Inverse Kinematics with Dual-Quaternions, Exponential-Maps, and Joint Limits

Ben Kenwright
Newcastle University
School of Computing Science
United Kingdom
b.kenwright@ncl.ac.uk

***Abstract*—We present a novel approach for solving articulated inverse kinematic problems (e.g., character structures) by means of an iterative dual-quaternion and exponential-mapping approach. As dual-quaternions are a break from the norm and offer a straightforward and computationally efficient technique for representing kinematic transforms (i.e., position and translation). Dual-quaternions are capable of represent both translation and rotation in a unified state space variable with its own set of algebraic equations for concatenation and manipulation. Hence, an articulated structure can be represented by a set of dual-quaternion transforms, which we can manipulate using inverse kinematics (IK) to accomplish specific goals (e.g., moving end-effectors towards targets). We use the projected Gauss-Seidel iterative method to solve the IK problem with joint limits. Our approach is flexible and robust enough for use in interactive applications, such as games. We use numerical examples to demonstrate our approach, which performed successfully in all our test cases and produced pleasing visual results.**

Keywords-Inverse Kinematics; Gauss-Seidel; Articulated Character; Games; Joint Limits; Iterative; Dual-Quaternion; Jacobian; Exponential-Map

I. Introduction

Generating fast reliable Inverse Kinematic (IK) solutions in real-time with angular limits for highly articulated figures (e.g., human bipeds including hands and feet) is challenging and important [1, 2, 3, 4, 5]. The subject is studied across numerous disciplines, such as graphics, robotics, and biomechanics, and is employed by numerous applications in the film, animation, virtual reality, and game industry

However, articulated models (e.g., bipeds and hands) can be highly complex; even the most simplified models of 20-30 joints can generate a vast number of poses [6, 7]. Whereby producing a simple pose to achieve a solitary task can produce ambiguous solutions that make the problem highly nonlinear and computationally expensive to solve. For example, even a straightforward task of reaching to pickup an object can be accomplished by means of any number of motions.

This paper focuses using dual-quaternions and quaternion exponential-maps with an iterative Gauss-Seidel algorithm [8] to solve an articulated IK problem; such as the hand model shown in Figure 2. The Gauss-Seidel algorithm is an iterative, efficient, low memory method of solving linear systems of equations of the form $Ax = b$. Hence, we integrate the Gauss-Seidel iterative algorithm with an articulated IK problem to produce a flexible whole system IK solution for time critical systems, such as games. This method is used as it offers a flexible, robust solution with the ability to trade accuracy for speed and give good visual outcomes.

Furthermore, to make the Gauss-Seidel method a practical IK solution for an articulated hand structure, it needs to enforce joint limits. We incorporate joint limits by modifying the update scheme to include an iterative projection technique. Additionally, to ensure real-time speeds we take advantage of spatial coherency between frames as a warm starting approximation for the solver. Another important advantage of the proposed method is the simplicity of the algorithm and how it can be easily configured for custom IK problems.

The main contribution of the paper is the practical demonstration and discussion of using the Gauss-Seidel method for real-time articulated IK problem with joint limits, dual-quaternions [9], and exponential-quaternion mapping [10]. Furthermore, we discuss constraint conditions, speedup approaches and robustness factors for solving highly non-linear IK problems in real-time.

The roadmap for rest of the paper is organized as follows. Firstly, we briefly review existing work in Section II. Section III describes the articulated model, we use for our simulations. Then in Section IV, we present essential mathematical algorithms and principles for the paper (e.g., dual-quaternion algebra). We follow on by explaining the IK problem in Section V. While in Section VI, we explain the Jacobian matrix, then in Section VII we discuss our approach for solving the IK problem with the Gauss-Seidel algorithm. Finally, we present results in Section IX, then Section X discusses limitations, followed by the closing conclusion and discussion in Section XI.

II. Related Work

Inverse kinematics is a popular problem across numerous disciplines (e.g., graphical animation, robotics, biomechan-

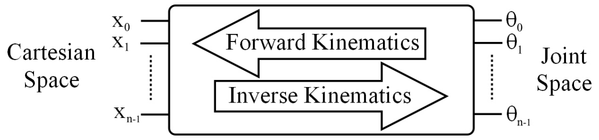

Figure 1. Forward & Inverse Kinematics. Illustrating the relationship between forward and inverse kinematics parameters.

ics). IK is a vital component that can be implemented using a wide range of solutions. We give a brief overview of existing, current, and cutting-edge approaches to help emphasise the different ways of approaching the problem; enabling the reader to see where our method sits.

In general, however, for very simple problems with just a few links, analytical methods are employed to solve the IK problem. Alternatively, for larger configurations, iterative numerical methods must be employed due to the complexity of the problem.

The articulated IK problem of finding a solution for poses that satisfy positional and orientation constraints has been well studied, e.g., [11, 3, 12, 7]. The problem is highly nonlinear, meaning there can be numerous solutions; hence, multiple poses fulfilling the constraint conditions. In practical situations, there can even be cases where no solution exists due to the poor placement of end-effectors. IK systems typically use cut down models, e.g., merely performing IK on individual limbs (as in body, arms, legs) [13, 14, 6]. This makes the problem computationally simpler and less ambiguous.

Numerous solutions from various fields of research have been implemented to solving the IK problem. The Jacobian-based matrix approach is one of the most popular methods and the method upon which we base our iterative solution [7, 15, 16]. The Jacobian matrix method aims to find a linear approximation to the problem by modelling the end-effectors movements relative to the instantaneous systems changes of the links translations and orientations. Numerous different methods have been presented for calculating the Jacobian inverse, such as, Jacobian Transpose, Damped Least-Squares (DLS), Damped Least-Squares with Singular Value Decomposition (SVD-DLS), Selectively Damped Least-Square (SDLS) [17, 5, 18, 19, 20, 15].

An alternative method uses the Newton method; whereby the problem is formulated as a minimization problem from which configuration poses are sought. The method has the disadvantage of being complex, difficult to implement and computationally expensive to computer-per-iteration [16].

The Cyclic Coordinate Descent (CCD) is a popular real-time IK method used in the computer games industry [21]. Originally introduced by Wang et al. [22] and then later extended to include constraints by Welman et al. [3]. The CCD method was designed to handle serial chains and is thus difficult to extend to complex hierarchies. It has the advantage of not needing to formulate any matrices and has a lower computational cost for each joint per iteration. Its downside is that the character poses even with constraints can produce sporadic and unrealistic poses. However, further work has been done to extend CCD to work better with human based character hierarchies [4, 6, 23].

A novel method recently proposed was to use a Sequential Monte Carlo approach but was found to be computationally expensive and only applicable for offline processing [24, 25].

Data driven IK systems have been presented; Grochow et al. [26] method searched a library of poses to determine an initial best guess solution to achieve real-time results. An offline mesh-based for human and non-human animations was achieved by learning the deformation space; generating new shapes while respecting the models constrains [27, 28].

A method known as ”Follow-The-Leader” (FTL) was presented by Brown et al. [29] and offered real-time results using a non-iterative technique. However, this approach was later built upon by Aristidou et al. [30] and presented an iterative version of the solver known as FABRIK.

The Triangular IK method [31, 32], uses trigonometric properties of the cosine rule to calculate joint angles, beginning at the root and moving outwards towards the end-effectors. While the algorithm can be computationally fast, due to it being able to propagate the full hierarchy in a single iteration, it cannot handle multiple end-effectors well and is primarily based around singly linked systems.

The advantages of an iterative IK system for articulated structures, such as character, was also presented by the interesting paper by Tang et al. [33] who explored IK techniques for animation using a method based on the SHAKE algorithm. The SHAKE algorithm is an iterative numerical integration scheme considered similar to the Verlet method [34], which can exploit substantial step-sizes to improve speed yet remain stable when solving large constrained systems. The algorithm is also proven to have the same local convergence criterion as the Gauss-Seidel method we present here as long as the displacement size is kept sufficiently small.

The paper by Arechavaleta et al. [35] presents a well written explanation of using iterative methods (primarily the Gauss-Seidel technique) for computing fast and accurate solutions for ill-conditioned LCP problems.

We use the iterative Gauss-Seidel approach presented by Kenwright [1], however, we store the joint angles as quaternion-exponent and employ dual-quaternion algebra [9] for solving forward and inverse kinematic problems. As unit-quaternions are an ideal tool for orientations of rigid transforms, however, they do not contain any translation information about the location of points in 3D space. Hence, dual-quaternions are an extension of quaternion by means of dual-number theory as a compact, efficient, and smart approach of representing both rotation and translation in a single vector with its own algebraic rules (e.g., calculating

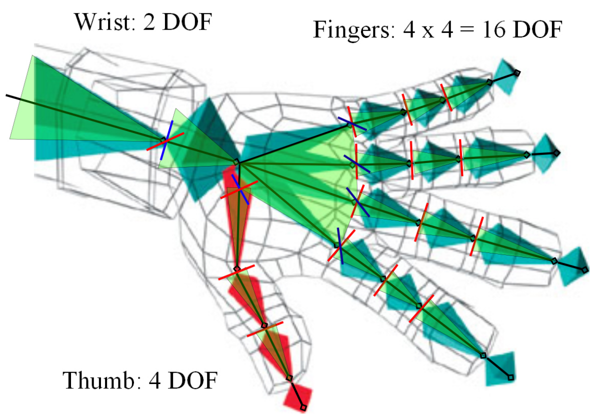

Figure 2. Articulated Hand Model. The articulated hand model (including the wrist) was used for the simulations and possessed 22 degrees of freedom (DOF). The structure comprised of 17 links and 16 joints.

differences, inverting, interpolating). The robotic community has exploited dual-quaternions to solve IK problems; for example, Hoang-Lan et al. [36] solved IK problem by formulating a dual-quaternion metric error measurement for constructing the Jacobian.

III. Articulated Model

The articulated model used for our simulations and evaluation of our approach is shown in Figure 2. The mechanical functioning is of a human hand and is constructed using a series of interconnected rigid segments (or links) connected by joints (also, note, an interconnected series of links is also called a kinematic chain). As shown in Figure 2, we represent the hand as a collection of 17 rigid body segments connected using 16 primary joints. The character's hand gives us 22 degrees of freedom (DOF). Joints such as the wrist have three DOF corresponding to abduction/adduction, flexion/extension and internal/external rotation (i.e., rotation around the x, y, and z axis). Where complex joints such as the ball-and-socket (i.e., with 3 DOF) can be formed from multiple simpler joints (i.e., 3 single DOF joints). So a joint with n DOF is equivalent to n joints of 1 DOF connected by n-1 links of length zero. Thus, for example, the wrist joint can be described as a sequence of 2 separate joints of 1 DOF, where 1 of the joints connecting links has a zero length, as done by Kenwright [1]. Euler angles are an intuitive and straightforward means of representing orientation, since they are easy to visualize and enforce upper and lower boundary limits. However, we store each joint rotation as a quaternion-exponential (i.e., axis-angle combination), as it offers a similarly compact parameterization as Euler angles but without the gimbals lock problem. We combine the exponential-mapping with quaternion and dual-quaternion algebra to solve the IK solution, while clamping angular limits through a twist-and-swing decomposition of the orientation.

As shown in Figure 2, the single DOF connected joints were colored in accordance with their axis type; the x, y, and z representing the colors red, green and blue. The foot was set as the base for the IK with five end-effectors (i.e., head, pelvis, right-hand, left-hand and left-foot). We developed an application for an artist to interrogate and experiment with the skeletal IK system; setting end-effectors locations and viewing the generated poses. Each end-effector has a 6 DOF constraint applied to it; representing the target position and orientation. The ideal end-effectors are drawn in red, and the current end-effectors are drawn in green. This can be seen clearly in Figure 5, where the target end-effectors are located at unreachable goals.

IV. Mathematical Background

We give a brief introduction to the essential mathematical definitions on quaternion and dual-quaternion algebra. For a more detailed introduction and an overview of their practical advantages, see Kenwright [9]. Since quaternions have proven themselves in many fields of science and computing as providing an unambiguous, un-cumbersome, computationally efficient method of representing rotational information. We combine dual-number theory to extend quaternions to dual-quaternions, so we can use them to represent rigid transforms (i.e., translations and rotations).

A. Definitions

To reduce ambiguity and make the paper as readable as possible, we define variable symbol definitions:

$$\begin{array}{rr} q \text{ quaternion} & \hat{q} \text{ unit-quaternion} \\ \underline{q} \text{ dual-quaternion} & \hat{\underline{q}} \text{ unit dual-quaternion} \\ \vec{v} \text{ vector} & \hat{v} \text{ unit-vector} \end{array} \tag{1}$$

While we mostly represent quaternions and dual-quaternions with the letter q, there are instances where we use the letter t to indicate that the dual-quaternion or quaternion is a pure translation component.

B. Quaternions

The quaternion was introduced by Hamilton [37] in 1860 and has the following form:

$$\mathbf{q} = q_s + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k} \quad (q_s, q_x, q_y, q_z \in \mathbb{R}) \tag{2}$$

where $\mathbf{ii} = \mathbf{jj} = \mathbf{kk} = \mathbf{ijk} = -1$.

While w is sometimes used to represent the scalar component in quaternions, we us the letter s, to avoid ambiguity with exponential-mapping variable ($\vec{w}$). Alternatively, it is more commonly defined as a pair $(s, \vec{v})$ with $s \in \mathbb{R}$ and $\vec{v} \in \mathbb{R}^3$.

A *unit-quaternion* has a unit-length with $q_s^2 + q_x^2 + q_y^2 + q_z^2 = 1$ and the inverse of a unit-quaternion is its conjugate

$q^* = q_s - \vec{v}$. Given an axis and angle of rotation a unit-quaternion can be calculated using:

$$\begin{aligned} q_s &= cos(\theta/2), & q_x &= n_x sin(\theta/2) \\ q_y &= n_y sin(\theta/2), & q_z &= n_z sin(\theta/2) \end{aligned} \tag{3}$$

where $\hat{n} = n_x, n_y, n_z$ is a unit-vector representing the axis of rotation and θ is the angle of rotation. Whereby, given a point in 3D space $\vec{x}$ we can rotate to give $\vec{x}'$ using:

$$\vec{x}' = \hat{q}\vec{x}\hat{q}^* \tag{4}$$

Addition, subtraction, and the product of two quaternions, is defined by:

$$\begin{aligned} \mathbf{q_0} + \mathbf{q_1} &= (s_0 + s_1, \vec{v_0} + \vec{v_1}) \\ \mathbf{q_0} - \mathbf{q_1} &= (s_0 - s_1, \vec{v_0} - \vec{v_1}) \\ \mathbf{q_0}\mathbf{q_1} &= (s_0 s_1 - \vec{v_0}\vec{v_1}, s_0\vec{v_1} + s_1\vec{v_0} + \vec{v_0}\vec{v_1}) \end{aligned} \tag{5}$$

A crucial fact that we exploit in this paper is that the exponential of a unit-quaternion is the combined axis-angle component:

$$\begin{aligned} exp(\hat{q}) &= [0, \hat{n}\theta] \\ &= [0, \vec{w}] \end{aligned} \tag{6}$$

where $\hat{n}$ is the unit-vector representing the axis of rotation and θ is the angle magnitude in radians. The logarithm of a quaternion is the inverse of the exponential enabling us to convert to and from the axis-angle component. The exponential of a unit-quaternion is often called the *exponential map*, which we denote as $\vec{w}$.

$$\begin{aligned} \theta &= ||\vec{w}|| \\ \hat{v} &= \frac{\vec{w}}{||\vec{w}||} \end{aligned} \tag{7}$$

The exponential-map can be computed robustly, even in the neighborhood of the origin [10].

C. Dual-Numbers

Dual-number theory was introduced by Clifford [38] in 1873 and is defined as:

$$\hat{z} = r + \epsilon d \quad \text{with } \epsilon^2 = 0 \text{ but } \epsilon \neq 0 \tag{8}$$

where r is the real-part, d is the dual-part, and ϵ is the dual operator. While dual-number theory can be used to represent different quantities (e.g., dual-vectors), we are primarily interested in dual-quaternions because it gives us the ability of unifying rigid transform space into a single state-space variable (i.e., position and translation).

D. Dual-Quaternions

A dual-quaternion is defined as a dual-number with quaternion components and has the ability to represent 3D Euclidean coordinate space (i.e., rotation and translation) as a single parameter.

$$\underline{q} = q_r + \epsilon q_d \tag{9}$$

where q_r and q_d are quaternions. Additionally, since the dual-quaternion consists of two quaternion components it can be represented as: $\underline{q} = [q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7]^T$. The common algebraic operations are defined as:

$$\begin{aligned} \alpha\underline{q} &= \alpha q_r + \alpha\epsilon q_d \\ \underline{q}_0 + \underline{q}_1 &= q_{r0} + q_{r1} + \epsilon(q_{d0} + q_{d1}) \\ \underline{q}_0\,\underline{q}_1 &= q_{r0}q_{r1} + \epsilon(q_{r0}q_{d1} + q_{d0}q_{r1}) \end{aligned} \tag{10}$$

The conjugate of a dual-quaternion $\underline{q}^* = q_r^* + \epsilon q_d^*$ with the norm (or length) of a dual-quaternion given by $||\underline{q}|| = \underline{qq^*}$ and the unity condition (i.e., for unit dual-quaternions) is:

$$\underline{q}\,\underline{q}^* = 1 \quad q_r^* q_d + q_d^* q_r = 0 \tag{11}$$

A **unit dual-quaternion** can be used to represent any rigid transformation (i.e., position and rotation); we construct a unit dual-quaternion rigid transformation using:

$$\begin{aligned} &\hat{\underline{q}} = q_{rot} + \epsilon\frac{1}{2}q_{rot}q_{pos} && \text{(rotation then translation)} \\ &\qquad\qquad\text{or} \\ &\hat{\underline{q}} = q_{rot} + \epsilon\frac{1}{2}q_{pos}q_{rot} && \text{(translation then rotation)} \\ &\quad = (1 + \epsilon\frac{1}{2}q_{pos})q_{rot} \end{aligned} \tag{12}$$

where q_{rot} and q_{pos} are the rotation and translation quaternions respectively, with the translation quaternion $q_{pos} = [0, t_x, t_y, t_z]$.

V. Forward and Inverse Kinematics

Forward and inverse kinematics is the process of calculating positions and orientations either from joint space (i.e., using interconnected positions and orientations of the joints) or from Cartesian space (i.e., the world positions and orientations) as shown in Figure 1.

A. Forward Kinematics (FK)

The FK problem is straightforward to calculate and has no ambiguity or singularities. For an articulated structure, we can concatenate the dual-quaternion transforms through multiplication to generate the final positions and orientations of the interconnected links. For example, a serial chain of n links would be:

$$\hat{\underline{q}} = \hat{\underline{q}}_0\hat{\underline{q}}_1\hat{\underline{q}}_2 \cdots \hat{\underline{q}}_{n-1} \tag{13}$$

where $\hat{\underline{q}}_0 \dots \hat{\underline{q}}_{n-1}$ define each individual joints rotation and translation.

B. Inverse Kinematics (IK)

IK is the reverse of FK. While FK remains fast and simple for large interconnected structures, IK solutions can be computationally expensive, possess singularity problems and contain multiple solutions. However, in practice, we attempt to find a best fit approximation that will meet the desired constraints. For example, in Equation (13), we know $\underline{\hat{q}}$ and would seek to find the orientation (and/or translation) for each link ($\underline{\hat{q}}_0..\underline{\hat{q}}_{n-1}$), and $\underline{\hat{q}}$ is the position and orientation of the combined links (i.e., for the end-effector).

C. Pure Rotation and Pure Translation

A dual-quaternion's transformation can be represented by two pure dual-quaternions, i.e., a *pure rotation* and *pure translation*:

$$\begin{aligned} \underline{\hat{q}}_{tra} &= 1 + \epsilon \frac{1}{2} \hat{q}_{pos} \quad \text{(pure translation)} \\ \underline{\hat{q}}_{rot} &= \hat{q}_{rot} + \epsilon \hat{0} \quad \text{(pure rotation)} \end{aligned} \tag{14}$$

where we concatenate the pure dual-quaternion transforms by multiplication to calculate the combined set of transforms, as shown in Equation (12); however, be warned the order of multiplication determines if translation or rotation is performed first. For example, we represent the FK problem as:

$$\begin{aligned} \underline{\hat{q}} &= \underline{\hat{q}}_0 \underline{\hat{q}}_1 \underline{\hat{q}}_2 ... \underline{\hat{q}}_{n-1} \\ &= (\underline{\hat{q}}_0^t \underline{\hat{q}}_0^r)(\underline{\hat{q}}_1^t \underline{\hat{q}}_1^r)(\underline{\hat{q}}_2^t \underline{\hat{q}}_2^r)...(\underline{\hat{q}}_{n-1}^t \underline{\hat{q}}_{n-1}^r) \end{aligned} \tag{15}$$

where the superscript letter r and t indicates a pure rotation or pure translation dual-quaternion transform respectively (as defined in Equation (14)).

VI. Jacobian Matrix

The Jacobian $\mathbf{J}$ is a matrix that represents the change in joint orientations to displacement of end-effectors. Each frame we calculate the Jacobian matrix from the current angles and end-effectors. We assume a right-handed coordinate system. To illustrate how we calculate the Jacobian for an articulated system, we consider the simple example shown in Figure 3. For a more detailed description see [17, 5, 18, 19, 20, 15]. The example demonstrates how we decompose the problem and represent it as a matrix for a sole linked chain with a single three DOF end-effector. We then extend this method to multiple linked-chains with multiple end-effectors (each with six DOF) to represent the character hierarchy.

Each joint is stored as an axis-angle component ($w = \hat{n}\theta$):

$$\mathbf{w} = \begin{bmatrix} \vec{w}_0 \\ \vec{w}_1 \\ \vec{w}_2 \\ ... \\ \vec{w}_n \end{bmatrix} \tag{16}$$

where $\vec{w}_i$ is the rotation (i.e., axis-angle) of joint i relative to joint $i-1$, and e for the end-effectors global position. From these matrices, we can determine that the end-effectors, and the joint angles are related. This leads to the forward kinematics definition, defined as:

$$\mathbf{e} = \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix} \tag{17}$$

The end-effectors and the joint orientation (i.e., quaternion-exponent) are related and is defined by:

$$\mathbf{e} = f(\ \mathbf{w}\ ) \tag{18}$$

We can differentiate the kinematic equation for the relationship between end-effectors and joint orientation. This relationship between a change in joint orientation and a change in end-effectors location is represented by the Jacobian matrix and is given by:

$$\dot{\mathbf{e}} = \mathbf{J}\dot{\mathbf{w}} \tag{19}$$

The Jacobian $\mathbf{J}$ is the partial derivatives for the change in end-effectors locations by change in joint angles.

$$\mathbf{J} = \frac{\partial \mathbf{e}}{\partial \mathbf{w}} \tag{20}$$

If we re-arrange the kinematic problem:

$$\mathbf{w} = f^{-1}(\mathbf{e}) \tag{21}$$

We can conclude a similar relationship for the Jacobian:

$$\dot{\mathbf{w}} = \mathbf{J}^{-1}\dot{\mathbf{e}} \tag{22}$$

For small changes, we can approximate the differentials by their equivalent deltas:

$$\Delta \mathbf{e} = \mathbf{e}_{target} - \mathbf{e}_{current} \tag{23}$$

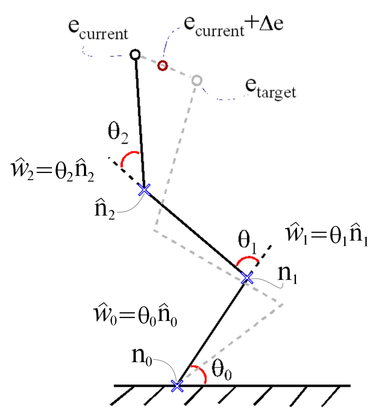

Figure 3. Forward & Inverse Kinematics Example. Illustrating the relationship between the different parameters, e.g., end-effector error and joint orientations. $\vec{w}$ quaternion exponential-map for each joint (i.e., axis-angle combination $\vec{w} = \hat{n}\theta$).

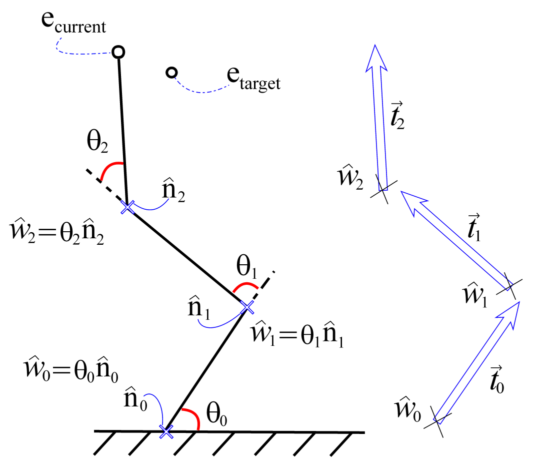

Figure 4. A simple three link serial chain example, where $\vec{w}$ is the axis-angle combination (i.e., quaternion- exponential-map), $\vec{t}$ is the translation vector, $\hat{n}$ is the unit-length axis of rotation, and θ is the angle. For a simple 2D case (i.e., only in the x-y axis) the axis of rotation is $\hat{n} = [0, 0, 1]$..

For these small changes, we can then use the Jacobian to represent an approximate relationship between the changes of the end-effectors with the changes of the joint angles.

$$\Delta \mathbf{w} = \mathbf{J}^{-1} \Delta \mathbf{e} \tag{24}$$

We can substitute the result back in:

$$\mathbf{w}_{current} = \mathbf{w}_{previous} + \Delta \mathbf{w} \tag{25}$$

For a step-by-step explanation of the process of calculating the Jacobian for the quaternion-exponent see Appendix B-B. For example, calculating the Jacobian for Figure 4 gives:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{e}}{\partial w_0} \\ \frac{\partial \mathbf{e}}{\partial w_1} \\ \frac{\partial \mathbf{e}}{\partial w_2} \end{bmatrix} \tag{26}$$

and

$$\mathbf{e} = \mathbf{e}_{current} - \mathbf{e}_{target}$$

The Jacobian matrix is calculated for the system so that we can calculate the inverse and hence the solution. Alternatively, a good explanation of the Jacobian and its applications is also presented by Buss [15], who gives an introduction to IK methods using the Transpose, Pseudoinverse, and Damped Least-Square method.

VII. Gauss-Seidel Algorithm

We set up the IK problem into a particular arrangement, so that we can solve for the unknowns using the Gauss-Seidel method. Whereby, we construct the IK formulation using the Jacobian matrix with the linear equation format of the form:

$$\text{Linear Equation:} \quad \mathbf{Ax} = \mathbf{b} \tag{27}$$

The IK problem is then composed as:

$$\mathbf{J}^T \mathbf{J} \Delta \mathbf{q} = \mathbf{J}^T \Delta \mathbf{e} \tag{28}$$

Equating equivalent variables:

$$\begin{aligned} \mathbf{A} &= \mathbf{J}^T \mathbf{J} \\ \mathbf{b} &= \mathbf{J}^T \Delta \mathbf{e} \\ \mathbf{x} &= \Delta \mathbf{q} = unknown \end{aligned} \tag{29}$$

A. Damping and Stability

1) Damping: With the Gauss-Seidel iterative method, we solve for the unknown x value. To prevent singularities and make the final method more stable and robust we incorporated a damping value:

$$\mathbf{A} = (\mathbf{J}^T \mathbf{J} + \delta \mathbf{I}) \tag{30}$$

where δ is a damping constant (e.g., $\sim$0.0001), and **I** is an identity matrix.

2) Singularities: The exponential-map of a quaternion (i.e., the axis-angle combination) is parameterized in $\in \mathbb{R}^3$ and hence contains singularities similar to Euler angles possessing gimbals lock singularities. However, for our IK iterative situation, we take small incremental steps (i.e., angular change is less than π) we can avoid the singularity problem, since we can shift the exponential-map singularity away from the safe working region [10]. The exponenential-map has singularities at a radius of $2n\pi$ (for $n = 1, 2, 3..$). Hence, if the exponential-map angle $||\vec{w}||$ is close to π we replace $\vec{w}$ by $(1 - 2\pi/||\vec{w}||)\vec{w}$, which is the same rotation but shifted away from the singularity problem.

B. Gauss-Seidel Implementation

The Gauss-Seidel iterative algorithm is a technique developed for solving a set of linear equations of the form $Ax = b$. The method has gained a great deal of acclaim in the physics-based community for providing a computationally fast robust method for solving multiple constraint rigid body problems [39, 40, 41]. The iterative algorithm is based on matrix splitting [42], and its computational cost per iteration is O(n), where n is the number of constraints. Furthermore, the number of constraints and the number of iterations is what dominates the performance of the algorithm. Algorithm 1 is the basic Gauss-Seidel method for a generic linear system of equations of the form $Ax = b$; for the unknowns, an initial guess is needed. Naively this value could be zero and result in the system having a cold start. Then the algorithm would proceed, while at each iteration, the corresponding elements from A, b and x (current) act as a feedback term to move x (next) closer to the solution.

Algorithm 1 Gauss-Seidel iterative algorithm to solve $Ax = b$ given x^0

```
1: $x = x^0$
2: for iter=1 to iteration limit do
3:     for i=1 to n do
4:         $\Delta x_i = \frac{\left[ b_i - \sum_{j=1}^{n} A_{ij} x_j \right]}{A_{ii}}$
5:         $x_i = x_i + \Delta x_i$
6:     end for
7: end for
```

The conditions for the Gauss-Seidel iterative Algorithm 1 terminating are:

- If a maximum number of iterations has been reached
- If the error $||Ax - b||$ drops below a minimum threshold
- If $||\Delta x_i||$ falls below a tolerance
- If $||\Delta x_i||$ remains the same as the previous frame (within some tolerance)

It is essential that the coefficients along the diagonal part of the matrix be dominant for the Gauss-Seidel method to converge on a solution.

C. Angular Limits - Twist-and-Swing

Any practical character-based IK system needs to have the ability to enforce angular joint limits before it can be considered a viable real-world solution. We incorporate angular joint limits into the simple iterative algorithm by clamping the modified angle orientations at each iteration update (see Equation (31)). While this can be accomplished easily with Euler angles by setting a minimum and maximum angle. For the exponential-map (i.e., the axis-angle combination) parameterization, we use the twist-and-swing decomposition [10], since it presented a fast, robust, and simple technique for robustly calculating angular errors and enforcing limits (as demonstrated and shown by Kallmann [43]).

$$\mathbf{w} = \begin{cases} lower & : \ if\,(\mathbf{w} + \mathbf{J}^{-1}\Delta\mathbf{e}) \ < lower \\ upper & : \ if\,(\mathbf{w} + \mathbf{J}^{-1}\Delta\mathbf{e}) \ > upper \\ \mathbf{w} + \mathbf{J}^{-1}\Delta\mathbf{e} & : \ otherwise \end{cases} \tag{31}$$

For complex joint models, such as the ball-and-socket joint, the twist-and-swing decomposition presents a practical and intuitive representation. The twist-and-swing allows us to define and enforce joint limits intuitively. The twist is around the 'x-axis' while the swing is around the 'yz-plane'. We can decompose a quaternion orientation into its twist and swing components shown in Equation 32. This is in world space but can easily be converted to local space (e.g., joint space).

$$\begin{aligned} \mathbf{q}_{twist_x} &= \left(\frac{q_s}{\sqrt{q_s^2 + q_x^2}}, \frac{q_x}{\sqrt{q_s^2 + q_x^2}}, 0, 0 \right) \\ \mathbf{q}_{swing_{yz}} &= \left(\sqrt{q_s^2 + q_x^2}, 0, \frac{q_s q_y - q_x q_z}{\sqrt{q_s^2 + q_x^2}}, \frac{q_s q_z + q_x q_y}{\sqrt{q_s^2 + q_x^2}} \right) \\ \mathbf{q} &= \mathbf{q}_{swing_{yz}} \mathbf{q}_{twist_x} \end{aligned} \tag{32}$$

$$\begin{aligned} \mathbf{q}_{twist_y} &= \left(\frac{q_s}{\sqrt{q_s^2 + q_y^2}}, 0, \frac{q_y}{\sqrt{q_s^2 + q_y^2}}, 0 \right) \\ \mathbf{q}_{swing_{xz}} &= \left(\sqrt{q_s^2 + q_y^2}, 0, \frac{q_s q_x + q_y q_z}{\sqrt{q_s^2 + q_y^2}}, \frac{q_s q_z - q_x q_y}{\sqrt{q_s^2 + q_y^2}} \right) \\ \mathbf{q} &= \mathbf{q}_{swing_{xz}} \mathbf{q}_{twist_y} \end{aligned} \tag{33}$$

$$\begin{aligned} \mathbf{q}_{twist_z} &= \left(\frac{q_s}{\sqrt{q_s^2 + q_z^2}}, 0, 0, \frac{q_z}{\sqrt{q_s^2 + q_z^2}} \right) \\ \mathbf{q}_{swing_{xy}} &= \left(\sqrt{q_s^2 + q_z^2}, 0, \frac{q_s q_x - q_y q_z}{\sqrt{q_s^2 + q_z^2}}, \frac{q_s q_y - q_x q_z}{\sqrt{q_s^2 + q_z^2}} \right) \\ \mathbf{q} &= \mathbf{q}_{swing_{xz}} \mathbf{q}_{twist_y} \end{aligned} \tag{34}$$

where q_x, q_y, and q_z are the rotations around the x-, y-, and z-axis respectively, and q_{xy}, q_{xz}, and q_{yz} are the rotations a vector in the xy-, xz-, and yz-plane respectively (see Appendix A for proof). We can validate the twist-and-swing decomposition by multiplying them together and reconstructing the original quaternion.

This extension of the basic Gauss-Seidel algorithm to handle constraint limits for the unknowns is called the Projected Gauss-Seidel (PGS) algorithm. The angular limits form bounds that are in form of upper and lower joint angles that are easily enforced through clamping. Furthermore, the PGS algorithm has O(n) running time and convergence is guaranteed as long as the matrix is positive definite [8]. In practice, we have found the algorithm to provide promising visual and numerical results.

VIII. Spatial and Temporal Coherency

We give the iterative solver a warm-start approximation at the start of each iteration update by taking advantage of spatial and temporal coherency of the problem. Since the PGS solver is iterative by design and without a warm-start approximation, its convergence rate can be very slow (i.e., depending upon the eigenvalues of the matrix it is solving). However, by caching the result between updates (i.e., use previous solution as the start for the next update), we can considerably reduce the number of iterations, especially for cases when there are only minuscule changes for the system.

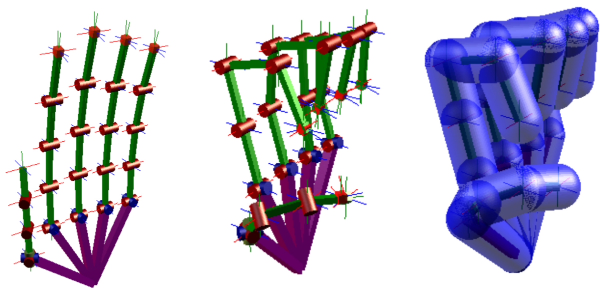

Figure 5. Simulation Screenshots A. Experimental poses for our articulated hand.

IX. Experimental Results

On average, the small spatial coherent transitions between frame updates resulted in the Gauss-Seidel method requiring only two or three iterations for the end-effectors to reach acceptable answers. This resulted in the IK solver being able to easily maintain a low-computational overhead and run at real-time frame-rates. While our Gauss-Seidel implementation was straightforward it was implemented in a single threaded program and did not exploit any parallel architecture speed ups (e.g., using a multi-core CPU or GPU); however, numerous methods were demonstrated by Courtecuisse et al. [44] for exploiting multi-core architectures to achieve much improved performance using the Gauss-Seidel algorithm.

The performance of our iterative Gauss-Seidel IK implementation was computationally fast and ran at real-time frame-rates enabling the IK problem to be modified on the fly. For cases when little or no movement occurred the solver would perform 1 to 2 iterations at most, while for sporadic changes in the articulated posture resulted in approx. 10 or more iterations. Furthermore, our Gauss-Seidel method would only require a few milliseconds to compute the solution. However, the cost of calculating the IK solution can vary greatly depending upon the starting assumption. Our implementation performed at real-time rates and maintained a consistent frame-rate well above a 100Hz. Simulations were performed on a machine with the following specifications: Windows7 64-bit, 16Gb Memory, Intel i7-2600 3.4Ghz CPU. Compiled and tested with Visual Studio 2012.

One important criteria was that the IK solver remained stable, e.g., when the end-effectors are placed out of reach, so that no solution exists. In practice, when no result was obtainable, a best reach condition was always presented, stretching to obtain the end-effectors but remaining stable (i.e., not oscillating or jittering). Furthermore, when end-effectors were started at radically different locations, the resulting solution would haphazardly jerk; however, the result always converged on acceptable poses.

We experimented with a diverse range of poses of generally unpredictable and chaotic finger movement to explore the stability and flexibility of our approach. For example, we did random on the spot compositions of the hand opening, closing, stretching, and so on, and always converged on a solution. It should be pointed out that the hand has less problems with angular limits and singularity conditions compared to a full body articulated structure. However, the hand has enough degrees of freedom and flexibility to perform a suitably detailed set of tests.

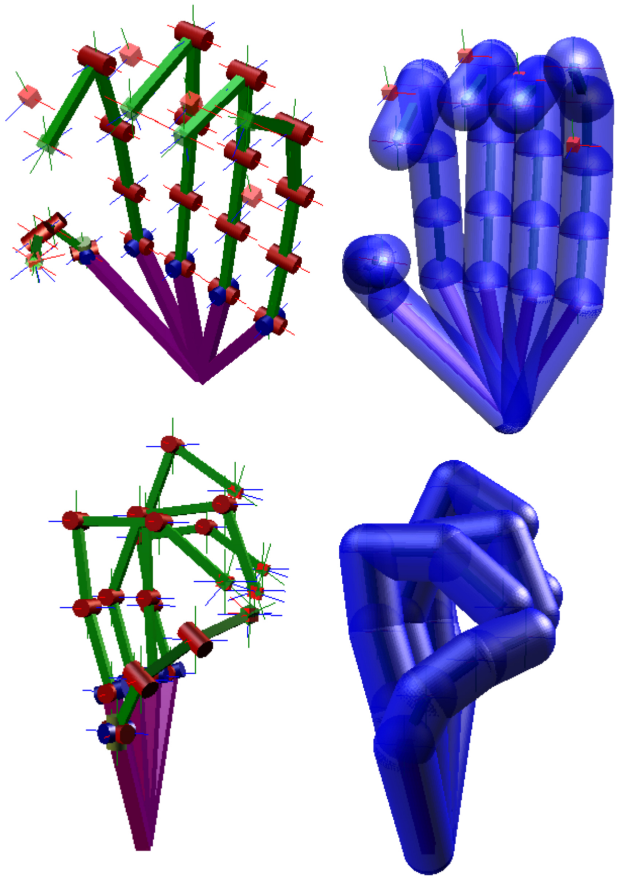

Figure 6. Simulation Screenshots B. Experimental poses for our articulated hand.

X. Limitations

We did not include any inter-link collision detection so it was possible for fingers to pass through one another. During situations when joints were against their angular limits, it would take longer for the iterative IK solver to converge on a solution, since joints that could not move further would be constantly pushed back. Finally, we did not weight or couple any of the angular joints; hence, the final pose could look uncomfortable and unnatural while still being physically-plausible. For example, in a real-world human hand, if the index finger is pulled downwards towards the wrist, it should affect its neighboring fingers.

XI. Conclusion and Discussion

We presented the Gauss-Seidel technique as a method for solving real-time articulated IK problems with quaternion-exponential maps and dual-quaternions. We used temporal caching to reduce the computational cost and gain real-time performance speeds. The results of the IK system performed well enough to be used in time critical systems (such as games). With the angular limits, the method can suffer from singularity problems if the end-effectors jump; however, due to the end-effectors following small spatial transitions singularities are mostly avoided. All in all, the algorithm is simple to implement, computationally fast, little memory overhead, and is fairly robust. The IK solution can work with multiple end-effectors to produce poses with smooth movement with and without constraints.

While we demonstrated the practical aspect of using the Gauss-Seidel method as a valid real-time method for a articulated IK system, further work still needs to be done for a more detailed statistical comparison between the afore-mentioned IK solutions; comparing memory, complexity and computational costs.

A further area of study would be combining the IK solver with a physics-based system (i.e., rigid body constraint solver) and explore object interaction (e.g., picking up a ball or a pencil). Furthermore, to enable greater simulation speeds, the possible investigation and exploration of making the solver more parallel, for example, Poulsen [45] demonstrated a Parallel Projected Gauss-Seidel Method.

This paper exploited the Gauss-Seidel iterative method in conjunction with a set of highly non-linear equations to solve an inverse kinematic problem for an articulated structure. While we demonstrated the practical viability of the Gauss-Seidel method with exponential-maps, we did not implement and compare our approach with the many different numerical techniques(e.g., Newton or Broyden approach) and is an area of further investigation.

Acknowledgment

The authors would like to thank the reviewers for taking time out of their busy schedules to provide valuable, helpful, and insightful feedback that has contributed towards making this better paper more clear, concise, and correct.

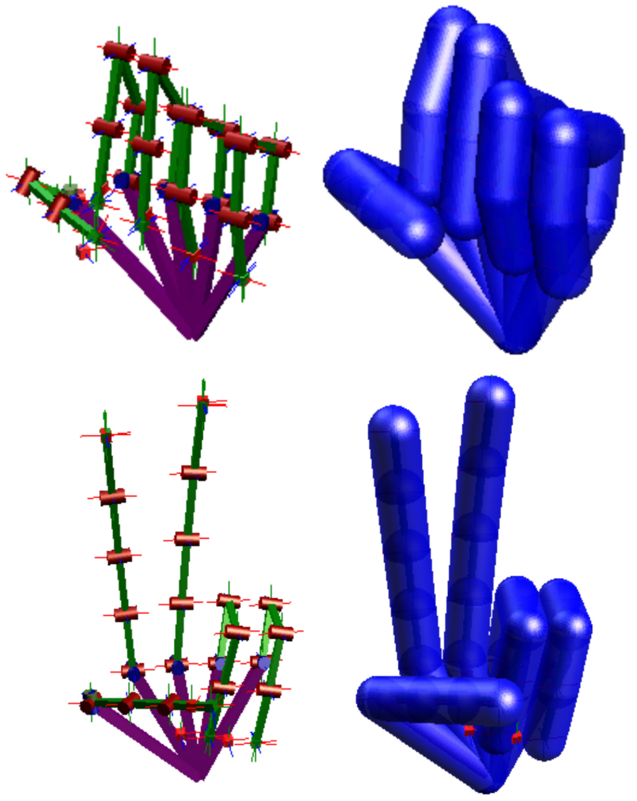

Figure 7. Simulation Screenshots C. Experimental poses for our articulated hand.

References

[1] B. Kenwright, "Real-time character inverse kinematics using the gauss-seidel iterative approximation method," *CONTENT 2012, The Fourth International Conference on Creative Content Technologies*, pp. 63–68, 2012. 1, 2, 3

[2] B. Kenwright, "Synthesizing balancing character motions," *Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS 2012*, pp. 87–96, 2012. 1

[3] C. Welman, *Inverse kinematics and geometric constraints for articulated figure manipulation*. PhD thesis, 1993. 1, 2

[4] L. Unzueta, M. Peinado, R. Boulic, and A. Suescun, "Full-body performance animation with Sequential Inverse Kinematics," *Graphical Models*, vol. 70, pp. 87–104, Sept. 2008. 1, 2

[5] W. Wolovich and H. Elliott, "A computational technique for inverse kinematics," in *The 23rd IEEE Conference on, Decision and Control, 1984*, vol. 23, pp. 1359–1363, IEEE, 1984. 1, 2, 5

[6] R. Kulpa and F. Multon, "Fast inverse kinematics and kinetics solver for human-like figures," *IEEE Humanoid Robots*, vol. December, no. 5, pp. 38–43, 2005. 1, 2

[7] J. Zhao and N. Badler, "Inverse kinematics positioning using nonlinear programming for highly articulated figures," *ACM Transactions on Graphics (TOG)*, vol. 13, no. 4, pp. 313–336, 1994. 1, 2

[8] W. R. Cottle, J.-S. Pang, and E. R. Stone, *The Linear Complementarity Problem*. Academic Press, 1992. 1,

7

[9] B. Kenwright, "A beginners guide to dual-quaternions: What they are, how they work, and how to use them for 3d character hierarchies," pp. 1–13, 2012. WSCG 2012 Communication Proceedings, Conference June. 2012. 1, 2, 3

[10] F. S. Grassia, "Practical parameterization of rotations using the exponential map," *J. Graph. Tools*, vol. 3, pp. 29–48, Mar. 1998. 1, 4, 6, 7

[11] M. Girard and A. Maciejewski, "Computational modeling for the computer animation of legged figures," in *ACM SIGGRAPH Computer Graphics*, vol. 19, pp. 263–270, ACM, 1985. 2

[12] B. Rose, S. Rosenthal, and J. Pella, "The process of motion capture: Dealing with the data," in *Computer Animation and Simulation*, vol. 97, pp. 1–14, Citeseer, 1997. 2

[13] K. Yamane and Y. Nakamura, "Natural motion animation through constraining and deconstraining at will," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, pp. 352–360, July 2003. 2

[14] L. Zhao, "Gesticulation behaviors for virtual humans," *Pacific Graphics '98. Sixth Pacific Conference on Computer Graphics and Applications*, pp. 161–168, 1998. 2

[15] S. R. Buss and J.-S. Kim, "Selectively damped least squares for inverse kinematics," *Journal of Graphics Tools*, vol. 10, pp. 37–49, 2004. 2, 5, 6

[16] R. Fletcher, *Practical methods of optimization, Volume 1*. Wiley, 1987. 2

[17] A. Balestrino, G. D. Maria, and L. Sciavicco, "Robust control of robotic manipulators," *Proc. Of the 9th IFAC World Congress*, vol. 5, pp. 2435–2440, 1984. 2, 5

[18] J. Baillieul, "Kinematic programming alternatives for redundant manipulators," in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, vol. 2, pp. 722–728, IEEE, 1985. 2, 5

[19] C. Wampler, "Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 16, pp. 93–101, Jan. 1986. 2, 5

[20] Y. Nakamura and H. Hanafusa, "Inverse kinematic solutions with singularity robustness for robot manipulator control," *Journal of Dynamic Systems, Measurement, and Control*, vol. 108, no. 3, pp. 163–171, 1986. 2, 5

[21] J. Lander, "Making kine more flexible," *Game Developer Magazine*, no. November, 1998. 2

[22] L. Wang, "A combined optimization method for solving the inverse kinematics problems of mechanical manipulators," *Robotics and Automation, IEEE*, vol. 1, no. 4, 1991. 2

[23] R. Boulic, J. Varona, L. Unzueta, M. Peinado, A. Suescun, and F. Perales, "Evaluation of on-line analytic and numeric inverse kinematics approaches driven by partial vision input," *Virtual Reality*, vol. 10, pp. 48–61, Apr. 2006. 2

[24] N. Courty and E. Arnaud, "Inverse kinematics using sequential monte carlo methods," *Articulated Motion and Deformable Objects*, pp. 1–10, 2008. 2

[25] C. Hecker, B. Raabe, R. W. Enslow, J. DeWeese, J. Maynard, and K. van Prooijen, "Real-time motion retargeting to highly varied user-created morphologies," *ACM Transactions on Graphics*, vol. 27, p. 1, Aug. 2008. 2

[26] K. Grochow, S. Martin, A. Hertzmann, and Z. Popović, "Style-based inverse kinematics," in *ACM Transactions on Graphics (TOG)*, vol. 23, pp. 522–531, ACM, 2004. 2

[27] R. Sumner, M. Zwicker, C. Gotsman, and J. Popović, "Mesh-based inverse kinematics," in *ACM Transactions on Graphics (TOG)*, vol. 24, pp. 488–495, ACM, 2005. 2

[28] K. Der, R. Sumner, and J. Popović, "Inverse kinematics for reduced deformable models," in *ACM Transactions on Graphics (TOG)*, vol. 25, pp. 1174–1179, ACM, 2006. 2

[29] J. Brown, J. Latombe, and K. Montgomery, "Real-time knot-tying simulation," *The Visual Computer*, vol. 20, no. 2, pp. 165–179, 2004. 2

[30] A. Aristidou and J. Lasenby, "FABRIK: A fast, iterative solver for the Inverse Kinematics problem," *Graphical Models*, vol. 73, pp. 243–260, Sept. 2011. 2

[31] R. Mukundan, "A robust inverse kinematics algorithm for animating a joint chain," *International Journal of Computer Applications in Technology*, vol. 34, no. 4, p. 303, 2009. 2

[32] R. Muller-Cajar and R. Mukundan, "Triangualtion-a new algorithm for inverse kinematics," *Proc. of the Image and Vision Computing New Zealand*, pp. 181–186, 2007. 2

[33] W. Tang, M. Cavazza, and D. Mountain, "A constrained inverse kinematics technique for real-time motion capture animation," *The Visual Computer*, vol. 15, pp. 413–425, Nov. 1999. 2

[34] L. Verlet, "Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules," *Physical Review*, vol. 159, no. 1, pp. 98–103, 1967. 2

[35] G. Arechavaleta, E. Lopez-Damian, and J. Morales, "On the use of iterative lcp solvers for dry frictional contacts in grasping," in *Advanced Robotics, 2009. ICAR 2009. International Conference on*, pp. 1–6, June. 2

[36] H.-L. Pham, V. Perdereau, B. V. Adorno, and P. Fraisse, "Position and orientation control of robot manipulators using dual quaternion feedback," in *IROS*, pp. 658–663, 2010. 3

[37] W. R. Hamilton, "Elements of quaternions," *Reprinted by Chelsea Pub, New York in 1969*, 1860. 3

[38] Clifford, "Preliminary sketch of biquaternions," *In Oxford Journals, Proceedings London Mathematical Society, 1-4,*, pp. 381–395, 1873. 4

[39] F. Jourdan, P. Alart, and M. Jean, "A Gauss-Seidel like algorithm to solve frictional contact problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 155, pp. 31–47, Mar. 1998. 6

[40] T. Liu and M. Wang, "Computation of three-dimensional rigid-body dynamics with multiple unilateral contacts using time-stepping and gauss-seidel methods," *Automation Science and Engineering, IEEE Transactions on*, vol. 2, no. 1, pp. 19–31, 2005. 6

[41] E. Catto, "Iterative dynamics with temporal coherence," in *Game Developer Conference*, pp. 1–24, 2005. 6

[42] H. G. Gene and F. V. L. Charles, *Matrix Computations*. The Johns Hopkins University Press, 3rd editio ed., 1996. 6

[43] M. Kallmann, "Analytical inverse kinematics with body posture control," *Computer Animation and Virtual Worlds*, vol. 19, pp. 71–91, May 2008. 7

[44] H. Courtecuisse and J. Allard, "Parallel Dense Gauss-Seidel Algorithm on Many-Core Processors," *2009 11th IEEE International Conference on High Performance Computing and Communications*, no. 1, pp. 139–147, 2009. 8

[45] M. Poulsen, "Parallel projected gauss-seidel method," Master's thesis, University of Copenhagen, 2010. 9

[46] M. Bartelink, "Global inverse kinematics solver for linked mechanisms under joint limits and contacts," Master's thesis, Universiteit Utrecht (European Design Centre), 2012. 12

Appendix A. Proof of Twist-and-Swing Decomposition Equation

We show through quaternion algebra the mathematical proof for Equation 32, which can similarly be applied to Equation (33) and Equation (34), and how a 3D unit-quaternion can be decomposed into two parts: the *twist* and *swing* components. We start with a unit-quaternion rotation shown in Equation 35.

$$\mathbf{q} = (q_s, q_x, q_y, q_z) \tag{35}$$

where is the vector component, and is the scalar component. We can calculate a quaternion from an axis-angle using Equation 36.

$$\begin{aligned} q_s &= c_{xyz} = \cos\left(\frac{\theta}{2}\right) \\ q_x &= s_x = v_x \sin\left(\frac{\theta}{2}\right) \\ q_y &= s_y = v_y \sin\left(\frac{\theta}{2}\right) \\ q_z &= s_z = v_z \sin\left(\frac{\theta}{2}\right) \end{aligned} \tag{36}$$

where v is a unit-vector representing the axis of rotation, and is the angle of rotation. Hence, we can say since the twist is only around the x-axis we can deduce that the yz-axis components will be zero and give us Equation 37.

$$\mathbf{q}_{twist} = \mathbf{q}_x = (c_x, s_x, 0, 0) \tag{37}$$

Furthermore, we can also deduce that the swing x-axis component will be zero in the resulting quaternion as shown in Equation 38.

$$\mathbf{q}_{swing} = \mathbf{q}_{yz} = (c_{yz}, 0, s_y, s_z) \tag{38}$$

where c and s represent the scalar cos and sin component of the half angles (i.e., see Equation 36). A unit-quaternion must obey Equation 39.

$$q_s^2 + q_x^2 + q_y^2 + q_z^2 = 1 \tag{39}$$

Hence, from Equation 37 and Equation 38 we can derive Equation 40.

$$\begin{aligned} c_x^2 + s_x^2 &= 1 \\ c_{yz}^2 + s_y^2 + s_z^2 &= 1 \end{aligned} \tag{40}$$

Subsequently, if we multiply the individual twist and swing quaternions together we can reconstruct the original quaternion as shown in Equation 41.

$$\begin{aligned} \mathbf{q}_{xyz} &= \mathbf{q}_{yz}\mathbf{q}_x \\ &= (c_{yz}, 0, s_y, s_z)(c_x, s_x, 0, 0) \\ &= ((c_x c_{yz}), (s_x c_{yz}), (c_x s_y + s_x s_z), (c_x s_z - s_x s_y)) \end{aligned} \tag{41}$$

Hence, from Equation 37 and knowing the vector sum of the two non-zero components from Equation 41 sums up to one, we can derive q_{twist}, as shown in Equation 42.

$$\begin{aligned} \mathbf{q}_{twist} &= q_s^2 + q_x^2 \\ &= (c_x c_{yz})^2 + (s_x c_{yz})^2 \\ &= c_{yz}^2(c_x^2 + s_x^2) \qquad (knowing, \quad \cos^2 + \sin^2 = 1) \\ &= c_{yz}^2 \end{aligned} \tag{42}$$

Therefore, we have Equation 43.

$$c_{yz} = \sqrt{q_s^2 + q_x^2} \tag{43}$$

We can multiply Equation 41 by the inverse of Equation 43 to remove the quaternion swing component and leave the quaternion twist part (shown in Equation 44).

$$\begin{aligned}\mathbf{q}_{twist} &= \mathbf{q}_x \\ &= (c_x, s_x, 0, 0) \\ &= ((c_x c_{yz}), (s_x c_{yz}), 0, 0)\frac{1}{c_{yz}} \\ &= (q_s, q_x, 0, 0)\frac{1}{\sqrt{q_s^2 + q_x^2}}\end{aligned} \tag{44}$$

We extract the swing component by multiply the quaternion by the inverted (conjugated) twist quaternion (shown in Equation 45).

$$\begin{aligned}\mathbf{q}_{swing} &= \mathbf{q}_{xyz}\mathbf{q}^*_{twist} \\ &= (q_s, q_x, q_y, q_z)(q_s, -q_x, 0, 0)\frac{1}{\sqrt{q_s^2 + q_x^2}} \\ &= (c_{yz}, 0, s_y, s_z) \\ &= ((q_s^2 + q_x^2), 0, (q_s q_y - q_x q_z), (q_s q_z + q_x q_y))\frac{1}{\sqrt{q_s^2 + q_x^2}}\end{aligned} \tag{45}$$

Whereby, Equation 44 and Equation 45 sums-up our algebraic proof. Similarly the twist-and-swing can be proved for the y-, and z-axis (as shown in Equation (33) and Equation (34)).

Appendix B. Calculating the Jacobian

The joint-space and Cartesian space are the two state space variables for the kinematic system (shown in Figure 1). The Jacobian matrix relates the changes in joint angles θ with change in position or orientation $\mathbf{X}$ of some point on the connected hierarchy of links (i.e., Cartesian space) and is defined as:

$$J = \frac{\partial \mathbf{X}}{\partial \theta} \tag{46}$$

A. Finite-Difference Method

The finite-difference method is an approximation technique given by Equation (47). The error is proportional to the δ step and is limited by numerical accuracy (e.g., a 32-bit floating point) and the computational speed of the system. The method is idea for situations where it is difficult or impossible to find an analytical solution for the kinematic system due to its complexity and size. However, the method is a good solution for estimating an approximate Jacobian solution.

$$\frac{dy}{dx} \approx \frac{y(x+\delta) - y(x)}{\delta} \tag{47}$$

The finite-difference implementation for calculating an approximate Jacobian is given in Algorithm 2.

Algorithm 2 Finite-Different Method for Approximating the Jacobian

```
p = f(q)
for $n = 0$ to $n - 1$ do
    $p_\delta = f(q + e_n\delta)$
    $\mathbf{J}_n = \frac{p_\delta - p}{\delta}$
end for
return J
```

B. Analytical Formulation of the Jacobian using Dual-Quaternions and Quaternion Exponentials

As step-by-step derivation of the Jacobian Dual-quaternion to Quaternion Exponential Mapping from the work by Bartelink [46] is presented below in Figure 8.

(*) Each dual-quaternion pos&rot transform is defined as (i.e., in respect of the exponential)

$$\underline{\hat{q}}_i^R(\vec{w}) = (1+\frac{1}{2}\epsilon\vec{t}_i^R)\;\hat{q}_i^R(\vec{w})$$

{superscript R and W indicate relative and world coordinate}
q_i - quaternion orientation
t_i - quaternion translation (i.e., $[0,t_x,t_y,t_z]$)
(subscript i, joint number)

$$\frac{\partial\underline{\hat{q}}_i^R(\vec{w})}{\partial w_k} = \frac{\partial}{\partial w_k}\left((1+\frac{1}{2}\epsilon\vec{t}_i^R)\;\hat{q}_i^R(\vec{w})\right)$$

(*) Differentiate it so we have the rate of change wrt. axis-angle

$$\frac{\partial\underline{\hat{q}}_i^R(\vec{w})}{\partial w_k} = (1+\frac{1}{2}\epsilon\vec{t}_i^R)\frac{\partial\hat{q}_i^R(\vec{w})}{\partial w_k} \checkmark$$

{solve using dual-quaternion product rule}

{we know how to calculate the derivative of the quaternion-exponential}

{because...
$$\underline{\hat{q}}_{end}^W(\vec{w}) = \underline{\hat{q}}_0^R\underline{\hat{q}}_1^R\cdots\underline{\hat{q}}_{i-1}^R\underline{\hat{q}}_i^R\underline{\hat{q}}_{i+1}^R\underline{\hat{q}}_{end}^R$$
$$= \underline{\hat{q}}_{i-1}^W\left(\underline{\hat{q}}_i^R(\vec{w})\right)\underline{\hat{q}}_m^i\underline{\hat{q}}_{end}^R$$
}

(*) See how the 'independent' dual-quaternions are connected and how each 'relative' transform contributes to the overall dual-quaternion world transform

$$\frac{\partial\underline{\hat{q}}_{end}^W(\vec{w})}{\partial w_k} = \underline{\hat{q}}_{i-1}^W\frac{\partial\underline{\hat{q}}_i^R(\vec{w})}{\partial\vec{w}}\underline{\hat{q}}_m^i\underline{\hat{q}}_{end}^R$$

{because, we want the derivative of
$$\vec{e}(\vec{w}) = \vec{e}$$
$$\vec{e} = (\vec{t}_{end}^W, \vec{o}_{end}^W)$$
$$\vec{o}_{end}^W = log\left(q_{end}^W(\vec{w})\right)$$

(*) We are interested in the rate of change of the axis-angles with the end-effector - so we 'expand' out what the dual-quaternion world transform is so we can extract the solution

$$\underline{\hat{q}}_{end}^W = (1+\frac{1}{2}\epsilon\vec{t}_{end}^W(\vec{w}))\;\hat{q}_{end}^W(\vec{w})$$

(*) We have:

$$\frac{\partial\vec{t}_{end}^W(\vec{w})}{\partial w_k}\quad\frac{\partial\underline{\hat{q}}_{end}^W(\vec{w})}{\partial\vec{w}_k}$$

which we need for:

$$\vec{e} = (\vec{t}_{end}^W, \vec{o}_{end}^W)$$

$$\frac{\partial\underline{\hat{q}}_{end}^W(\vec{w})}{\partial w_k} = \frac{\partial}{\partial w_k}\left((1+\frac{1}{2}\epsilon\vec{t}_{end}^W(\vec{w}))\;\hat{q}_{end}^W(\vec{w})\right)$$
$$= \frac{\partial\hat{q}_{end}^W(\vec{w})}{\partial w_k} + \frac{1}{2}\epsilon\left(\frac{\partial\vec{t}_{end}^W(\vec{w})}{\partial w_k}\hat{q}_{end}^W(\vec{w}) + \vec{t}_{end}^W(\vec{w})\frac{\partial\hat{q}_{end}^W(\vec{w})}{\partial w_k}\right)$$

{dual-quaternion product rule}

$$\frac{\partial\vec{t}_{end}^W(\vec{w})}{\partial w_k} = \left(2\mathbb{D}\left(\frac{\partial\underline{\hat{q}}_{end}^W(\vec{w})}{\partial w_k}\right)\right)\left(\mathbb{R}\left(\frac{\partial\underline{\hat{q}}_{end}^W(\vec{w})}{\partial w_k}\right)\right)^{-1}$$
$$\frac{\partial\hat{q}_{end}^W(\vec{w})}{\partial w_k} = \mathbb{R}\left(\frac{\partial\underline{\hat{q}}_{end}^W(\vec{w})}{\partial w_k}\right)$$

{convert dual-quaternion to individual components, i.e., translation, and orientation: }
$$\varsigma = q_r + q_d$$
$$r = q_r$$
$$t = 2q_d\,q_r^*$$

(*) We obtain them through substitution

$$\frac{\partial\vec{t}_{end}^W(\vec{w})}{\partial w_k} = \left(2\mathbb{D}\left(\frac{\partial q_{end}^W(\vec{w})}{\partial w_k}\right) - \vec{t}_{end}^W(\vec{w})\mathbb{R}\left(\frac{\partial q_{end}^W(\vec{w})}{\partial w_k}\right)\right)\left(q_{end}^W(\vec{w})\right)^{-1}$$

$\mathbb{R}(\cdot)$ gives the real-part of the dual-quaternion
$\mathbb{D}(\cdot)$ gives the dual-part of the dual-quaternion

$$\frac{\partial q_{end}^W(\vec{w})}{\partial w_k} = \mathbb{R}\left(\frac{\partial\underline{\hat{q}}_{end}^W(\vec{w})}{\partial w_k}\right)$$

$$\vec{o}_{end}^W(\vec{w}) = \left(o_1(\vec{w}), o_2(\vec{w}), o_3(\vec{w})\right) = \log(q_{end}^W(\vec{w}))$$

$$\frac{\partial q_{end}^W(\vec{w})}{\partial w_k} = \frac{\partial q_{end}^W(\vec{w})}{\partial\vec{o}_l(\vec{w})}\frac{\partial\vec{o}_l(\vec{w})}{\partial w_k}$$

{quaternion chain rule}

$$\frac{\partial\vec{o}_l(\vec{w})}{\partial w_k} = \mathbb{S}\left(\left(\frac{\partial q_{end}^W(\vec{w})}{\partial\vec{o}_l}\right)^*\left(\frac{\partial q_{end}^W(\vec{w})}{\partial w_k}\right)\right)$$

$\mathbb{S}(\cdot)$ gives the scalar-part of the quaternion

(*) Desired Jacobian components

$$\frac{\partial\vec{e}(\vec{w})}{\partial w_k} = \left(\frac{\partial\vec{t}_{end}^W(\vec{w})}{\partial w_k}\frac{\partial\vec{o}_{end}^W(\vec{w})}{\partial w_k}\right) \checkmark$$

Figure 8. Step-by-Step Formulation of Dual-Quaternion and Quaternion Exponential Jacobian.