2) arm{i}(2)], ...
         [arm{i-1}(3) arm{i}(3)], '-k', 'LineWidth', 5)
      hold on
      plot3(arm{i}(1), arm{i}(2), arm{i}(3), 'pr', 'Markersize', 25, 'MarkerFaceColor', 'red')
      hold on
   else
      plot3([arm{i-1}(1) arm{i}(1)],[arm{i-1}(2) arm{i}(2)], ...
         [arm{i-1}(3) arm{i}(3)], '-ok', 'LineWidth', 5, 'MarkerSize', 12, 'MarkerFaceColor', 'blue')
      hold on
   end

end
title('YZ Plane')
hold off
grid on
xlabel('x')
ylabel('y')
zlabel('z')
xlim([-2 2])
ylim([-2 2])
```

```
        zlim([0 4])
        view(-90,0)

        subplot(2,2,4)
        for i = 2:length(arm)
            if i == length(arm)
                plot3([arm{i-1}(1) arm{i}(1)],[arm{i-1}(2) arm{i}(2)], ...
                    [arm{i-1}(3) arm{i}(3)], '-k', 'LineWidth', 5)
                hold on
                plot3(arm{i}(1), arm{i}(2), arm{i}(3), 'pr', 'Markersize', 25, 'MarkerFaceColor', 'red')
                hold on
            else
                plot3([arm{i-1}(1) arm{i}(1)],[arm{i-1}(2) arm{i}(2)], ...
                    [arm{i-1}(3) arm{i}(3)], '-ok', 'LineWidth', 5, 'MarkerSize', 12, 'MarkerFaceColor', 'blue')
                hold on
            end

        end
        title('XZ Plane')
        hold off
        grid on
        xlabel('x')
        ylabel('y')
        zlabel('z')
        xlim([-2 2])
        ylim([-2 2])
        zlim([0 4])
        view(0,0)
        sgtitle(fig_title{j})

    else
        Error = qf
    end

end
```

IV. Derivation of the Geometric Set of Solutions

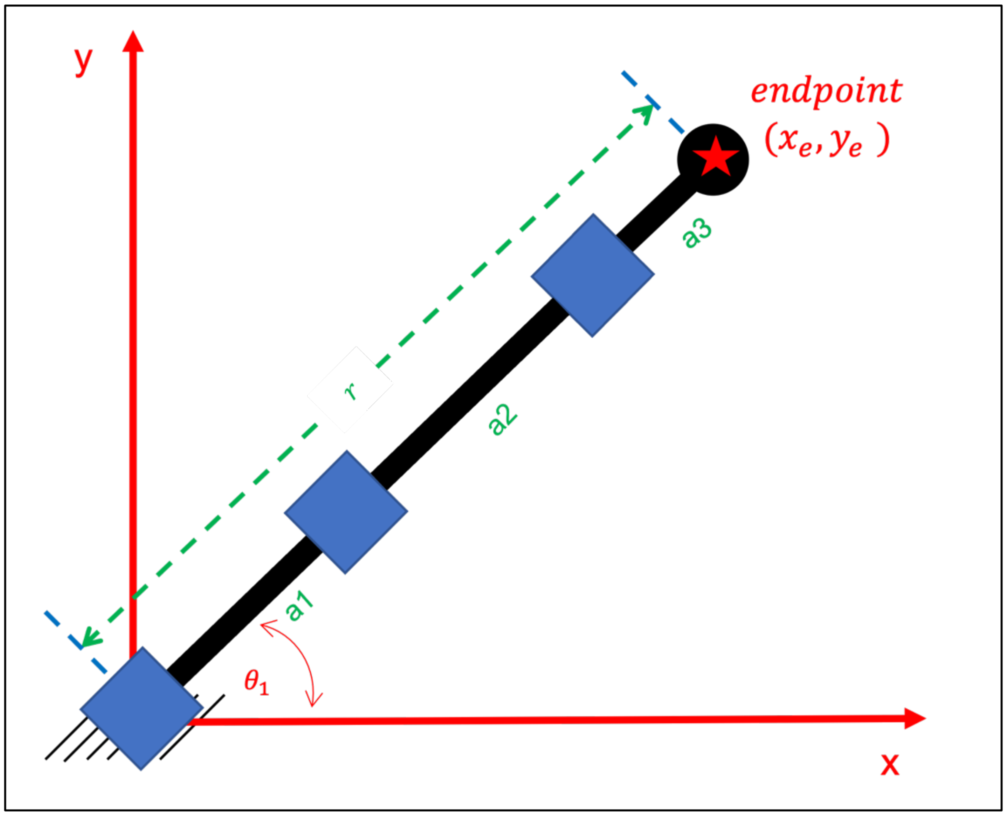

Figure 14. Looking through the top view of the 4-DOF RRRR Manipulator to derive equation for θ_1.

Solving for θ_1:

$$\theta_1 = tan^{-1}\left(\frac{y_e}{x_e}\right) \quad (eq.4)$$

$$r = \sqrt{x_e{}^2 + y_e{}^2}$$

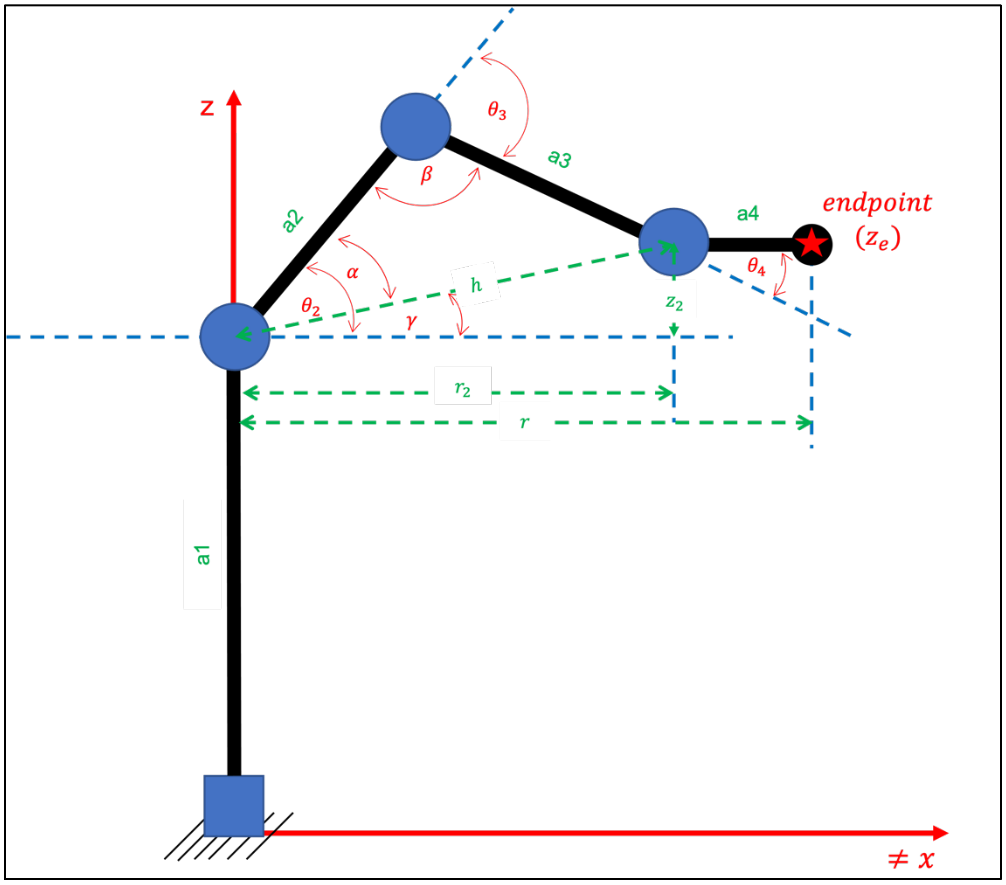

Figure 15. Looking through the side view and drawing similar triangles & intermediate angles to derive equations for θ_2, θ_3, and θ_4.

$$r_2 = r - a4$$
$$z_2 = z_e - a1$$
$$h = \sqrt{{r_2}^2 + {z_2}^2}$$

Solving for θ_2:

$$\gamma = tan^{-1}\left(\frac{z_2}{r_2}\right)$$
$$\propto = cos^{-1}\left(\frac{h^2 + a2^2 - a3^2}{2 \times a2 \times h}\right)$$
$$\theta_2 = \propto + \gamma$$

$$\theta_2$$
$$= cos^{-1}\left(\frac{(\sqrt{{x_e}^2 + {y_e}^2} - a4)^2 + (z_e - a1)^2 + a2^2 - a3^2}{2 \times a2 \times \sqrt{(\sqrt{{x_e}^2 + {y_e}^2} - a4)^2 + (z_e - a1)^2}}\right) \times tan^{-1}\left(\frac{z_e - a1}{\sqrt{{x_e}^2 + {y_e}^2} - a4}\right) \quad (eq. 5)$$

Solving for θ_3:

$$\beta = cos^{-1}\left(\frac{a2^2 + a3^2 - h^2}{2 \times a2 \times a3}\right)$$

$$\theta_3 = -(\pi - \beta)$$

$$\theta_3 = -\left[\pi - cos^{-1}\left(\frac{a2^2 + a3^2 - \left(\sqrt{x_e{}^2 + \ y_e{}^2} - a4\right)^2 + \ (z_e - a1)^2}{2 \times a2 \times a3}\right)\right] \quad (eq. 6)$$

Solving for θ_4:

$$\theta_4 = -(\theta_2 + \ \theta_3) \quad (eq. 7)$$