ABSTRACT

BRYNILDSEN, ALEXANDER GREGORY. Optimization Algorithms and Digital Signal Processing For Inverse Kinematics and Upper Extremity System Identification. (Under the direction of Dr. Katherine Saul).

Upper extremity stiffness is a biomechanical parameter with implications for rehabilitation engineering, human operator dynamics and robotics. It is also a multidisciplinary topic, relying on digital signal processing and inverse kinematics. This thesis examines three numerical inverse kinematics solution techniques as applied a robotic end effector and human upper extremity. A method for measuring limb stiffness in the frequency domain using a seven-axis robot and digital signal processing techniques is also outlined. Through simulation, it was verified that the BFGS algorithm is preferable for situations with high initial estimate error, while the NR and LM methods are superior when minimizing computing cost is a priority as in real-time applications such as human controlled robots in rehabilitation engineering. Synthetic limb stiffness data was found to produce plots similar to those found in other research, however, the collection frequency of experimental data was too low to capture the full frequency range needed for dynamic modeling.

Optimization Algorithms and Digital Signal Processing For Inverse Kinematics and Upper Extremity System Identification

by
Alexander Gregory Brynildsen

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Mechanical Engineering

Raleigh, North Carolina
2022

APPROVED BY:

Dr. Andre Mazzoleni

Dr. Matthew Bryant

Dr. Katherine Saul
Chair of Advisory Committee

ACKNOWLEDGEMENTS

I would like to thank my parents who have been endlessly supportive; my advisor, Dr. Katherine Saul, who I have studied with with since my sophomore year of my bachelor's degree; my committee consisting of Dr. Matthew Bryant and Dr. Andre Mazzoleni; and my employer, Rockwell Automation, for providing many opportunities to me during my time in graduate school.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

Dynamic joint stiffness is the physical relation between joint positions and angles in humans and the resulting resistance to forces and moments due to variations in muscle lengths and posture [1]. It is a necessary parameter for developing mathematical models of human locomotion and motor tasks, and is important to understanding human movement and posture selection. Dynamic joint stiffness has applications in a range of engineering and medical related fields of study including motor control, rehabilitation engineering and neuromuscular disease, robotics, and human operator dynamics [2], making it an important research topic.

Both inverse kinematics (IK) and digital signal processing (DSP) are critical for developing mathematical models of the upper extremity as demonstrated by the flowchart in Figure 1.1. While stiffness is generally measured in Cartesian space, IK is necessary to transform the position and orientation of the final coordinate frames (in this case, the human wrist and hand) to joint space such that useful physical models can be created. Transforming stiffness to joint space also facilitates the evaluation of functional performance and stability post-injury, and enables analysis of self-selected posture based on expected load direction and magnitude.

Several options are available for determining the angle inputs required to achieve a desired end effector or hand configuration: analytical or closed form, numerical, and semi-analytical [3]. For a Cartesian coordinate frame to achieve any desired position and orientation in 3D space, at least six degrees of freedom are required: three for the x, y and z Cartesian position, and another three for the rotational elements, R_{ij} [4]. In the case of a six axis robot, the first three joint inputs (q_{1-3}) dictate the position of the end effector, while the last three (q_{4-6}) determine its orientation [5]. For IK and stiffness measurements, the human upper extremity can be simplified to a seven-axis model as demonstrated by Figure 1.2.

The addition of a seventh axis in the Kuka iiwa LBR 14 R820 (hereafter referred to as "LBR iiwa") means that the robot has redundant degrees of freedom, resulting in unlimited IK solutions to achieve a desired end effector position and configuration. This redundancy gives the robot several advantages, enabling it to avoid singularities in IK solutions, better avoid obstacles within a given workspace, and increase manipulability of the end effector [6]. Furthermore, the redundant joint enables the robot to achieve anthropomorphic configurations [7]. The anthropomorphism of the LBR iiwa makes it an excellent candidate for the study and development of inverse kinematics, as it can mimic human movement with

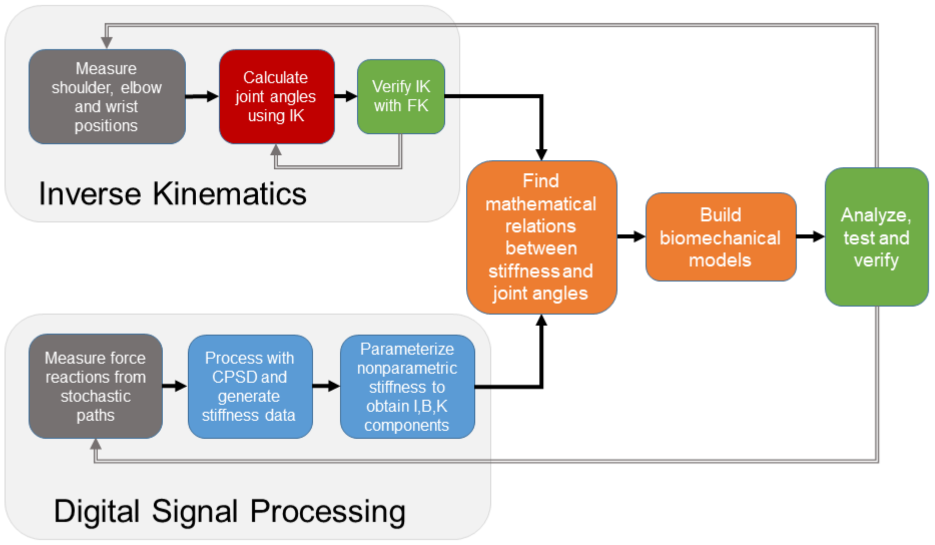

Measure shoulder, elbow and wrist positions
Calculate joint angles using IK
Verify IK with FK
Inverse Kinematics
Find mathematical relations between stiffness and joint angles
Build biomechanical models
Analyze, test and verify
Measure force reactions from stochastic paths
Process with CPSD and generate stiffness data
Parameterize nonparametric stiffness to obtain I,B,K components
Digital Signal Processing

Figure 1.1: Proposed workflow for modeling the upper extremity.

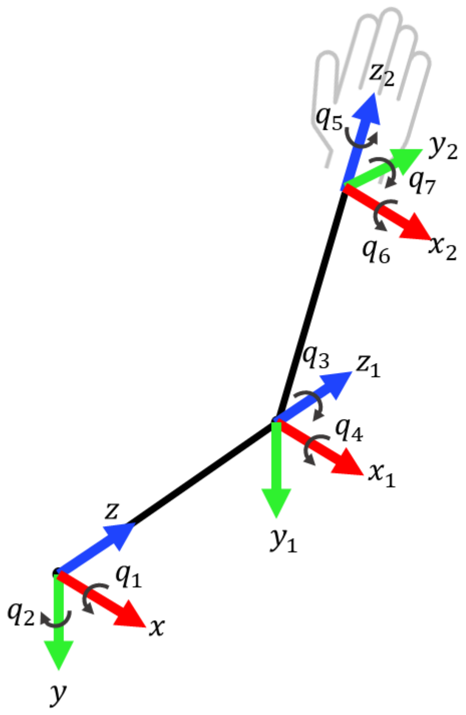


Figure 1.2: Proposed coordinate frames for the upper extremity.

high accuracy. The inverse kinematics algorithms provided in this thesis can be generally applied to either a redundant robot or a human upper extremity.

Previous research has noted that the position and orientation of the elbow is an important factor in human grasping tasks [8, 9, 10]. For a human controlling a robot, it is more intuitive to specify the position of their elbow as an input to a robot than the input angles [7]. Furthermore, the position of the elbow can be easily measured using an infrared camera or electromagnetic tracking system [8]. The LBR iiwa serves as a useful tool for developing inverse kinematics algorithms, as all of the required parameters are known, and IK results can be quickly verified using forward kinematics or the MATLAB robotics toolbox. Measuring the human upper extremity to obtain required parameters can be difficult not only because measurement tools such as infrared cameras or electromagnetic devices introduce error, but also because joint limits vary on an individual basis and must be determined experimentally. For this reason, numerical solutions were selected for inverse kinematics as applied to the LBR iiwa and human upper extremity, with elbow position specification enabled for the upper extremity model.

Stochastic inputs are advantageous in the study of limb stiffness because they provide sufficient input power over a range of frequencies, meaning that they can be used to gather data using relatively short data segments (sixty seconds or less, depending on collection frequency) [2]. The shortness of trials is important because it allows for a range of posture experiments without fatiguing the subject. While the stochastic input generates useful data, the fact that it inherently covers a range of frequencies means that data must be processed to generate trends for analysis. To achieve this, a Fast Fourier Transform (FFT) can be used to transform time domain records into an infinite basis of sinusoids [11], turning the input and output data into a useful tool that can predict the response of a system to an input using the frequency response.

FFT analysis begins with factoring a given data set into smaller sets of data, greatly reducing computing effort [12]. Nonparametric joint stiffness can then be calculated in Cartesian space using an impedance matrix of transfer functions generated using cross spectral density analysis [13]. The cross power spectral density employs FFTs to generate trends between inputs and outputs, which is the goal of system identification and stiffness measurement detailed in this thesis. The most important assumption is that the limb behaves as a linear system for small perturbations, and since the displacements used in experimentation were on the order of ± 1.5 mm, this is reasonable.

This thesis is divided into two main sections: the first presents a variety of forward kinematics and IK algorithms applicable to a seven-axis robot or a human upper extremity, and the second presents a method for analyzing human upper extremity stiffness to perform system identification using DSP techniques. Equations needed to produce the algorithms are provided, and a comparison of each is presented along with an example of an optimization algorithm applied to a 2DoF system. A link to complete MATLAB code is available at https://github.com/abrynildsen/IK.

CHAPTER

2

FORWARD KINEMATICS

2.1 Forward Kinematics

Forward kinematics (FK) refers to the mathematical methods used to calculate the Cartesian coordinate position and orientation of a robotic end effector using only the geometry of the robot and input angles to the servos. FK is necessary to develop objective equations and Jacobian matrices for IK optimization algorithms and it is necessary for verifying IK results. The Denavit-Hartenberg (DH) convention [14] is used almost exclusively for calculating robot arm positions and orientations because it can be used for any number of robotic arms and accounts for angular and Cartesian offsets of the joints, meaning it can be applied to any robot or human upper extremity provided the correct parameters are provided. Several recent works have examined methods for calculating end effector orientation and position specific to the seven-axis LBR iiwa using the DH method [15, 16, 17].

To calculate the end effector orientation and position of a given robot, the following parameters and variables are required:

- angular servo inputs for each of the joints, θ_i
- lengths of each of the robot arms, d_i
- rotation of the current coordinate frame of the robot joint about the x-axis from the previous coordinate frame, α_i
- x-axis offsets of the robot joint from the previous axis, a_i

where i denotes a given arm in the robot. To demonstrate the importance of these parameters, a graphic displaying the joint locations and arm lengths d_i of the robot can be found in Figure 2.1 and a graphic displaying the coordinate transformations and axis offsets is shown in Figure 2.2.

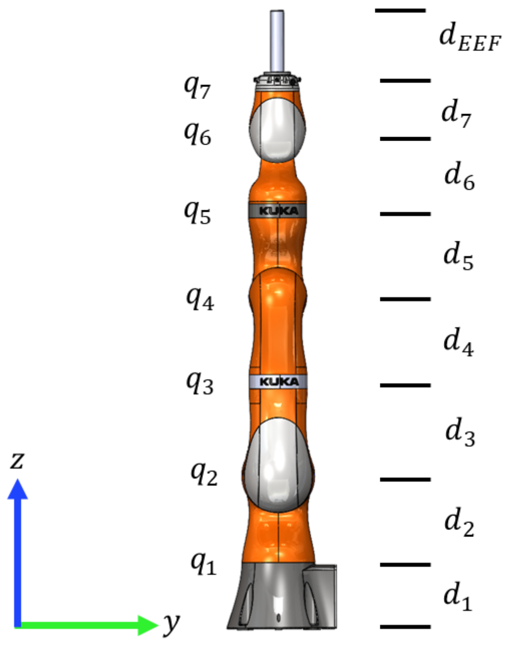


Figure 2.1: Joint locations and arm dimensions of the LBR iiwa.

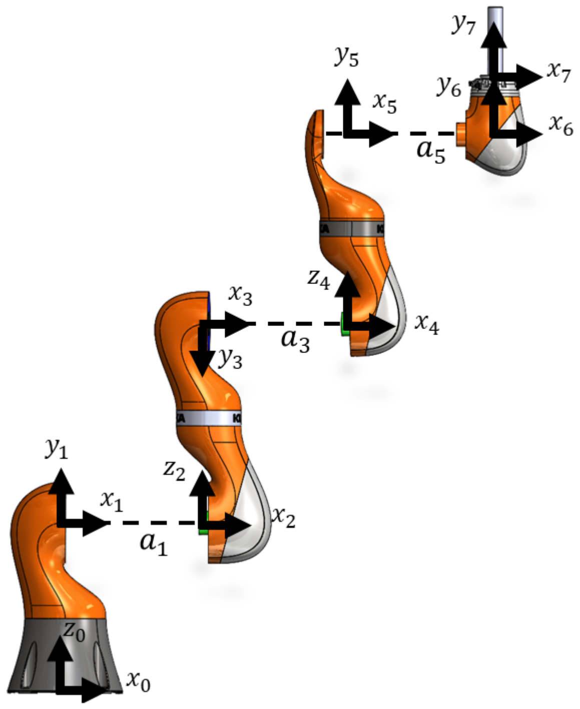


Figure 2.2: Exploded view of the LBR iiwa to demonstrate axis offsets and coordinate transformations.

The rotation matrix about the z_i axis for each arm of the robot is calculated as follows

$$\mathbf{R}_{z_i} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The translation matrix in the z_i direction for each arm of the robot is given by

$$\mathbf{T}_{z_i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The translation matrix in the x_i direction is calculated

$$\mathbf{T}_{x_i} = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

And lastly, the rotation matrix about the x_i axis for each arm of the robot is calculated

$$\mathbf{R}_{x_i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

These matrices are multiplied together to form the following Denavit-Hartenberg matrix which can be used to calculate the coordinate frame of one axis of the robot with respect to its previous coordinate frame.

$$\begin{aligned} \mathbf{T}_i &= \mathbf{R}_{z_i}\mathbf{T}_{z_i}\mathbf{T}_{x_i}\mathbf{R}_{x_i} \\ &= \begin{bmatrix} \cos\theta_i & -\cos\alpha_i\sin\theta_i & \sin\alpha_i\sin\theta_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\alpha_i\sin\theta_i & -\sin\alpha_i\cos\theta_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

To calculate the position and orientation of the end effector of the LBR iiwa, the DH matrix must be calculated for all seven arms and then all seven matrices must be multiplied together to transform the coordinate frame from the base to the end effector. This operation is easy to perform in MATLAB or similar computational software using a `for` loop.

$$\mathbf{T}_{EE} = \mathbf{T}_1\mathbf{T}_2\mathbf{T}_3\mathbf{T}_4\mathbf{T}_5\mathbf{T}_6\mathbf{T}_7$$

Which has the following parameters

Table 2.1: DH parameters for the LBR iiwa.

a_i	α_i	d_i	θ_i
0	90	0.36	θ_1
0	-90	0	θ_2
0	-90	0.42	θ_3
0	90	0	θ_4
0	90	0.40	θ_5
0	-90	0	θ_6
0	0	$0.1199 + L_{EE}$	θ_7

Where L_{EE} is the length of the end effector.

CHAPTER

3 INVERSE KINEMATICS

3.1 Numerical Approach to IK

Numerical IK is an optimization problem wherein the variables being solved for are the input angles of the coordinate frames and the output is the position and orientation of a given arm of a robot or human limb, which is typically the end effector or hand. MATLAB offers a robotics toolbox that can be used to solve FK and IK problems in addition to visualizing coordinate frame transformations. The toolbox is especially useful for verifying the accuracy of forward kinematics DH parameters and IK algorithms and was used extensively throughout the work completed for this thesis.

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) gradient projection algorithm is assigned by default when the function `inverseKinematics` is called in MATLAB. This algorithm is highly robust and handles large error in initial guesses well, but converges slower than other algorithms such as the Levenberg-Marquardt (LM) method [18]. For the stochastic perturbations used in this experiment, relative joint velocities and displacements are low. As a result, an initial guess for angular joint inputs at a given time step can be based on the previous time step. Therefore, the LM method, commonly referred to as the damped least squares method [19], can be employed to yield suitable results with a faster convergence rate [20]. The LM method is advantageous for solving IK problems because it does not require a pseudoinverse of the Jacobian, which leads to singularities in other methods [21] such as the Newton Raphson (NR) method.

3.1.1 NR Method

The NR method is the simplest numerical method to implement for IK solutions. NR is similar to the LM method in that it uses the Jacobian matrix of a set of DH configuration equations to minimize error but differs because it does not have a damping term, λ. This means that the method can fail due to singularities in the Jacobian matrices [22] which occur when the determinant of the Jacobian is zero. This means that an inverse cannot exist, and the problem cannot be solved. However, for small displacement situations where initial estimates of robot configurations have relatively low error, it can be a very fast and efficient solution. Detailed derivations for this method are available in [23, 24].

As with the LM method, the most computationally expensive component of the NR method is calculating the Jacobian. The algorithm outlined below uses an exact Jacobian, as the cost is relatively low once all parameters have been defined for a given robot, in our case the LBR iiwa. For algorithms in which values for d, α and a are meant to be variable inputs, an approximate Jacobian is recommended. Approximation methods are available [25] in situations where computation time is a concern.

As with all numerical methods, the NR method requires an initial guess for the servo input angles q_0 and a desired DH configuration for the end effector, T_d. The updating equation for q is given by

$$\delta q_{k+1} = \mathbf{J}^{+} e_k \tag{3.1}$$

Where k is the current step and $\mathbf{J}^{+}$ is calculated

$$\mathbf{J}^{+} = \left(\mathbf{J}^{T}\mathbf{J}\right)^{-1}\mathbf{J}^{T}$$

Which can be reliably calculated using the MATLAB command `pinv`. The residual e_k is given by

$$e_k = T_d - T(q_k) \tag{3.2}$$

Where T_d, $T(q_k)$ are the desired and current step DH configuration matrices, input as a 12 by 1 vector so that they can be multiplied by the pseudoinverse of the Jacobian, $\mathbf{J}^{+}$. The value for q is then updated for the next iteration

$$q_{k+1} = q_k + \delta q_{k+1} \tag{3.3}$$

Clearly, this method is quite easy to implement using a `while` loop, which can exit when the following condition is met:

$$\frac{|T\left(q_k\right) - T\left(q_{k+1}\right)|}{|T\left(q_k\right)|} \leq \epsilon \tag{3.4}$$

where ϵ is defined as an acceptable error.

3.1.2 LM Method

The LM algorithm iteratively minimizes a nonlinear least squares function by interpolating between the Taylor series and gradient methods [26], thereby taking advantage of the strengths of both. While both the LM and NR methods iterate based on some variation of an inverted Jacobian, the LM method differs from NR in that it employs a weighted matrix multiplied by an objective function and implements as a damping factor λ, which avoids algorithm failure due singularities in the pseudoinverse used in the NR method. The LM method begins with a calculation of the residual as given by equation 3.2. This residual is then multiplied with a matrix of weights to form the objective function to be minimized by the algorithm:

$$\phi = \frac{1}{2} e^T \mathbf{W} e \tag{3.5}$$

The Jacobian of the objective function is then calculated using equation and multiplied into the following [27]

$$\left(\mathbf{J}^T \mathbf{J} + \lambda_n^2 \mathbf{I}\right) \delta_n = \mathbf{J}^T e \tag{3.6}$$

in which the left hand side is inverted to yield the update increment

$$\delta_n = \left(\mathbf{J}^T \mathbf{J} + \lambda_n^2 \mathbf{I}\right)^{-1} \mathbf{J}^T e \tag{3.7}$$

The angles q_{1-7} are then updated for step $n+1$ by the following

$$q_{n+1} = \delta_n + q_n \tag{3.8}$$

Where λ is updated at each iteration by the following criteria [26]

$$\lambda_n = \begin{cases} \frac{\lambda_{n-1}}{\nu} & \text{if} \quad \phi\left(\frac{\lambda_{n-1}}{\nu}\right) \le \phi_n \\ \lambda_{n-1} & \text{if} \quad \phi\left(\frac{\lambda_{n-1}}{\nu}\right) > \phi_n \quad \text{and} \quad \phi(\lambda_{n-1}) \le \phi_n \\ \lambda_{n-1}\,\nu^w & \text{if} \quad \phi\left(\frac{\lambda_{n-1}}{\nu}\right) > \phi_n \quad \text{and} \quad \phi(\lambda_{n-1}) > \phi_n \end{cases} \tag{3.9}$$

where w denotes the smallest exponent for the third condition to be met. This loop is repeated until the exit condition is met

$$\phi \le \phi_{max} \tag{3.10}$$

Where ϕ_{max} is the largest acceptable value for the objective function.

3.1.3 BFGS Method

Another numerical solution that may be preferable to LM in situations with high initial estimate error can be found with the BFGS algorithm [28, 29]. This algorithm does not require the calculation of a Jacobian, using instead a gradient of the objective function given by 3.23 and the Hessian matrix which is defined by the following for a 7 DoF robot:

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 \phi}{\partial q_1^2} & \frac{\partial^2 \phi}{\partial q_1 \partial q_2} & \cdots & \frac{\partial^2 \phi}{\partial q_1 \partial q_7} \\ \frac{\partial^2 \phi}{\partial q_2 \partial q_1} & \frac{\partial^2 \phi}{\partial q_2^2} & \cdots & \frac{\partial^2 \phi}{\partial q_2 \partial q_7} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 \phi}{\partial q_7 \partial q_1} & \frac{\partial^2 \phi}{\partial q_7 \partial q_2} & \cdots & \frac{\partial^2 \phi}{\partial q_7^2} \end{bmatrix} \tag{3.11}$$

Since the Hessian matrix involves one more derivative than the Jacobian for each index, it can be computationally expensive for systems with higher degrees of freedom. To improve solver convergence time, an approximation is used. To apply the BFGS method to a non-linear system, the following are needed:

- an initial estimate for the angle inputs to the servos, denoted q^0
- a convergence tolerance, ϵ
- an initial estimate of the Hessian, which can be the identity matrix
- an initial estimate for the step length, α
- an initial estimate for the step length scaling factor, β

The following algorithm can be placed in a `while` loop which exits upon the following condition:

$$\|\nabla\phi\| \leq \epsilon \tag{3.12}$$

The following steps are placed in the `while` loop until condition 3.12 is reached.

1. compute a search p_k direction to calculate the proceeding step

$$p_k = -\mathbf{H}_k \nabla\phi_k \tag{3.13}$$

2. update the servo angles q for the next step using the following

$$q_{k+1} = q_k + \alpha_k p_k \tag{3.14}$$

   where α_k is the step length, which must meet Wolfe's criteria [29] (see following section for details)

3. calculate the servo angle residual s_k and the objective function residual y_k between the updated step $k+1$ and the current step k, both of which are used to update the Hessian estimate for the next step

$$s_k = q_{k+1} - q_k \tag{3.15}$$

$$y_k = \nabla\phi_{k+1} - \nabla\phi_k \tag{3.16}$$

4. update the Hessian estimate using the following

$$\mathbf{H}_{k+1} = \left(\mathbf{I} - \rho_k s_k y_k^T\right)\mathbf{H}_k\left(\mathbf{I} - \rho_k y_k s_k^T\right) + \rho_k s_k s_k^T \tag{3.17}$$

   where the term ρ_k is defined

$$\rho_k = \frac{1}{y_k^T s_k} \tag{3.18}$$

Within the `while` loop outlined above, Wolfe's criteria [29], was used to update the step size α to an acceptable size for each iteration. First, we define Wolfe's criteria in two steps:

1. the first condition that must be met for a given value of α is defined by the following:

$$\phi\left(q_k + \alpha_k p_k\right) \leq \phi\left(q_k\right) + c_1 \alpha_k \nabla\phi_k^T p_k \tag{3.19}$$

   where $0 < c_1 < 1$. This condition is often referred to as the Armijo condition [30].

2. the second condition for α is defined:

$$\nabla\phi\left(q_k+\alpha_k p_k\right)^T p_k \geq c_2 \nabla\phi_k^T p_k \tag{3.20}$$

where $c_1 < c_2 < 1$

it is recommended that values of around 10^{-4} are used for c_1 [29], though this number can be determined experimentally. In summary, the first condition of Wolfe's criteria avoids large step sizes that result in high error, and the second condition avoids small step sizes that result in high computing cost. To correct for inappropriate values of α, a multiplying factor β can be introduced [31]. The method below is often referred to as Armijo backtracking and provides a simple and effective method for reducing step size to an appropriate value while guaranteeing a terminating loop.

- if condition 1 is not met, decrease the step size by the following

$$\alpha_k = \beta\alpha$$

Where β is a value between zero and one. The reduction in α can be accomplished easily using `while` loops until it meets condition 3.19. This method neglects Wolfe's second condition, but is sufficiently accurate to provide optimization solutions with low computing cost.

3.1.4 Example: LM Method Applied to the Elbow

The following is a derivation of the LM equations which can be placed in an iterative loop. The LM method was selected because it is more detailed than the NR method, but for two degrees of freedom it produces equations that are brief enough to be presented as a useful example problem. Since the robot has zero offsets ($a_i = 0$ for all joints $i = 1...7$), the Cartesian position of the elbow (joint 4 axis center) can be set using only inputs to joints 1 and 2, as demonstrated by the graphic in Figure 3.1.

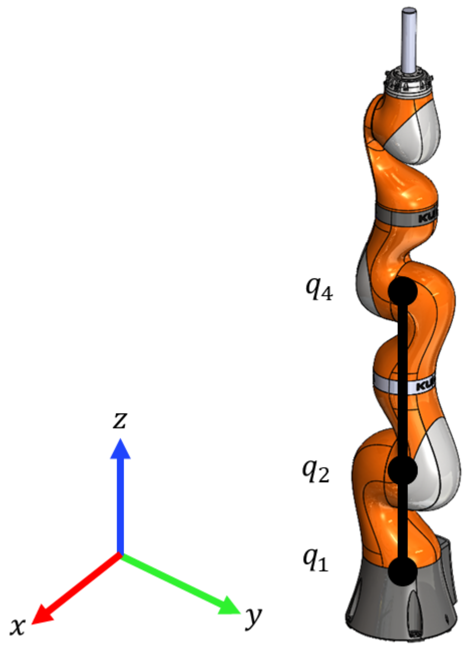


Figure 3.1: Graphic demonstrating the axes and zero-offset design of the LBR iiwa.

The graphic in Figure 3.2 displays a simplified model of the robot's segments spanning coordinates q_{1-4}. This demonstrates the Cartesian position of the robot elbow in terms of the angles q_1 and q_2, where x_d, y_d and z_d denote the desired x, y, and z coordinates of the robot elbow (joint q_4), respectively. The following substitutions are made to make calculations and diagrams easier to read:

$$l_1 = d_1 + d_2$$
$$l_2 = d_3 + d_4$$

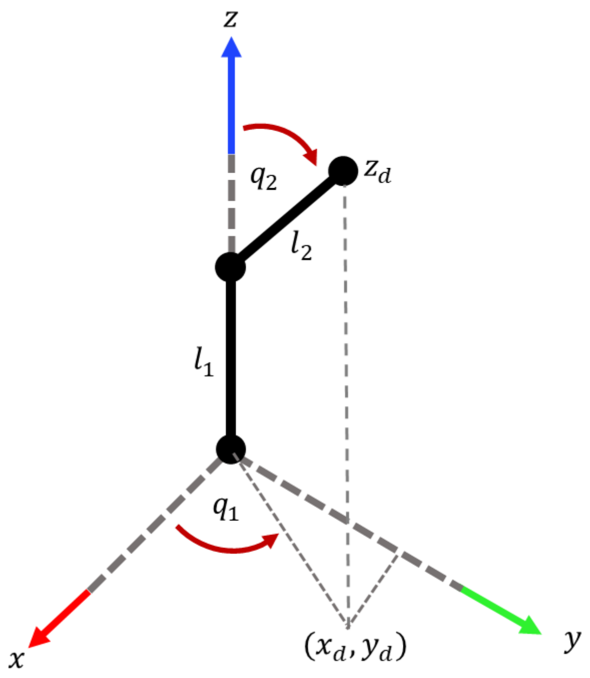


Figure 3.2: Coordinate frames of the robot from the base to the elbow center.

From Figure 3.2, we can derive the following position vector that relates the input angles q_1 and q_2 to the output Cartesian coordinates, x_d, y_d and z_d.

$$P(q_1,q_2)=\begin{bmatrix} P_x(q_1,q_2) \\ P_y(q_1,q_2) \\ P_z(q_2) \end{bmatrix} = \begin{bmatrix} l_2\sin(q_2)\cos(q_1) \\ l_2\sin(q_2)\sin(q_1) \\ l_1+l_2\cos(q_2) \end{bmatrix} \tag{3.21}$$

Where the residuals are defined

$$e(q_1,q_2)=\begin{bmatrix} x_d-P_x(q_1,q_2) \\ y_d-P_y(q_1,q_2) \\ z_d-P_z(q_2) \end{bmatrix} = \begin{bmatrix} x_d-l_2\sin(q_2)\cos(q_1) \\ y_d-l_2\sin(q_2)\sin(q_1) \\ z_d-l_1+l_2\cos(q_2) \end{bmatrix} \tag{3.22}$$

The aim of the LM method is to minimize the objective function

$$\phi(q_1,q_2)=\frac{1}{2}e^T\mathbf{W}e \tag{3.23}$$

where **W** is a diagonal matrix of weights corresponding to the desired x, y, and z coordinates.

$$\mathbf{W}=\begin{bmatrix} W_x & 0 & 0 \\ 0 & W_y & 0 \\ 0 & 0 & W_z \end{bmatrix} \tag{3.24}$$

the expansion of 3.23 is given by the following

$$\phi(q_1,q_2)=\frac{1}{2}\Big[W_x\left(x_d-l_2\sin(q_2)\cos(q_1)\right)^2+W_y\left(y_d-l_2\sin(q_2)\sin(q_1)\right)^2 \\ +W_z\left(l_1-z_d+l_2\cos(q_2)\right)^2\Big] \quad (3.25)$$

The change in q_i is calculated by 3.7, and the left hand side is inverted to isolate δ, which is a 2 by 1 vector containing update increments for q_1 and q_2. Substituting equation 3.22 in for e results in the following

$$\begin{bmatrix}\delta_{1,n}\\ \delta_{2,n}\end{bmatrix}=\left(\mathbf{J}^T\mathbf{J}+\lambda_n^2\mathbf{I}\right)^{-1}\mathbf{J}^T\begin{bmatrix}x_d-P_x(q_1,q_2)\\ y_d-P_y(q_1,q_2)\\ z_d-P_z(q_2)\end{bmatrix} \quad (3.26)$$

where λ is the damping factor, n is the iteration number, and $\mathbf{J}$ is the Jacobian matrix given by

$$\mathbf{J}(q_1,q_2)=\begin{bmatrix}\frac{\partial}{\partial q_1}P_x(q_1,q_2) & \frac{\partial}{\partial q_2}P_x(q_1,q_2)\\ \frac{\partial}{\partial q_1}P_y(q_1,q_2) & \frac{\partial}{\partial q_2}P_y(q_1,q_2)\\ \frac{\partial}{\partial q_1}P_z(q_2) & \frac{\partial}{\partial q_2}P_z(q_2)\end{bmatrix} \quad (3.27)$$

Substituting in equation 3.21 and calculating partial derivatives yields the following

$$\mathbf{J}(q_1,q_2)=\begin{bmatrix}-l_2\sin(q_1)\sin(q_2) & l_2\cos(q_1)\cos(q_2)\\ l_2\cos(q_1)\sin(q_2) & l_2\sin(q_1)\cos(q_2)\\ 0 & -l_2\sin(q_2)\end{bmatrix} \tag{3.28}$$

where λ is updated using 3.9

This brief example demonstrates an application of the LM optimization algorithm outlined in Section 3.1.2 as applied to the first two joints of the LBR iiwa. The angular position of these two joints dictate the Cartesian position of the fourth joint, often referred to as the elbow. This method can be expanded to a system with any number of degrees of freedom and is useful for determining required servo positions for a 6+ axis robot. It can also be used to calculate the angles in the glenohumeral joint that are required to achieve a given elbow position of a human in Cartesian space. The example is therefore a useful tool for transforming joint stiffness measurements from Cartesian space to joint space which is needed for developing models of the human upper extremity.

CHAPTER

4 SYSTEM IDENTIFICATION

4.1 Joint Stiffness

The following section outlines a method for determining nonparametric stiffness given a three-dimensional set of displacement and force data recordings over a time range. Nonparametric stiffness calculation is the first step in system identification, as it generates matrices of combined inertial, viscous, and elastic stiffness in the frequency domain. Inputs denoted x are the Carteisan coordinates of the LBR iiwa end effector for a given trial, and the outputs denoted y are the forces produced by the human subject as measured by the robot. Data is processed by calculating the cross-spectral density of each input and output using index notation, and inverting the resulting matrices for each frequency to determine stiffness in the frequency domain over a range of frequencies.

The nonparametric stiffness in Cartesian space is calculated in the frequency domain as follows:

$$\begin{bmatrix} F_x(f) \\ F_y(f) \\ F_z(f) \end{bmatrix} = \begin{bmatrix} H_{xx}(f) & H_{xy}(f) & H_{xz}(f) \\ H_{yx}(f) & H_{yy}(f) & H_{yz}(f) \\ H_{zx}(f) & H_{zy}(f) & H_{zz}(f) \end{bmatrix} \begin{bmatrix} X(f) \\ Y(f) \\ Z(f) \end{bmatrix}$$

where f is frequency, $F_i(f)$ is the Fourier transform of the endpoint force in the i direction, $X(f)$, $Y(f)$, and $Z(f)$ are the Fourier transforms in the Cartesian endpoint displacements, and $H_{ij}(f)$ are the transfer functions that relate the displacements in direction j to the forces in direction i [32]. $H_{ij}(f)$ can be further subdivided into a set of parametric transfer functions. In the frequency domain we have:

$$\mathbf{H}_{ij}(s) = \mathbf{I}_{ij} s^2 + \mathbf{B}_{ij} s + \mathbf{K}_{ij} \qquad (4.1)$$

where

$$s = 2\pi f \sqrt{-1}$$

And the matrix **I** is the endpoint inertia, **B** is the endpoint viscosity, and **K** is the elasticity defined in equation 4.2

$$\mathbf{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \quad , \quad \mathbf{B} = \begin{bmatrix} B_{xx} & B_{xy} & B_{xz} \\ B_{yx} & B_{yy} & B_{yz} \\ B_{zx} & B_{zy} & B_{zz} \end{bmatrix} \quad , \quad \mathbf{K} = \begin{bmatrix} K_{xx} & K_{xy} & K_{xz} \\ K_{yx} & K_{yy} & K_{yz} \\ K_{zx} & K_{zy} & K_{zz} \end{bmatrix} \tag{4.2}$$

For a system of 3 inputs and 3 outputs, the nonparametric stiffness is defined by

$$\mathbf{H}_{y_j x_i} = \begin{bmatrix} H_{y_1 x_1}(f) & H_{y_2 x_1}(f) & H_{y_3 x_1}(f) \\ H_{y_1 x_2}(f) & H_{y_2 x_2}(f) & H_{y_3 x_2}(f) \\ H_{y_1 x_3}(f) & H_{y_2 x_3}(f) & H_{y_3 x_3}(f) \end{bmatrix} \tag{4.3}$$

Cross-spectral density matrices are determined [33, 34] for the 3-input 3-output system as follows

$$\mathbf{G}_{x_i y_j} = \begin{bmatrix} G_{x_1 y_1}(f) & G_{x_1 y_2}(f) & G_{x_1 y_3}(f) \\ G_{x_2 y_1}(f) & G_{x_2 y_2}(f) & G_{x_2 y_3}(f) \\ G_{x_3 y_1}(f) & G_{x_3 y_2}(f) & G_{x_3 y_3}(f) \end{bmatrix} \tag{4.4}$$

$$\mathbf{G}_{x_i x_j} = \begin{bmatrix} G_{x_1 x_1}(f) & G_{x_1 x_2}(f) & G_{x_1 x_3}(f) \\ G_{x_2 x_1}(f) & G_{x_2 x_2}(f) & G_{x_2 x_3}(f) \\ G_{x_3 x_1}(f) & G_{x_3 x_2}(f) & G_{x_3 x_3}(f) \end{bmatrix} \tag{4.5}$$

Where $\mathbf{G}_{x_i x_j}$ is the cross spectral density between inputs x_i and x_j using index notation, and $\mathbf{G}_{x_i y_j}$ is the cross-spectral density between inputs x_i and outputs y_j [35]. These matrices are calculated using the MATLAB function `cpsd`. Once both of these matrices have been calculated, the nonparametric stiffness matrix **H** for a given frequency f can be solved by inversion.

$$\mathbf{H}_{y_j x_i} = \mathbf{G}^{-1}_{x_i x_j} \mathbf{G}_{x_i y_j}$$

The resulting **H** matrix is referred to as nonparametric stiffness because it does not differentiate between inertial, viscous and elastic components. Techniques used to differentiate the results of these plots into specific components are not discussed in this thesis, however, the reader can find more resources on this topic [36, 37, 38, 39].

CHAPTER

5 RESULTS

5.1 Inverse Kinematics

The following section presents sample results from an IK trial comparing the three algorithms discussed in Chapter 3. Randomized initial angle estimates, convergence rates, final position and configuration error, and computing cost are all discussed.

5.1.1 Comparison of IK Algorithms

The following results were obtained by running 100 trials with randomly generated configurations within the joint limits listed in Table 5.1.

Table 5.1: Joint limits for the LBR iiwa.

Joint	**Limit, °**
q_1	±170
q_2	±120
q_3	±170
q_4	±120
q_5	±170
q_6	±120
q_7	±175

Estimate error was randomly generated between ±45° and added to the target input angles. It was found that LM and NR methods tend to experience convergence issues when initial error estimates exceed ±45°, so error was limited to this range. Table 5.2 displays the values of parameters chosen for the three algorithms. These variables were chosen because they reliably worked for each algorithm and produced similar error results for all three.

Table 5.2: Parameters chosen for optimization algorithms.

NR	ϵ_{max}	$1e-12$
LM	ϕ_{max}	$1e-20$
	ν	1.1
	λ_0	$5e-5$
BFGS	ϵ_{max}	$1e-12$
	α_0	1
	β	0.5
	c_1	$1e-4$

A summary of the results from the 100 trials is shown in Table 5.3

Table 5.3: Summary of algorithm performance from 100 randomly generated trials.

Metric		**NR**	**LM**	**BFGS**
Computation Time (ms)	Mean	5.4841	9.9810	40.100
	STDEV	4.1118	10.792	11.552
Position RSS	Mean	1.1028e-29	1.6948e-22	6.6343e-26
	STDEV	7.9588e-29	9.2661e-22	1.9815e-25
Orientation RSS	Mean	2.387e-28	7.6437e-22	2.4431e-26
	STDEV	1.589e-27	2.7983e-21	4.5991e-26
Number of iterations	Mean	11.09	11.27	32.13
	STDEV	8.8193	10.1552	8.1211

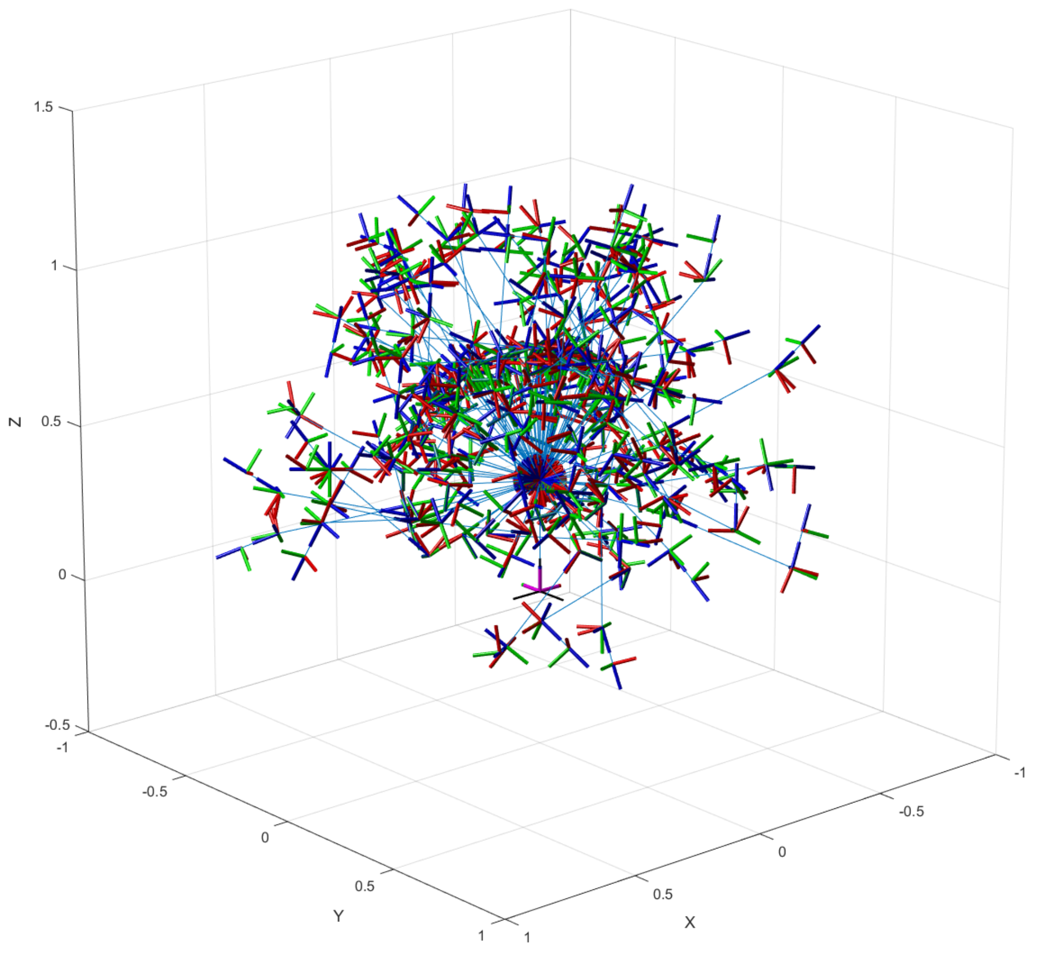


Figure 5.1: 100 LBR iiwa configurations randomly generated within the joint limits in Table 5.1.

Results from a Single Trial

The results presented below were obtained with an initial configuration and error presented in Table 5.4.

Table 5.4: Sample initial error applied to NR, LM, and BFGS algorithms.

Joint	Target Position, °	Initial Estimate Error, °
q_1	0	−32
q_2	−7	+16
q_3	0	+33
q_4	−70	+38
q_5	0	−44
q_6	120	+39
q_7	0	−44

The residual sum of squares for the orientation and position are shown Figure 5.2, and the residual sum of squares at the final iteration for each algorithm are listed in Table 5.5.

Table 5.5: Sample residual sum of squares at final iteration for NR, LM, and BFGS algorithms.

Method	Iteration	Position RSS	Orientation RSS
NR	6	$2.8504e{-34}$	$1.3084e{-31}$
LM	6	$1.9366e{-28}$	$3.0378e{-26}$
BFGS	31	$1.0312e{-26}$	$1.0590e{-26}$

Figure 5.3 shows the change in angles over the range of iterations for each method, and the final servo angles are listed for each algorithm in Table 5.6. Table 5.6 demonstrates that while BFGS takes the most iterations to meet specified error criteria, it is closest to

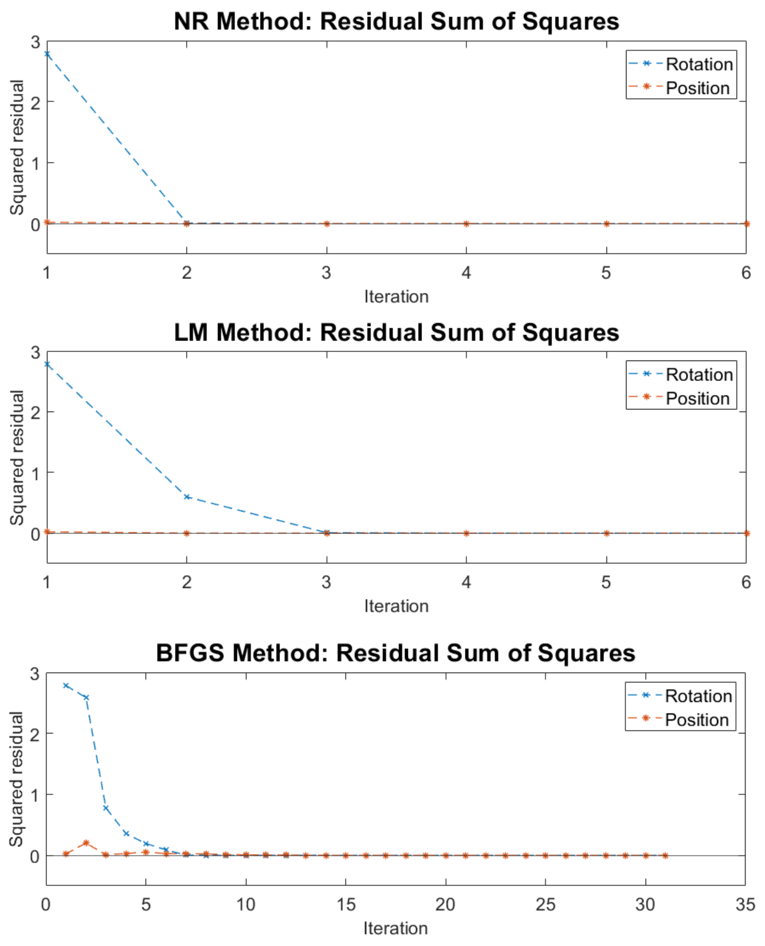


Figure 5.2: Sample residual sum of squares at each iteration for NR, LM, and BFGS algorithms.

the target position. Figure 5.3 indicates that while NR and LM yield nearly identical results, the path to convergence is slightly different due to differences in the iterative process.

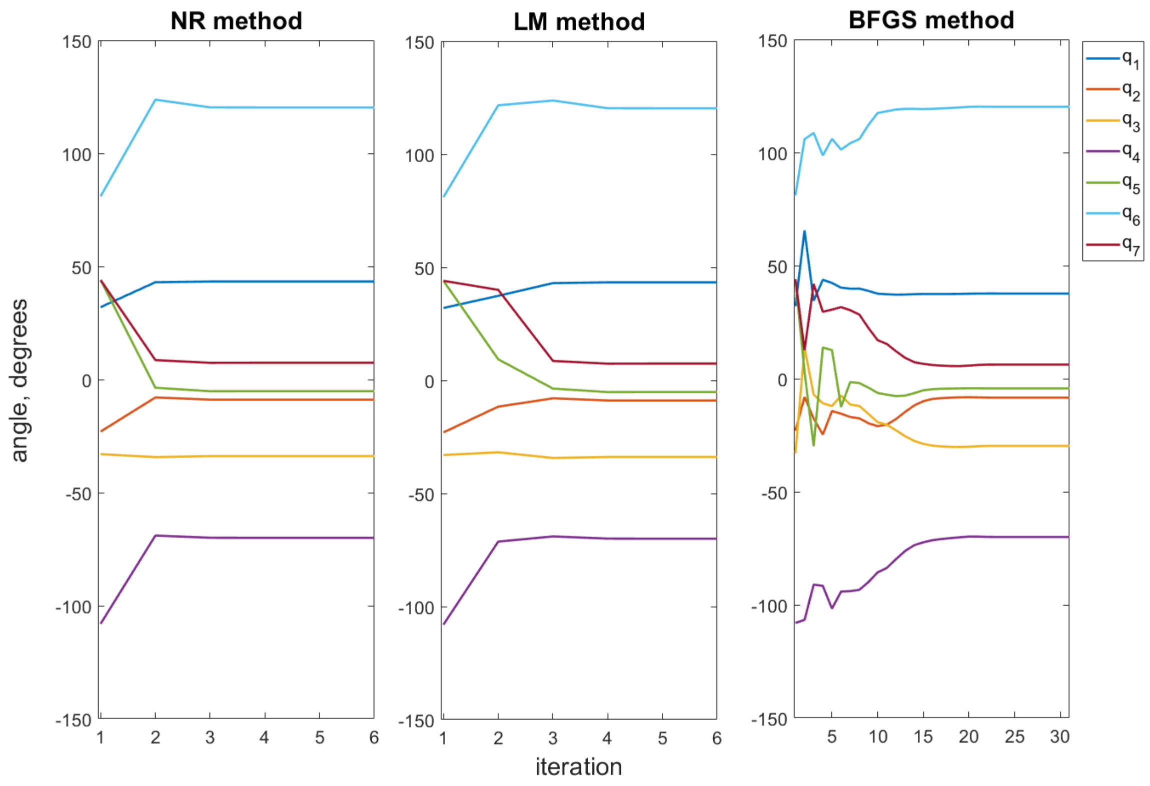


Figure 5.3: Sample IK results using the NR, LM, and BFGS algorithms.

Table 5.6: Sample final servo angles for NR, LM, and BFGS algorithms.

Target	**NR Result** (°)	**LM Result** (°)	**BFGS Result** (°)
0	43.323	43.323	37.632
−7	−8.9182	−8.9182	−8.3871
0	−33.858	−33.858	−29.723
−70	−70	−70	−70
0	−5.1419	−5.1419	−4.2998
120	120.29	120.29	120.20
0	7.4313	7.4314	6.2116

The computation times are listed in Table 5.7

Table 5.7: Sample computation times of NR, LM, and BFGS algorithms.

Method	**Computation Time** (milliseconds)
NR	3.3027
LM	4.9619
BFGS	47.768

Last, the final coordinate frames are overlaid using the MATLAB robotics toolbox to demonstrate that while each method results in different joint configurations, the orientation and position off the end effector is identical for all three.

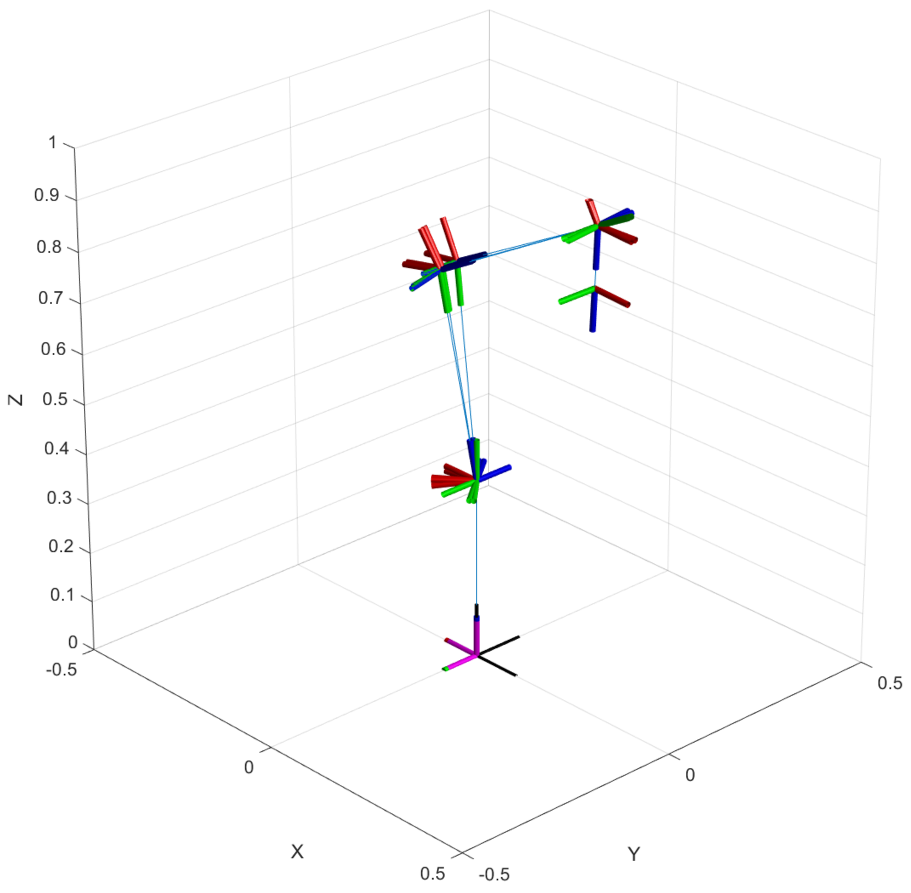


Figure 5.4: Sample coordinate frame comparison for NR, LM, and BFGS algorithms.

5.1.2 Considerations for Numerical Inverse Kinematics

The DH matrix that dictates the orientation and Cartesian position of the endpoint is as follows

$$\mathbf{T}_{EE} = \begin{bmatrix} n_{xEE} & s_{xEE} & a_{xEE} & P_{xEE} \\ n_{yEE} & s_{yEE} & a_{yEE} & P_{yEE} \\ n_{zEE} & s_{zEE} & a_{zEE} & P_{zEE} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

Where the indices n, s, a are orientation components and the vector of P values specify the Cartesian position of the end effector. These are all equations dependent upon variables q_{1-7}, parameters α, d, and a and are obtained by multiplying the DH matrices for arms 1-7 using the method outlined in section 2.

It is important to note that in the 7 DoF scenario, the Jacobian matrix is quite lengthy and takes several minutes to compute. If angles q_1 through q_7 are calculated simultaneously, the Jacobian matrix is size 12 by 7, meaning that 84 equations must be solved to fill the matrix. For this reason, it is recommended to parameterize the Jacobian function with d, a, α and o before inserting it into a function as doing so will greatly reduce the length of the equations, improving computing time. The script for calculating the Jacobian was found to have over one million characters if left with variable inputs for d, a, α and o, which can be reduced to around 24,000 characters if all parameters are added beforehand such that the only variables are q_{1-7} and d_7, which is the length of the end effector. It is recommended that d_7 is left a variable because the length of end effectors can vary based on the robot specifications and application.

It should also be noted that if ill-conditioned matrix warnings appear in MATLAB while running the LM algorithm, adjusting the values for ν and λ_0 will usually fix the issue. Since the position and orientation of the end effector requires solving for seven angle inputs based on twelve outputs, the IK algorithms are too long to be included in this paper, however, the reader can download the code to perform this operation at
https://github.com/abrynildsen/IK

5.2 Digital Signal Processing

The following plots display the path inputs and nonparametric stiffnesses found using the methods presented in Chapter 5. The sample perturbation shown in Figure 5.6 was generated using a cutoff frequency of 5 Hz, a sampling frequency of 50 Hz, an amplitude of 1.5 mm, and a duration of 45 seconds. Since collection frequencies from the LBR iiwa were too low to capture the full frequency range found in other literature, synthetic data was generated using the mean and standard deviation of an experimental trial [40]. Figure 5.5 displays the experimental data, and Figure 5.6 displays the synthetic data. The figures are zoomed to a five second period to demonstrate the difference.

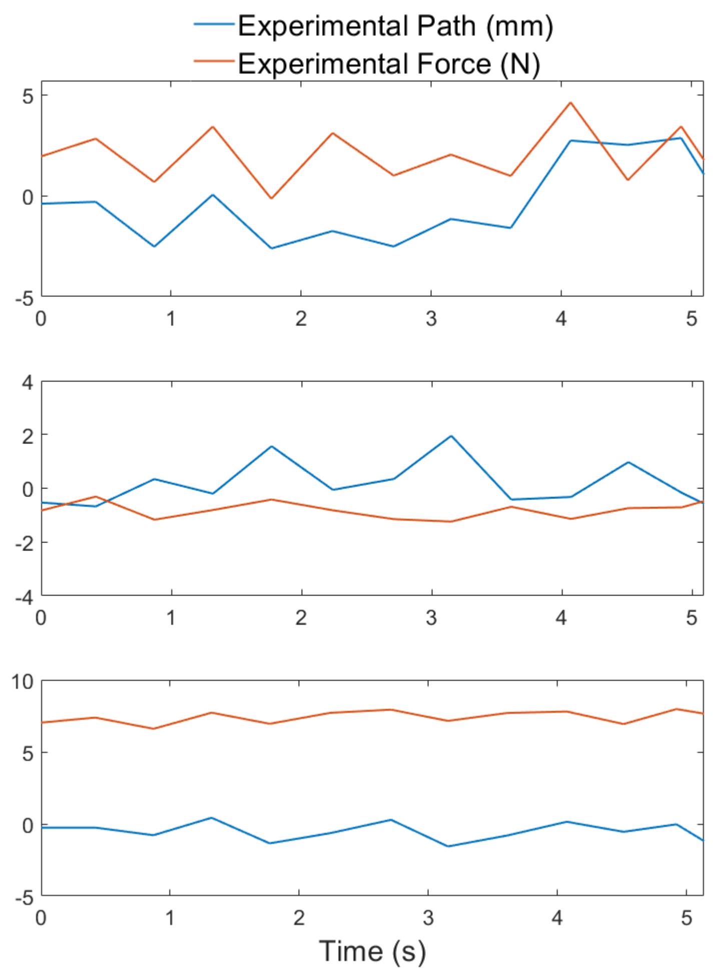


Figure 5.5: Experimental position and force data.

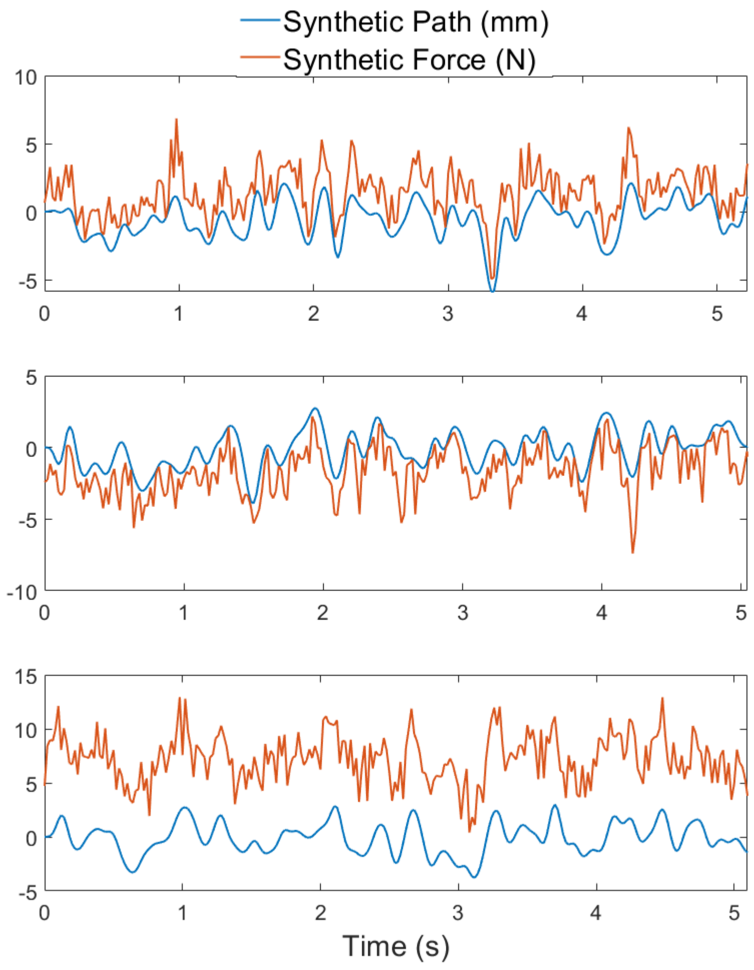


Figure 5.6: Synthetic position and force data.

Figure 5.7 displays a set of plots of the nonparametric stiffnesses obtained using methods outlined in Chapter 4. Experimental data from Figure 5.5 was used to generate the plots in Figure 5.7 and the synthetic force and position data shown in Figure 5.6 was used to generate the plots in Figure 5.8.

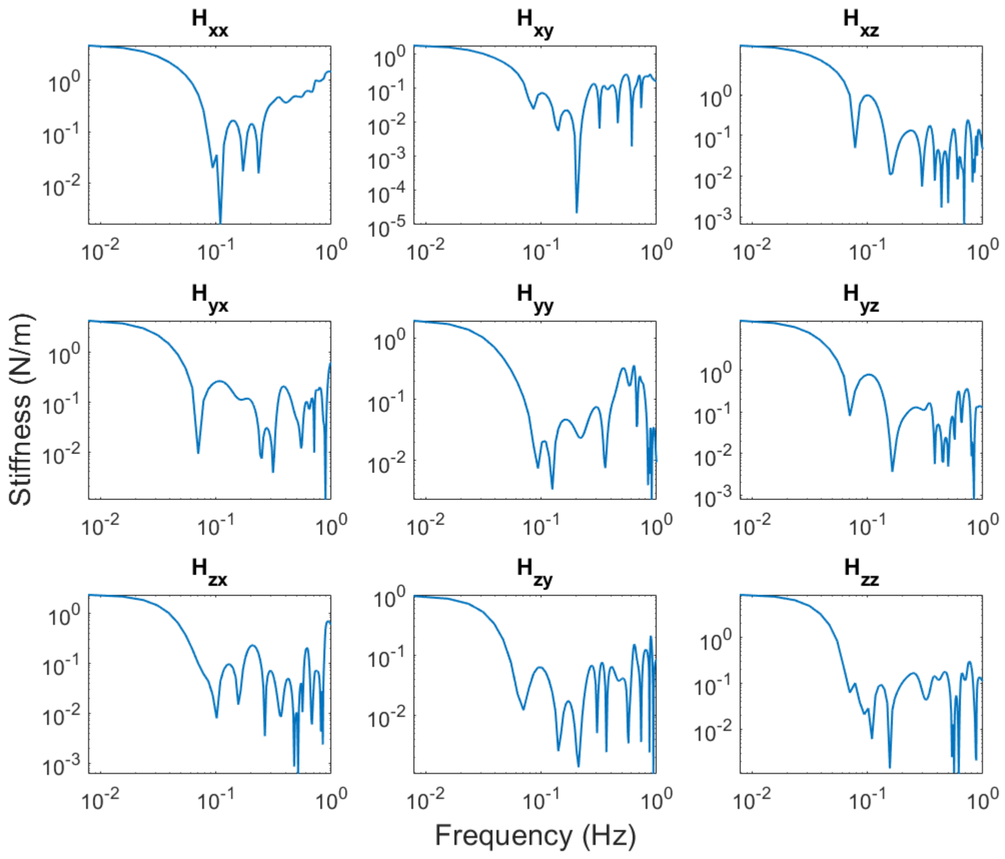


Figure 5.7: Nonparametric stifnesses obtained from experimental data in Figure 5.5.

The plots in Figure 5.7 display the stiffness with respect to frequency and are calculated using cross-spectral density matrices which relate inputs to outputs in the frequency domain. They indicate that at low frequencies, stiffness values are generally quite low. The plots are an indicator of the influence that the three inputs and three outputs have on each other,

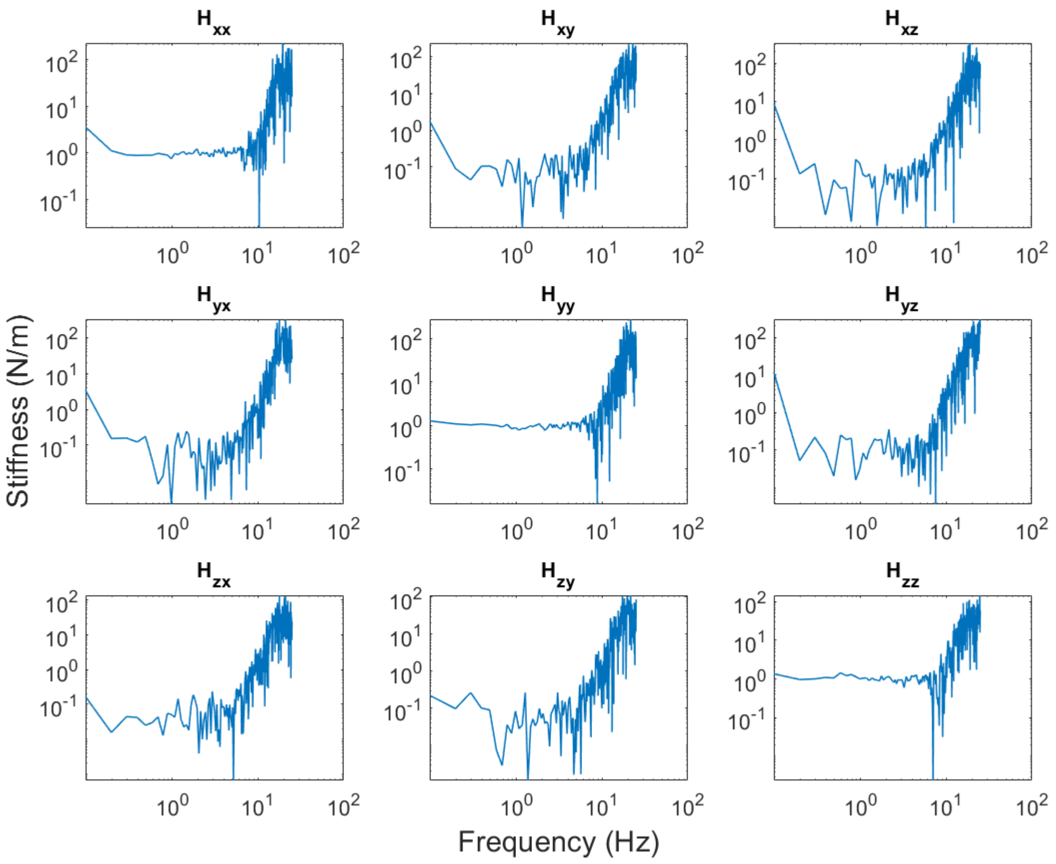


Figure 5.8: Nonparametric stifnesses obtained from synthetic data in Figure 5.6.

and can be used to calculate inertial, viscous, and elastic stiffness parameters using statistical methods. The data can then be transformed to joint space using inverse kinematics, and mathematical models of upper extremity parameters can be developed.

CHAPTER

6 DISCUSSION

6.0.1 Inverse Kinematics

Through experimentation, it was determined that the BFGS algorithm most reliably produces IK results within joint limits even with input estimate error greater than ±45 degrees, but comes at the cost of slower computing times. In the context of motion planning and research this is likely not an issue, but for real-time simulations or control systems it may become a hindrance, and either NR or LM algorithms may be preferable. Each of the methods produced very low error, which can be adjusted to meet required tolerances by changing the acceptable values of the objective function that terminate the iterative loop.

6.0.2 Digital Signal Processing

Synthetic data generated using random number generation with the mean and standard deviation of the experimental data produced nonparametric stiffness results that resemble those found in other literature [35, 32, 13]. Stiffness increases in a roughly linear manner with frequency after a certain frequency level is reached, likely due to the constant inertial term which is multiplied by squared frequency as demonstrated by equation 4.1. Experimental data collected using an LBR iiwa produced the results shown in Figure 5.7, however, the low sampling rate of the robot (around 2Hz) meant that the full frequency range found in other literature was not captured. While most human movement does occur at low frequencies a larger frequency range of displacement and force data should be collected to generate dynamic models that can account for inputs with position, velocity, and acceleration.

6.1 Future Work

6.1.1 Inverse Kinematics

While the algorithms provided in this paper produce reliable results with reasonable computing costs, the following features should be implemented at some point in the future to maximize their versatility and robustness.

Limit Avoidance

Issues occur with the IK algorithms presented in this paper for two primary reasons:

- high initial servo input angle estimate error
- large displacement requirement for desired DH configuration

The most common error is that the servo input angles calculated by the algorithm violate the limits of the robot, meaning that the configuration is not actually obtainable. To avoid this scenario, a gradient projection [41], weighted least-norms, or clamping weighted least-norms [42] method can be implemented to the IK algorithm of choice.

BFGS Line Search

While the Armijo method is suitable for producing an appropriately small step size, it can likely be improved by being expanded to meet both of Wolfe's criteria. This could be accomplished by introducing a line search algorithm [43] that increases the step size to meet Wolfe's second criteria in addition to decreasing it to meet the first criteria, resulting in lower computing cost.

Optimization for Stiffness Parameterization

The next step toward developing dynamic models of the upper extremity is to separate the nonparametric stiffness matrices into inertial, viscous, and elastic components. This can be accomplished using a method such as the Nelder-Mead multi-dimensional optimization algorithm [36].

6.1.2 Digital Signal Processing

While the method presented in Chapter 4 yielded reasonable results for synthetically generated stiffness data, the recording frequency of the LBR iiwa using the Kuka Sunrise Toolbox was too low to produce stiffness values that capture the full frequency range of human motion, which in other literature is at or above 10 Hz [44, 13, 1]. The collection frequency for randomized stiffness trials was around 2 Hz, which is insufficient to generate plots that show a defined trend for power and nonparametric stffness beyond 1Hz. Options are available for increasing the collection frequency, including the Kuka Fast Robot Interface (FRI) and programming the robot directly using Java to reduce communications latency caused by the MATLAB client. Through experimentation with synthetic data, it was determined that collection frequency needs to be around 50 Hz to produce results that better capture the full frequency range.

REFERENCES

[1] K. Jalaleddini, M. A. Golkar, and R. E. Kearney, "Measurement of Dynamic Joint Stiffness from Multiple Short Data Segments," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 25, no. 7, pp. 925–934, 2017.

[2] R. E. Kearney and I. W. Hunter, "System identification of human joint dynamics," *Critical Reviews in Biomedical Engineering*, vol. 18, no. 1, pp. 55–87, 1990.

[3] S. Kücük and Z. Bingül, "The inverse kinematics solutions of industrial robot manipulators," *Proceedings of the IEEE International Conference on Mechatronics 2004, ICM'04*, pp. 274–279, 2004.

[4] G. K. Singh and J. Claassens, "An analytical solution for the inverse kinematics of a redundant 7DoF manipulator with link offsets," *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pp. 2976–2982, 2010.

[5] D. A. Drexler and I. Harmati, "Joint constrained differential inverse kinematics algorithm for serial manipulators," *Periodica Polytechnica Electrical Engineering and Computer Science*, vol. 56, no. 4, pp. 95–104, 2012.

[6] H. H. An, W. I. Clement, and B. Reed, "Analytical inverse kinematic solution with self-motion constraint for the 7-DOF restore robot arm," *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM*, pp. 1325–1330, 2014.

[7] W. Liu, D. Chen, and J. Steil, "Analytical Inverse Kinematics Solver for Anthropomorphic 7-DOF Redundant Manipulators with Human-Like Configuration Constraints," pp. 63–79, 2017.

[8] P. K. Artemiadis, P. T. Katsiaris, and K. J. Kyriakopoulos, "A biomimetic approach to inverse kinematics for a redundant robot arm," *Autonomous Robots*, vol. 29, no. 3-4, pp. 293–308, 2010.

[9] Y. Wang, "Closed-Form Inverse Kinematic Solution for Anthropomorphic Motion in Redundant Robot Arms," Ph.D. dissertation, Arizona State University, 2013.

[10] M. Gong, X. Li, and L. Zhang, "Analytical Inverse Kinematics and Self-Motion Application for 7-DOF Redundant Manipulator," *IEEE Access*, vol. 7, pp. 18 662–18 674, 2019.

[11] D. T. Westwick and R. E. Kearney, *Identification of Nonlinear Physiological Systems*. Piscataway, New Jersey: IEEE Press, 2003.

[12] C. Bingham, M. D. Godfrey, and J. W. Tukey, "Modern Techniques of Power Spectrum Estimation," *IEEE Transactions on Audio and Electroacoustics*, vol. 15, no. 2, pp. 56–66, 1967.

[13] R. D. Trumbower, M. A. Krutky, B. S. Yang, and E. J. Perreault, “Use of self-selected postures to regulate multi-joint stiffness during unconstrained tasks,” *PLoS ONE*, vol. 4, no. 5, 2009.

[14] J. Denavit and R. S. Hartenberg, “A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices,” *Journal of Applied Mechanics*, pp. 215–221, 1955.

[15] Z. Li, M. Brandstötter, and M. Hofbaur, “Analysis of Kinematic Singularities for a Serial Redundant Manipulator with 7 DOF,” *Springer Proceedings in Advanced Robotics*, vol. 8, pp. 179–186, 2019.

[16] X. Tian, Q. Xu, and Q. Zhan, “An analytical inverse kinematics solution with joint limits avoidance of 7-DOF anthropomorphic manipulators without offset,” *Journal of the Franklin Institute*, vol. 358, no. 2, pp. 1252–1272, 2021. [Online]. Available: https://doi.org/10.1016/j.jfranklin.2020.11.020

[17] S. S. Hjorth and M. Thesis, “Investigation and Implementation of Workspace restrictions for the KUKA LBR iiwa,” 2019.

[18] “Inverse Kinematics Algorithms.” [Online]. Available: https://www.mathworks.com/help/robotics/ug/inverse-kinematics-algorithms.html

[19] S. R. Buss, “Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods,” *Department of Mathematics, University of California, San Diego*, 2009.

[20] T. Sugihara, “Solvability-Unconcerned Inverse Kinematics by the LevenbergMarquardt Method,” vol. 27, no. 5, pp. 984–991, 2011.

[21] I. B. Love, “Levenberg-Marquardt Algorithm in Robotic Controls,” Ph.D. dissertation, University of Washington, 2020.

[22] D. Tolani, A. Goswami, and N. I. Badler, “Real-time inverse kinematics techniques for anthropomorphic limbs,” *Graphical Models*, vol. 62, no. 5, pp. 353–388, 2000.

[23] J. J. Uigker, J. Denavit, and R. S. Hartenberg, “An iterative method for the displacement analysis of spatial mechanisms,” *Journal of Applied Mechanics, Transactions ASME*, vol. 31, no. 2, pp. 309–314, 1964.

[24] D. L. Pieper, “THE KINEMATICS OF MANIPULATORS UNDER COMPUTER CONTROL,” Ph.D. dissertation, Stanford University, 1969.

[25] A. A. Goldenberg, B. Benhabib, and R. G. Fenton, “A Complete Generalized Solution to the Inverse Kinematics of Robots,” *IEEE Journal on Robotics and Automation*, vol. 1, no. 1, pp. 14–20, 1985.

[26] D. W. Marquardt, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," *Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[27] S. R. Buss and J.-S. Kim, "Selectively Damped Least Squares for Inverse Kinematics," *Journal of Graphics Tools*, vol. 10, no. 3, pp. 37–49, 2005.

[28] D. Goldfarb, "Extension of Davidon ' s Variable Metric Method to Maximization Under Linear Inequality and Equality Constraints," *Society for Industrial and Applied Mathematics*, vol. 17, no. 4, pp. 739–764, 1969.

[29] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed., T. V. Mikosch, Ed. New York: Springer, 2006.

[30] L. Armijo, "MINIMIZATION OF FUNCTIONS HAVING LIPSCHITZ CONTINUOUS FIRST PARTIAL DERIVATIVES," *Pacific Journal of Mathematics*, vol. 16, no. 1, 1966.

[31] N. Andrei, "An acceleration of gradient descent algorithm with backtracking for unconstrained optimization," *Numerical Algorithms*, vol. 42, no. 1, pp. 63–73, 2006.

[32] E. J. Perreault, R. F. Kirsch, and P. E. Crago, "Effects of voluntary force generation on the elastic components of endpoint stiffness," *Experimental Brain Research*, vol. 141, no. 3, pp. 312–323, 2001.

[33] D. O. Smallwood, "Using singular value decomposition to compute the conditioned cross-spectral density matrix and coherence functions," Albuquerque, NM, 1995. [Online]. Available: https://www.osti.gov/servlets/purl/106476-vgSP7p/webviewable/

[34] J. S. Bendat and A. G. Piersol, *Random Data Analysis and Measurement Procedures*, 4th ed. Hoboken, New Jersey: John Wiley and Sons, Inc., 2010.

[35] E. J. Perreault, R. F. Kirsch, and A. M. Acosta, "Multiple-input, multiple-output system identification for characterization of limb stiffness dynamics," *Biological Cybernetics*, vol. 80, no. 5, pp. 327–337, 1999.

[36] E. J. Perreault, R. F. Kirsch, and P. E. Crago, "Multijoint dynamics and postural stability of the human arm," *Experimental Brain Research*, vol. 157, no. 4, pp. 507–517, 2004.

[37] E. J. Perreault, P. E. Crago, and R. F. Kirsch, "Estimation of intrinsic and reflex contributions to muscle dynamics: A modeling study," *IEEE Transactions on Biomedical Engineering*, vol. 47, no. 11, pp. 1413–1421, 2000.

[38] S. Stroeve, "Impedance characteristics of a neuromusculoskeletal model of the human arm I. Posture control," *Biological Cybernetics*, vol. 81, no. 5-6, pp. 475–494, 1999.

[39] J. M. Dolan, M. B. Friedman, and M. L. Nagurka, "Dynamic and Loaded Impedance Components in the Maintenance of Human Arm Posture," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, no. 3, pp. 698–709, 1993.

[40] S. Kaplan, “The MoBL KUKA Programmer’s Instruction Manual,” North Carolina State University Movement and Biomechanics Laboratory, Raleigh, North Carolina, Tech. Rep., 2019.

[41] X. Zhang, B. Fan, C. Wang, and X. Cheng, “An improved weighted gradient projection method for inverse kinematics of redundant surgical manipulators,” *Sensors*, vol. 21, no. 21, 2021.

[42] S. Huang, Y. Peng, W. Wei, and J. Xiang, “Clamping weighted least-norm method for the manipulator kinematic control with constraints,” *International Journal of Control*, vol. 89, no. 11, pp. 2240–2249, 2016.

[43] M. Engell-Nørregård and K. Erleben, “A projected back-tracking line-search for constrained interactive inverse kinematics,” *Computers and Graphics (Pergamon)*, vol. 35, no. 2, pp. 288–298, 2011.

[44] E. De Vlugt, A. C. Schouten, and F. C. Van Der Helm, “Closed-loop multivariable system identification for the characterization of the dynamic arm compliance using continuous force disturbances: A model study,” *Journal of Neuroscience Methods*, vol. 122, no. 2, pp. 123–140, 2003.

APPENDICES

APPENDIX

A MATLAB CODE

The following is the two degree of freedom LM algorithm detailed in section 3.1.4

```
%% Levenberg-Marquardt algorithm for 2 DoF
clc;clear
% parameters
d1 = 0.36; % length of robot arm 1
d2 = 0;    % length of robot arm 2
d3 = 0.42; % length of robot arm 3
d4 = 0;    % length of robot arm 4

% put in arbitrary angle targets, radians
q1_target = 55*pi/180;
q2_target = 35*pi/180;

% give the angle an initial guess, add some error
```

```
q1_cur = -q1_target/.3;
q2_cur = q2_target/3;

% the following coordinates were obtained with q1 = 55, q2 =
    35
xd = FKxcalc(d3,d4,q1_target,q2_target);  % desired x
   position for robot elbow
yd = FKycalc(d3,d4,q1_target,q2_target);  % desired y
   position for robot elbow
zd = FKzcalc(d1,d2,d3,d4,q2_target); % desired z position
   for robot elbow

Wx = 1; % weight for x direction
Wy = 1; % weight for y direction
Wz = 1; % weight for z direction


phi_cutoff = 1e-6;    % define acceptable error
v = 10;               % scaling factor, determined
   experimentally
lambda_r = 1e-3;      % initial estimate for lambda

% compute a new phi value given a q value
phi_prev = phicalc(xd,yd,zd,Wx,Wy,Wz,d1,d2,d3,d4,q1_cur,
   q2_cur);

while phi_prev > phi_cutoff
    lambda_cur = lambdacalc(d1,d2,d3,d4,phi_prev,lambda_r,v,
       xd,yd,zd,Wx,Wy,Wz,q1_cur,q2_cur);
    delta_cur = deltacalc(d1,d2,d3,d4,xd,yd,zd,lambda_cur,
       q1_cur,q2_cur);
    q1_cur = q1_cur + delta_cur(1);
    q2_cur = q2_cur + delta_cur(2);
    phi_prev = phicalc(xd,yd,zd,Wx,Wy,Wz,d1,d2,d3,d4,q1_cur,
       q2_cur)
end

% calculate final xyz coordinates
xf = FKxcalc(d3,d4,q1_cur,q2_cur)
yf = FKycalc(d3,d4,q1_cur,q2_cur)
zf = FKzcalc(d1,d2,d3,d4,q2_cur)
```

```
% calculate error
xe = xd - xf
ye = yd - yf
ze = zd - zf
```

The following are functions used in the 2DoF algorithm listed above

```
%% Functions
% forward kinematics for x coordinate
function FKx = FKxcalc(d3,d4,q1,q2)
FKx = (d3 + d4)*sin(q2)*cos(q1);
end

% forward kinematics for y coordinate
function FKy = FKycalc(d3,d4,q1,q2)
FKy = (d3 + d4)*sin(q2)*sin(q1);
end

% forward kinematics for z coordinate
function FKz = FKzcalc(d1,d2,d3,d4,q2)
FKz = (d1 + d2) + (d3 + d4)*cos(q2);
end

% calculate delta
function delta = deltacalc(d1,d2,d3,d4,xd,yd,zd,lambda,q1,q2
   )
% calculate the Jacobian
J = [-sin(q1)*sin(q2)*(d3 + d4), cos(q1)*cos(q2)*(d3 + d4);
     cos(q1)*sin(q2)*(d3 + d4) , cos(q2)*sin(q1)*(d3 + d4);
     0                         , -sin(q2)*(d3 + d4)       ];

% calculate residuals
x_res = xd - FKxcalc(d3,d4,q1,q2);     % calculate x residual
y_res = yd - FKycalc(d3,d4,q1,q2);     % calculate y residual
z_res = zd - FKzcalc(d1,d2,d3,d4,q2); % calculate z residual
% assemble residuals into vector
e = [x_res;y_res;z_res];

% solve for delta
lhs = transpose(J)*J + lambda^2*eye(2);
rhs = transpose(J)*e;
delta = lhs\rhs;
end

% calculate phi
function phi = phicalc(xd,yd,zd,Wx,Wy,Wz,d1,d2,d3,d4,q1,q2)
```

```
x_res = xd - FKxcalc(d3,d4,q1,q2);     % calculate x residual
y_res = yd - FKycalc(d3,d4,q1,q2);     % calculate y residual
z_res = zd - FKzcalc(d1,d2,d3,d4,q2); % calculate z residual

% calculate phi
phi = 0.5*(x_res^2*Wx + y_res^2*Wy + z_res^2*Wz);
end

% calculate lambda
function lambda = lambdacalc(d1,d2,d3,d4,phi_prev,lambda_r,v
   ,xd,yd,zd,Wx,Wy,Wz,q1_r,q2_r)
% compute new lambda given prior phi value and current q
   value
% try case 1 lambda = lambda_{r-1} / v
lambda_temp = lambda_r/v;
delta_temp = deltacalc(d1,d2,d3,d4,xd,yd,zd,lambda_temp,q1_r
   ,q2_r);
% update q1 and q2
q1_temp = q1_r + delta_temp(1);
q2_temp = q2_r + delta_temp(2);
phi_temp = phicalc(xd,yd,zd,Wx,Wy,Wz,d1,d2,d3,d4,q1_temp,
   q2_temp);

if phi_temp < phi_prev
    lambda = lambda_temp;
    disp('case 1 used')
else
    % if case 1 fails, try case 2
    lambda_temp = lambda_r;
    delta_temp = deltacalc(d1,d2,d3,d4,xd,yd,zd,lambda_temp,
       q1_r,q2_r);
    q1_temp = q1_r + delta_temp(1);
    q2_temp = q2_r + delta_temp(2);
    phi_temp = phicalc(xd,yd,zd,Wx,Wy,Wz,d1,d2,d3,d4,q1_temp
       ,q2_temp);
    if phi_temp < phi_prev
        lambda = lambda_temp
        disp('case 2 used')
    else
        % if case 2 fails, try case 3
        while phi_temp <= phi_prev
```

```
            % keep multiplying lambda_prev by v until it
               satisfies error criteria
            % this is the equivalent of solving for w
            lambda_temp = lambda_prev*v;
            delta_temp = deltacalc(d1,d2,d3,d4,xd,yd,zd,
               lambda_temp,q1_r,q2_r);
            q1_temp = q1_r + delta_temp(1);
            q2_temp = q2_r + delta_temp(2);
            phi_temp = phicalc(xd,yd,zd,Wx,Wy,Wz,d1,d2,d3,d4
               ,q1_temp,q2_temp);
            lambda_prev = lambda_temp;
            disp('case 3 used')
        end
        lambda = lambda_temp
    end
end
end
```

APPENDIX

B

SIMSCAPE MULTIBODY

Simscape Multibody can be used to create multiphysics simulations and visual demonstrations of inverse and forward kinematics. To use the software, Simscape Multibody Link needs to be installed in the CAE software used for modeling. This will enable the user to export a CAD assembly as a `.xml` file containing all models and physical properties that can be imported to MATLAB. Instructions for installing Simscape Multibody Link are available at https://www.mathworks.com/help/physmod/smlink/index.html?s_tid=CRUX_lftnav. After setting up Simscape Multibody Link, the model can be imported with the following command:

```
% run this line of code to import the robot CAD to simscape
iiwa_LBR_14_R820 = smimport('iiwa LBR 14 R820.xml')
```

Where the name `iiwa LBR 14 R820.xml` corresponds to the `.xml` file exported using Simscape Multibody Link. After running this command, Simulink will automatically add connections between the blocks, as well as apply gravity conditions. These blocks can be rearranged as in the graphic displayed in Figure B.1

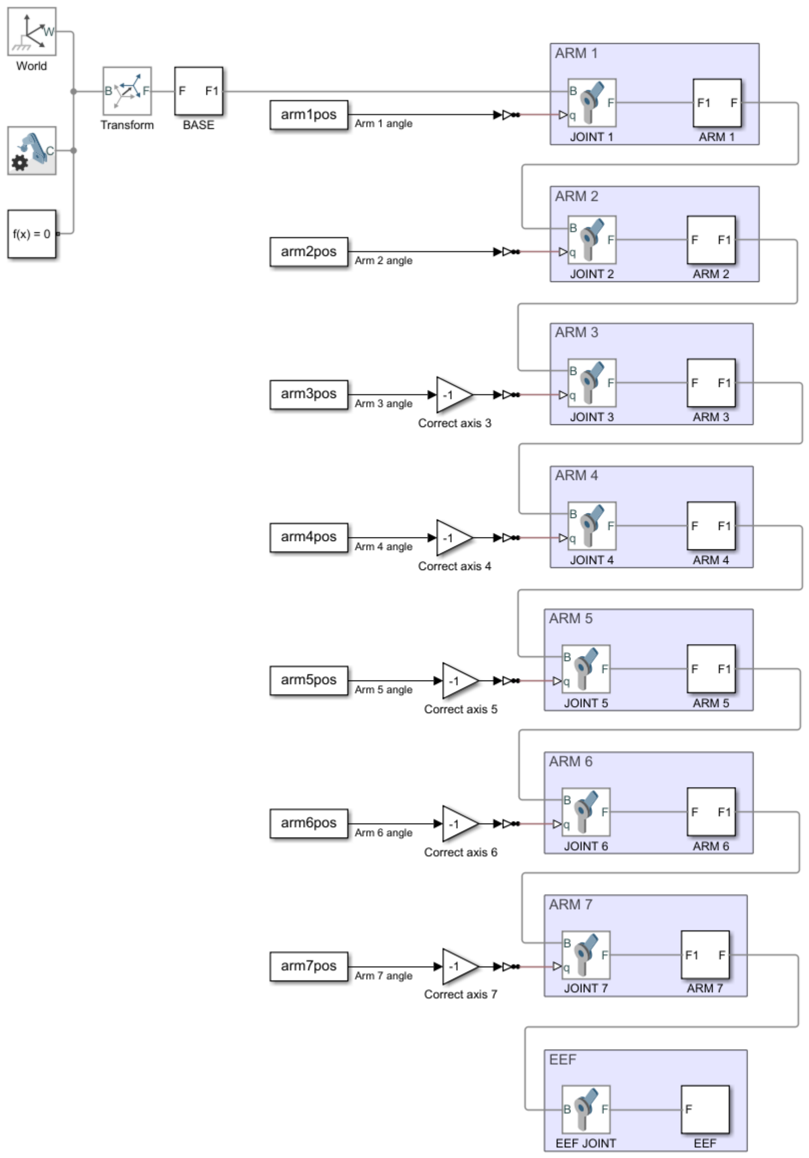
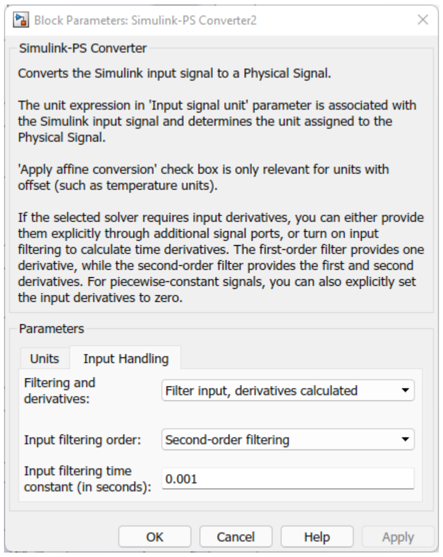


Figure B.1: Simulink block diagram of the LBR iiwa.

The inputs to the joints are angles specified in the workspace corresponding to the Simulink model, imported using `from workspace` blocks. It is important to note that the position data must be converted to a Simscape compatible format using a `Simulink PS Converter`, and a second derivative needs to be applied as follows so that Simulink calculates velocity and acceleration automatically based on the timestep.


Block Parameters: Simulink-PS Converter2

Simulink-PS Converter

Converts the Simulink input signal to a Physical Signal.

The unit expression in 'Input signal unit' parameter is associated with the Simulink input signal and determines the unit assigned to the Physical Signal.

'Apply affine conversion' check box is only relevant for units with offset (such as temperature units).

If the selected solver requires input derivatives, you can either provide them explicitly through additional signal ports, or turn on input filtering to calculate time derivatives. The first-order filter provides one derivative, while the second-order filter provides the first and second derivatives. For piecewise-constant signals, you can also explicitly set the input derivatives to zero.

Parameters

Units | Input Handling

Filtering and derivatives: Filter input, derivatives calculated

Input filtering order: Second-order filtering

Input filtering time constant (in seconds): 0.001

Figure B.2: `Simulink-PS Converter` with second derivatives.

Another aspect to note is the axis correction blocks, which are constant gains set to -1. It is common for imported models to be assigned rotation $\pm$ conventions that are opposite from the user's intended conventions, and a gain can be used to correct this. Next, joint configurations with respect to time can be determined using forward and inverse kinematics. The following code is used to create position and time step vectors for a desired joint configuration and relative velocity

```
function [tend, t, pos] = poscalc(pos_in, relVel, dt)
% calculates trajectory for all 7 DOF in radians using
   forward kinematics

% INPUTS
% pos_in - desired final joint configuration, radians
% relVel - desired relative velocity of joints, rads/s
% dt - desired time step, seconds
% OUTPUTS
% tend - simulation end time, seconds
% t - vector of all time values, seconds
% pos - angular position of the servos, radians

tend = max(abs(pos_in))/relVel;         % determine end time
t = 0:dt:tend;                          % set up time step
   vector
pos = zeros(length(t),length(pos_in)); % allocate empty
   position matrix

for i = 1:length(pos_in)                % calculate angle
   positions
    sigfig = numel(num2str(dt));
    tend_temp = round(abs(pos_in(i))/relVel,sigfig);
    rep_start = length(0:dt:tend_temp);
    pos(rep_start:length(t),i) = pos_in(i);
    for j = 2:rep_start
        pos(j,i) = pos(j-1,i) + relVel*dt*sign(pos_in(i));
    end
end
end
```

And the inverse kinematics for a given stochastic path can be calculated either using the MATLAB robotics toolbox, or one of the algoritms provided in this thesis. An example setup for the MATLAB robotics toolbox is given:

```
% create the robot rigidBodyTree
robot = rigidBodyTree;

% create the base
arm0 = rigidBody('arm0');
jnt0 = rigidBodyJoint('jnt0','fixed');
setFixedTransform(jnt0,[0 0 0 pi],"dh");
arm0.Joint = jnt0;
addBody(robot,arm0,'base'); % add arm 0 to base

% create the 1st arm
arm1 = rigidBody('arm1');
jnt1 = rigidBodyJoint('jnt1','revolute');
jnt1.HomePosition = q(1);
```

```
setFixedTransform(jnt1,[a(1) alpha(1) d(1) 0],"dh");
arm1.Joint = jnt1;
addBody(robot,arm1,'arm0'); % add arm 1 to arm 0

% create the 2nd arm
arm2 = rigidBody('arm2');
jnt2 = rigidBodyJoint('jnt2','revolute');
jnt2.HomePosition = q(2);
setFixedTransform(jnt2,[a(2) alpha(2) d(2) 0],"dh");
arm2.Joint = jnt2;
addBody(robot,arm2,'arm1'); % add arm 2 to arm 1

% create the 3rd arm
arm3 = rigidBody('arm3');
jnt3 = rigidBodyJoint('jnt3','revolute');
jnt3.HomePosition = q(3);          % rotation about z-axis
setFixedTransform(jnt3,[a(3) alpha(3) d(3) 0],"dh");
arm3.Joint = jnt3;
addBody(robot,arm3,'arm2'); % add arm 3 to arm 2

% create the 4th arm
arm4 = rigidBody('arm4');
jnt4 = rigidBodyJoint('jnt4','revolute');
jnt4.HomePosition = q(4);          % rotation about z-axis
setFixedTransform(jnt4,[a(4) alpha(4) d(4) 0],"dh");
arm4.Joint = jnt4;
addBody(robot,arm4,'arm3'); % add arm 4 to arm 3

% create the 5th arm
arm5 = rigidBody('arm5');
jnt5 = rigidBodyJoint('jnt5','revolute');
jnt5.HomePosition = q(5);          % rotation about z-axis
setFixedTransform(jnt5,[a(5) alpha(5) d(5) 0],"dh");
arm5.Joint = jnt5;
addBody(robot,arm5,'arm4'); % add arm 5 to arm 4

% create the 6th arm
arm6 = rigidBody('arm6');
jnt6 = rigidBodyJoint('jnt6','revolute');
jnt6.HomePosition = q(6);          % rotation about z-axis
setFixedTransform(jnt6,[a(6) alpha(6) d(6) 0],"dh");
arm6.Joint = jnt6;
```

```
addBody(robot,arm6,'arm5'); % add arm 6 to arm 5

% create the 7th arm
arm7 = rigidBody('arm7');
jnt7 = rigidBodyJoint('jnt7','revolute');
jnt7.HomePosition = q(7);          % rotation about z-axis
setFixedTransform(jnt7,[a(7) alpha(7) d(7) 0],"dh");
arm7.Joint = jnt7;
addBody(robot,arm7,'arm6'); % add arm 7 to arm 6

% create the end effector
EEF = rigidBody('EEF');
jnt8 = rigidBodyJoint('jnt8','revolute');
setFixedTransform(jnt8,[0 0 0 0],"dh");
EEF.Joint = jnt8;
addBody(robot,EEF,'arm7'); % add end effector to arm 7
```

Where inverse kinematics can be calculated for each time step by the following

```
config = homeConfiguration(robot)
% calculate transform from robot base to EEF
tform0EEF = getTransform(robot,config,'EEF','base')
% create inverse kinematics object
ik = inverseKinematics('RigidBodyTree',robot,'
   SolverAlgorithm','LevenbergMarquardt');
% create weights for each joint
weights = [1 1 1 1 1 1];
% set initial guess to home configuration
initialguess = robot.homeConfiguration;
for i = 1:length(t_exp)
    % create a matrix of zeros, add xyz from experiment to
       DH matrices
    IK_new = zeros(4);
    IK_new(1,4) = x(i)*10^-3;
    IK_new(2,4) = y(i)*10^-3;
    IK_new(3,4) = z(i)*10^-3;
    % update xyz coordinates, keep orientation the same
    IK_tform{i} = tform0EEF + IK_new;
    configSoln = struct2cell(ik('EEF',IK_tform{i},weights,
       initialguess));
    for j = 1:length(d)
        qSoln(j,i) = cell2mat(configSoln(2,1,j));
    end
    qSoln(7,i) = qSoln(7,i) + cell2mat(configSoln(2,1,8));
end
```

Alternatively, the BFGS algorithm described in section 3.1.3 can be used as follows

```
epsilon = 1e-3;   % define acceptable error
alpha0 = 1;        % intial guess for step size
beta = 0.5;        % step size scaling factor
% value for c1
% note: 0 < c1 < c2 < 1
c1 = 1e-4;
% initial Hessian estimate
H = eye(7);

% specify initial estimate for joint configuration
q10 = 0*pi/180;
q20 = -7*pi/180;
q30 = 0*pi/180;
q40 = -70*pi/180;
q50 = 0*pi/180;
q60 = 120*pi/180;
q70 = 0*pi/180;
q0 = [q10 q20 q30 q40 q50 q60 q70];
% set up desired orientation and position for end effector
Td = DHcalc(a,alpha,d,q0,o);

for i = 1:length(t_exp)
    % create a matrix of zeros, add xyz from experiment to
       DH matrices
    IK_new = zeros(4);
    IK_new(1,4) = x(i)*10^-3;
    IK_new(2,4) = y(i)*10^-3;
    IK_new(3,4) = z(i)*10^-3;
    % update xyz coordinates, keep orientation the same
    IK_tform{i} = Td + IK_new;
    IK_tform_vec{i} = reshape(IK_tform{i}(1:3,:),[12,1]);
end
IK_tform_vec = cell2mat(IK_tform_vec);
for i = 1:length(t_exp)
    [q_BFGS, k_BFGS, qcheck_BFGS,BFGS_time] = IK_BFGS_iiwa(
       q0,H,epsilon,alpha0,beta,c1,W,IK_tform_vec(:,i),d,a,
       alphar,o);
    qSoln(:,i) = q_BFGS;
```

```
end
```

Last, these inputs must be formatted for the Simscape `from workspace` blocks as follows.

```
% determine simulation end time
tend = max(t_exp)+t_pos_i(end);
t_sim = [t_pos_i, [t_exp + t_pos_i(end)]'];

% assign joint angles calculated using IK as vectors for
   Simulink
% input to ARM 1
arm1pos.time = t_sim;
arm1pos.signals.values = [pos_i(:,1); qSoln(1,:)'];
arm1pos.signals.dimensions = 1;

% input to ARM 2
arm2pos.time = t_sim;
arm2pos.signals.values = [pos_i(:,2); qSoln(2,:)'];
arm2pos.signals.dimensions = 1;

% input to ARM 3
arm3pos.time = t_sim;
arm3pos.signals.values = [pos_i(:,3); qSoln(3,:)'];
arm3pos.signals.dimensions = 1;

% input to ARM 4
arm4pos.time = t_sim;
arm4pos.signals.values = [pos_i(:,4); qSoln(4,:)'];
arm4pos.signals.dimensions = 1;

% input to ARM 5
arm5pos.time = t_sim;
arm5pos.signals.values = [pos_i(:,5); qSoln(5,:)'];
arm5pos.signals.dimensions = 1;

% input to ARM 6
arm6pos.time = t_sim;
arm6pos.signals.values = [pos_i(:,6); qSoln(6,:)'];
arm6pos.signals.dimensions = 1;

% input to ARM 7
arm7pos.time = t_sim;
arm7pos.signals.values = [pos_i(:,1);qSoln(7,:)'];
```

```
arm7pos.signals.dimensions = 1;
```

Running the Simulink model for a given experimental path and initial configuration results in the following.

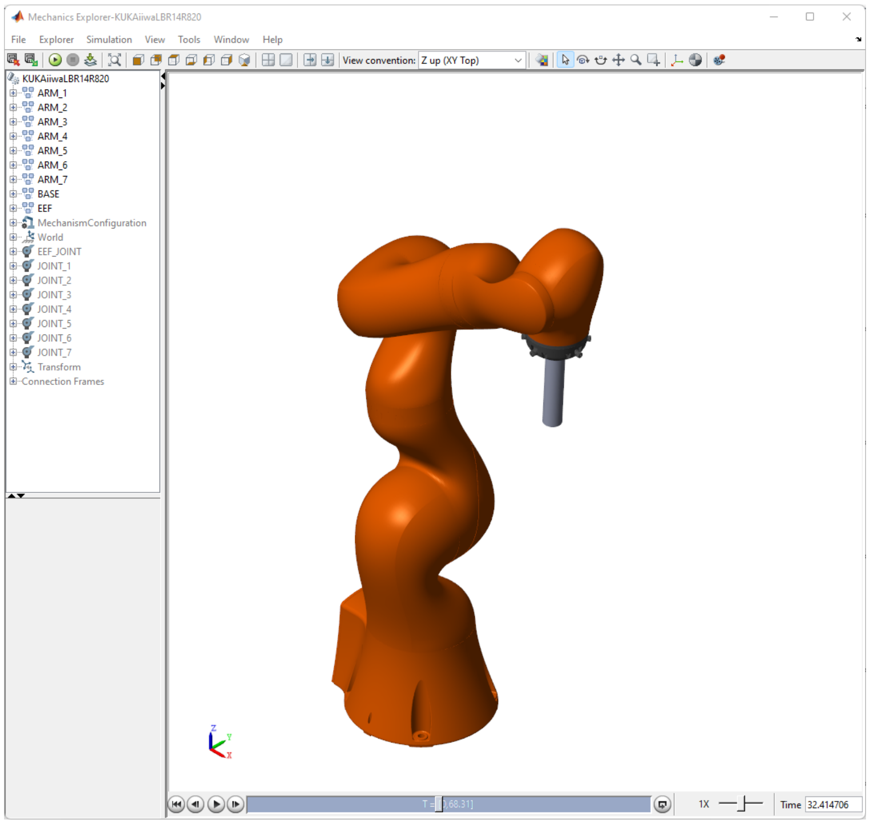


Figure B.3: Simscape multibody model of the LBR iiwa in the MoBL EPS configuration.