Inverse Kinematics Algorithm in Dual Quaternion Form Based on FABRIK (Forward and Backward Reach Inverse Kinematics) Algorithm

Muhammad Hafidz Bin Azmi[1, a)] and Sergey Akhramovich[2, b)]

[1]Department of System Analysis and Control 604, Moscow Aviation Institute, Moscow Russia

[a)]hafidzazmi07@icloud.com
[b)]akhramovichsa@gmail.com

Abstract. This article examines a formulation method to solve the Inverse Kinematics problem of serial robot manipulators in Dual Quaternion form. The formulation is based on Forward and Backward Reaching Inverse Kinematics (FABRIK) algorithm. Inverse Kinematics, consists of searching possible and feasible joints motions to reach a desired position and orientation as smoothly, rapidly and as accurately as possible. One of the current solutions used in computer graphics is the FABRIK algorithm. FABRIK algorithm, reposition each joint one at a time in a forward and backward iterative mode. However, the algorithm alone is not suitable for robot manipulators as it does not consider the joint direction of rotation and orientation. In addition, this article described an Inverse Kinematics algorithm in Dual Quaternion form on the basis of an existing FABRIK algorithm and compared with other heuristic methods. The proposed algorithm uses the advantages of Dual Quaternion that fully defined the orientation and position in a single form with 8 components. Besides, the proposed algorithm does not use any trigonometry function compared to other traditional methods. Finally, in this article, different models were tested with the algorithm ranging from 2 DOF (Degree of freedom) to 6 DOF and the results are presented here.

INTRODUCTION

Since the beginning of industrial revolution 4.0 in 2011, the emerging of industrial robots in different fields has significantly increased in trend. Industrial robots have been used ranging from the aerospace industry in Curiosity Rover, to the food industry in robot barista manufactured by CAFE X. This increasing trend makes it difficult to fulfill the needs of the diverse industries as the industrial robot model and the movement required for each industry are different. The control system for an industrial robot consists of a few components and one of the fundamentals is the Inverse Kinematics.

Inverse Kinematics, consists of finding a possible and feasible joints motions solution for which the end effectors move to the desired position as smoothly, rapidly and as accurately as possible. During the last decades, several methods and techniques have been presented to solve the Inverse Kinematics problem in Dual Quaternion form, such as Inverse Jacobian, Cyclic coordinate descent (CCD) and Geometrical approach (Paden-Kahan subproblems) [1][2][3]. However, these methods involve high computational costs, are mathematically complicated and result in unrealistic movement.

One of the current solutions to solve Inverse Kinematics problem in computer graphics is the Forward and Backward Reach Inverse Kinematics (FABRIK) algorithm [4][5]. FABRIK algorithm, focuses on finding the joint locations as a problem of finding a point on a line. The algorithm does not consider the direction of rotation and joint orientation. Thus, the algorithm alone is not suitable for an industrial robot as robots use motor to control the joints. To rotate the joints, one needs to know how much rotation is needed for each joint.

This article proposes to design a new Inverse Kinematics algorithm in Dual Quaternion form on the basis of an existing FABRIK algorithm. This algorithm uses the advantages of Dual Quaternion that fully defined the orientation and position in a single form. Moreover, Dual Quaternion form requires less calculation in chain transformation [6].

This paper will be constructed as follow. Firstly, to design the Inverse Kinematics algorithm in the Dual Quaternion form. Next, to test different models with the algorithm ranging from 2 Degree of freedom (DOF) to 6 DOF. Finally, to make it work in real-life situations with realistic movements. Constraints in the Dual Quaternion form will be applied in order for this to be applicable.

SERIAL MANIPULATOR

Serial manipulator is the most common industrial robots, that consist of a chain of links connected by motor-actuated joints that extend from the base to an end effector as in FIGURE 3. There are two common type of joints used in manipulator, revolute joint and prismatic joint. This article focuses on revolute joint where the joint is rotated in a single axis rotation.

MATHEMATICAL BACKGROUND

This section provides a brief introduction to Quaternion, Dual Quaternion, and the use of Dual Quaternion as rigid body transformation. Some basic understanding of this are required to understand the proposed algorithm for solving the inverse kinematics of robot manipulators. For more details about Quaternion and Dual Quaternion refer to [2][7]. The notation used for Quaternions and Dual Quaternion are explained in the next section. .

Definitions

To reduce ambiguity and make the article as readable as possible, the variable symbols are defined as follow

$\vec{u}$ - vector	$\hat{q}$ - Dual Quaternion
q - Quaternion	$\hat{Q}_b^n$ - Joint n relative to the reference frame b

In this article the letter represents either Quaternion or Quaternion component, and symbol "$\wedge$" represent Dual Quaternion or Dual Quaternion component.

Quaternion

Quaternion is a hyper-complex number of rank 4, constituting of four-dimensional vector space over the field of real numbers [7]. A Quaternion can be represented as

$$q = \left[q_s, q_v \right] \tag{1}$$

Where $q_s = q_0$ is the scalar component and $q_v = \left[q_1, q_2, q_3 \right]$ is the vector component. A Quaternion can be illustrated as a single rotation around an axis in a 3-dimensional space [6]. Rotation about a unit axis $d = \left[\, d_x, d_y, d_z \right]^T$ with an angle θ can be expressed as

$$q = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ d * \sin\left(\frac{\theta}{2}\right) \end{bmatrix} \tag{2}$$

Some basic operation of two Quaternions q and p, such as addition, subtraction and multiplication can be expressed as follow

$$q \pm p = \begin{bmatrix} q_s \pm p_s \\ q_v \pm p_v \end{bmatrix}$$
$$q \otimes p = \begin{bmatrix} q_s p_s - {q_v}^T p_v \\ q_s p_v + p_s q_v + q_v \times p_v \end{bmatrix} \quad (3)$$

Where "$\otimes$" and "$\times$" denotes Quaternion product and cross product respectively. Lastly, the inverse of Quaternion [9] or Quaternion conjugate is defined as

$$q^* = [q_s, -q_v] \quad (4)$$

Quaternion with $q_v = 0$, is known as a real Quaternion, and a Quaternion with $q_s = 0$, is called as a pure Quaternion (or vector Quaternion) [10]. Quaternion are non-commutative and a unit Quaternion has a unit length of ${q_0}^2 + {q_1}^2 + {q_2}^2 + {q_3}^2 = 1$.

One of the advantages of using Quaternion for chain rotation is the number of operations required is less compared to the traditional rotation matrix [7]. Moreover, Quaternion can be used to represent a rotation that resulted from two vectors, vector u to vector v as

$$q = \left[\vec{u} \bullet \vec{v}, \vec{u} \times \vec{v} \right] \quad (5)$$

Dual Quaternion

Dual Quaternion is a combination of dual-number theory introduced by Clifford with Quaternion components [9]. Dual Quaternion has the ability to represent 3 dimension Euclidean coordinate space (i.e. rotation and translation) in a single form with 8 components as

$$\hat{q} = [q_r, q_d] = q_r + \varepsilon q_d \quad (6)$$

Where q_r and q_d represent dual scalar and dual vector respectively and both are Quaternions, ε is the dual-factor. Common operation of two Dual Quaternion such as, addition, subtraction and multiplication is defined as follow

$$\hat{q} \pm \hat{p} = \begin{bmatrix} q_r \pm p_r \\ q_d \pm p_d \end{bmatrix}$$
$$\hat{q} \odot \hat{p} = \begin{bmatrix} q_r \otimes p_r \\ q_r \otimes p_d + q_d \otimes p_r \end{bmatrix} = q_r \otimes p_r + \varepsilon \left(q_r \otimes p_d + q_d \otimes p_r \right) \quad (7)$$

Where "$\otimes$" and "$\odot$" denote Quaternion and Dual Quaternion product respectively. Finally, the conjugate of a Dual Quaternion is represented as

$$\hat{q}^* = \left[{q_r}^*, {q_d}^* \right] = {q_r}^* + \varepsilon \left(q_d \right)^* \quad (8)$$

By equating the dual part to zero, Dual Quaternion can represent a pure rotation similar to a Quaternion [7] as

$$\hat{q} = [q_r, 0] = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 & 0 & 0 & 0 & 0 \end{bmatrix}^T \quad (9)$$

A Dual Quaternion can represent a pure translation with no rotation as

$$\hat{q} = [1\ , q_d] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \frac{t_x}{2} & \frac{t_y}{2} & \frac{t_z}{2} \end{bmatrix}^T \tag{10}$$

Dual Quaternion Rigid Transformation

Unit Dual Quaternion can be used to represent a rigid transformation including translation and rotation. There are two common forms used to represent the transformation between two frames [3]. Firstly, Dual Quaternion that is obtained from a rotation of q then a translation of t as

$$\hat{q} = q + \varepsilon\left(\frac{1}{2}qt\right) = \hat{q}_{rot} \odot \left(\hat{q}_{trans}\right) \tag{11}$$

Secondly, Dual Quaternion which is the result of a translation of t and then a rotation of q as

$$\hat{q} = q + \varepsilon\left(\frac{1}{2}tq\right) = \left(\hat{q}_{trans}\right) \odot \hat{q}_{rot} \tag{12}$$

Where, $\hat{q}_{rot} = [q, 0]$ Dual Quaternion pure rotation, $\hat{q}_{trans} = \left[1, \frac{1}{2}t\right]$ Dual Quaternion pure translation, q is the unit Quaternion that describe the rotation and $t = \left[0, \vec{t}\right]$ is the Quaternion that describe the translation represented by the vector $\vec{t}$. In this article, rigid transformation will be represented by equation (12).

MANIPULATOR KINEMATICS

Forward Kinematic

Forward kinematics is the process of calculating the orientation and position of the manipulator joint relative to the reference frame [8]. The forward kinematics equations for the n series chain of a manipulator robot can be formed in a Dual Quaternion form as

$$\hat{Q}_0^n = \hat{q}_0 \hat{q}_1 \hat{q}_2 \cdots \hat{q}_n \tag{13}$$

Where $\hat{q}_1 \ldots \hat{q}_{n-1}$ defines each revolute joint rotation and translation in the joint frame. $\hat{q}_0$ is the joint reference frame and $\hat{q}_n$ is the joint end effector, where n it the total number of joints. Each joint rotation and translation in the joint frame can be represented using equation (12) as

$$\hat{q}_i = q_i + \varepsilon \frac{1}{2} t_i q_i \tag{14}$$

Where the Quaternion that describes the rotation is represented as

$$q_i = \begin{bmatrix} \cos\left(\frac{\theta_i}{2}\right) & d_i * \sin\left(\frac{\theta_i}{2}\right) \end{bmatrix}^T \tag{15}$$

d_i is the direction of rotation, using right-handed coordinate system. Meanwhile the Quaternion that describe the translation can be interpreted as follow

$$q_{d_i} = \frac{1}{2} t_i q_i$$
$$t_i = 2 q_{d_i} q_i^* \tag{16}$$

Where t_i is a vector Quaternion defining the distance between joint i and $i-1$ for $i = 1, \ldots n$.

Inverse Kinematics

Inverse Kinematics is the opposite of Forward Kinematics. Inverse Kinematics consists of searching for the geometry parameters necessary to reach a given position and orientation [9]. There are a few methods and techniques that have been presented to solve the Inverse kinematics problem in a pure Dual Quaternion. The most common approach is the Paden-Kahan subproblems and Cyclic coordinate descent (CCD). However, these methods involve high computational costs, are mathematically complicated and result in unrealistic movement. This article aims to design an alternative solution for Inverse Kinematics problems in Dual Quaternion form based on Forward and Backward Reaching Inverse Kinematics (FABRIK) algorithm.

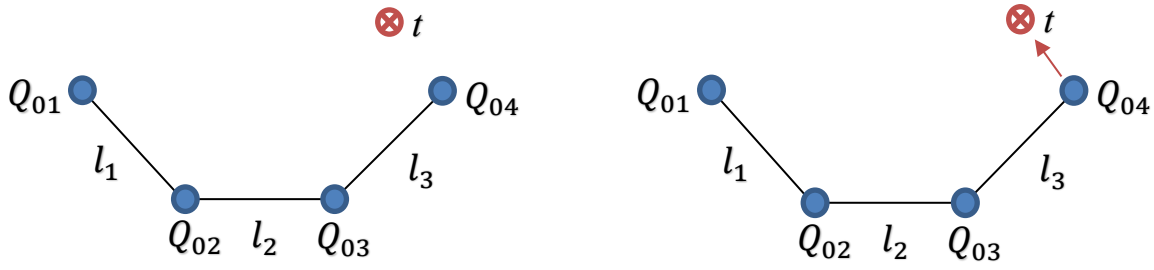
FABRIK

Forward and Backward Reaching Inverse Kinematics (FABRIK) algorithm is a heuristic method design by Andreas Aristidou and Joan Lasenby. Although, FABRIK algorithm is a relatively new solution, it has gain traction notably in the computer graphics industry. This algorithm has a low computational cost and produces visually realistic poses [4]. FABRIK algorithm finds the joints location by searching for a point on a line and minimizing the system error by adjusting each joint position one at a time in a forward and backward iterative mode. The algorithm can be split into two main stages, Forward Reaching and Backward Reaching.

Forward Reaching stage relocate joint position starting from the end effector and finish at the first joint. During the beginning of the first stage, the end effector is repositioned on the target as in FIGURE 1 (b). Then, the joints are repositioned starting from the joint $n-1$ until the first joint. As an example, to find the new position of the joint $n-1$, the joint should lie on the line that passes through the new joint n and joint $n-1$, with a distance of l as shown in FIGURE 1 (c). This process is repeated until the first joint.

Meanwhile, the second stage Backward Reaching reposition the joint starting from the first joint to the end effector. First of all the first joint is repositioned back to its original position as the first joint position is fixed to the base. Next, the joints are repositioned starting from the second joint to the n joint, the end effector. The second stage ends when all joints position are updated. At this stage, the end effector should be closer to the target. These two stages are repeated until the end effector is as close as possible to the target or on the target.

FABRIK algorithm avoids the uses of rotational angles or matrices and uses only the joint position information. As a result, the joint direction of rotation and orientation were not considered in the algorithm. Thus, this leads to some minor drawbacks. Firstly, the algorithm is difficult to be implemented in a system with multiple directions of rotation. Secondly, it is not that easy to apply a joint rotational limit as it required an additional task to derive the joint orientation from their position. Lastly, there is no general solution to apply orientation control with the position information.



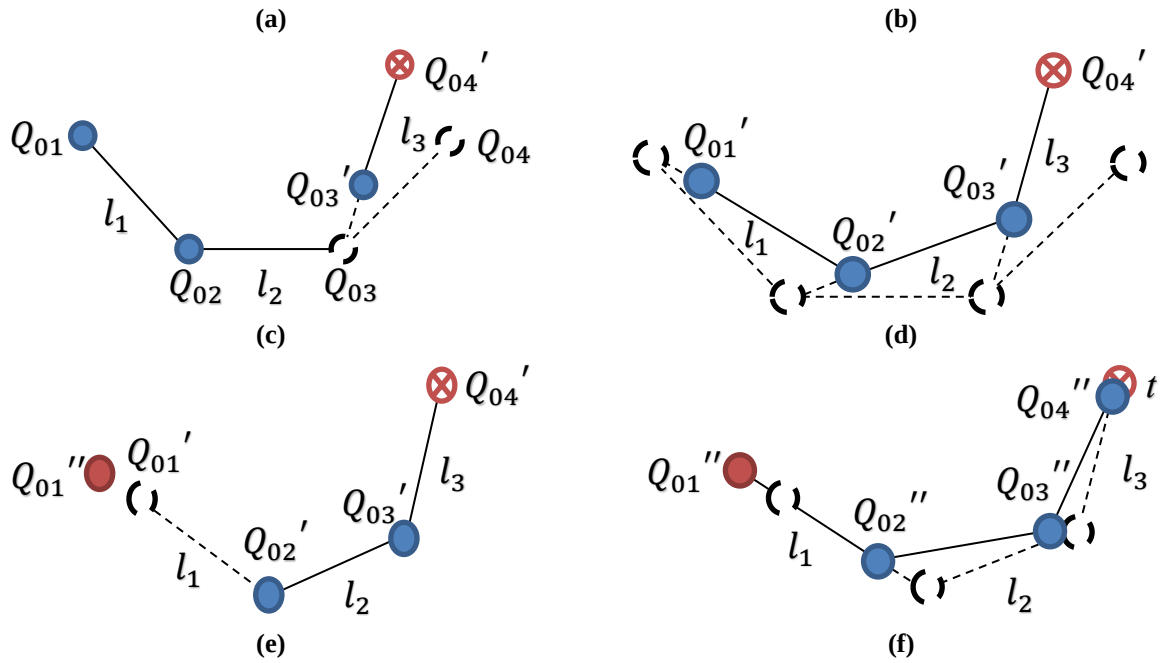


FIGURE 1. Example of FABRIK full iteration on SCARA manipulator. (a) The initial position of the manipulator and target, (b) begin the Forward reaching by reposition the end effector on target, (c) find new joint k position on the line, (d) complete iteration of Forward reaching, (e) begin the Backward reaching by position the first joint back to the based, (d) full iteration on backward reaching

DUAL QUATERNION FABRIK

This section explains the Inverse Kinematics algorithm for position control in the Dual Quaternion form. Dual Quaternion was chosen as it required less number of operations compared to the traditional transformation matrix. The new algorithm aims to provide a more general solution for manipulator robot inverse kinematics problem in Dual Quaternion form. In addition, preserve the advantages of the FABRIK algorithm.

The proposed algorithm is divided into three main sections, pre iteration, main iteration and post iteration. There are a few assumptions made in this algorithm. The first joint position and the distances between each joint is fixed.

Pre iterations

If the target position t and all joints initial position and orientation is given as in equation (10). Each joint rotation and translation relative to the joint frame (i.e. joint configuration) can be derived as the difference between joint j and $j-1$

$$\hat{q}_j = \left(\hat{Q}_0^{j-1}\right)^* \hat{Q}_0^j \tag{17}$$

for $j = 1,\ldots n$. Where $\hat{Q}_0^0 = \hat{q}_0$ is the fixed reference frame. Next, find the first line vector u_j . The first line vector passes through joint j and $j-1$ as shown in FIGURE 2 (a). Line vector u_j can be derived from the Dual Quaternion translation part of Equation (17) as

$$u_j = \frac{2q_{d_j}\left(q_{r_j}\right)^*}{\left\|2q_{d_j}\left(q_{r_j}\right)^*\right\|} \tag{18}$$

In case the robot manipulator has more than one direction of rotation, the line vector u_j should be mapped relative to their plane as

$$u_j = \begin{bmatrix} 0 & u_{v_j} - \left(u_{v_j}{}^T d\right)d \end{bmatrix}^T \tag{19}$$

Where d is the joint direction of rotation.

Main Iteration

The main iteration is divided into two stages Forward reaching and Backward reaching, as in traditional FABRIK.

Forward Reaching (Stage 1)

At the outset, based on the assumption that was made, the first joint is set as the base ***b***

$$\hat{b} = \hat{Q}_0^1 \tag{20}$$

The end effector, joint n is repositioned to the target ***t***.

$$\hat{Q}_0^n = \hat{Q}_0^t \tag{21}$$

Then, to find the new position and orientation of joint k, for $k = n-1,\ldots 1$, second line vector v_k is required. The line vector v_k passes through joint k and new joint $k+1$ as shown in FIGURE 2 (a). The line vector can be found as

$$\hat{r}_k = \left(\hat{Q}_0^k\right)^* \left(\hat{Q}_0^{k+1}\right)'$$
$$v_k = \frac{2r_{d_k} r_{r_k}{}^*}{\left\|2r_{d_k} r_{r_k}{}^*\right\|} \tag{22}$$

Should be noted that the line vector v_k are in vector Quaternion form. In case there are more than one direction of rotation, the line vector v_k must be focused relative to their plane as

$$v_k = \begin{bmatrix} 0 & v_{v_k} - \left(v_{v_k}{}^T d\right)d \end{bmatrix}^T \tag{23}$$

Dual Quaternion pure rotation $\hat{\lambda}_k$, that reorient the joint k to the direction of joint $k+1$ can be formulated using equation (5) with the line vectors v_k and u_{k+1} as

$$\hat{\lambda}_k = \begin{bmatrix} \begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}^T - u_{k+1}v_k, 0 \end{bmatrix} \tag{24}$$

During the first stage, all joints were assumed rotatable including the end effector. Based on that assumption, the new joint $k+1$ can be defined as

$$\left(\hat{Q}_0^{k+1}\right)' = \left(\hat{Q}_0^{k}\right)' \left(\hat{q}_{k+1}\right)\hat{\beta}_k \tag{25}$$

Where $\hat{\beta}_k$ is the rotation on the joint $k+1$ as shown in FIGURE 2 (b) and is defined as

$$\hat{\beta}_k = \left(\widehat{q_{rot\,k+1}}\right)^* \left(\lambda_k\right)^* \left(\widehat{Q_{rot\,0}}^{k}\right)^* \left(\widehat{Q_{rot\,0}}^{k+1}\right)' \tag{26}$$

From equation (25), new joint k can be represented as

$$\left(\hat{Q}_0^{k}\right)' = \left(\hat{Q}_0^{k+1}\right)' \left(\hat{\beta}_k\right)^* \left(\hat{q}_{k+1}\right)^* \tag{27}$$

This process is repeated for all joints. The first stage finished when the first joint position and orientation are updated as in FIGURE 2 (d).

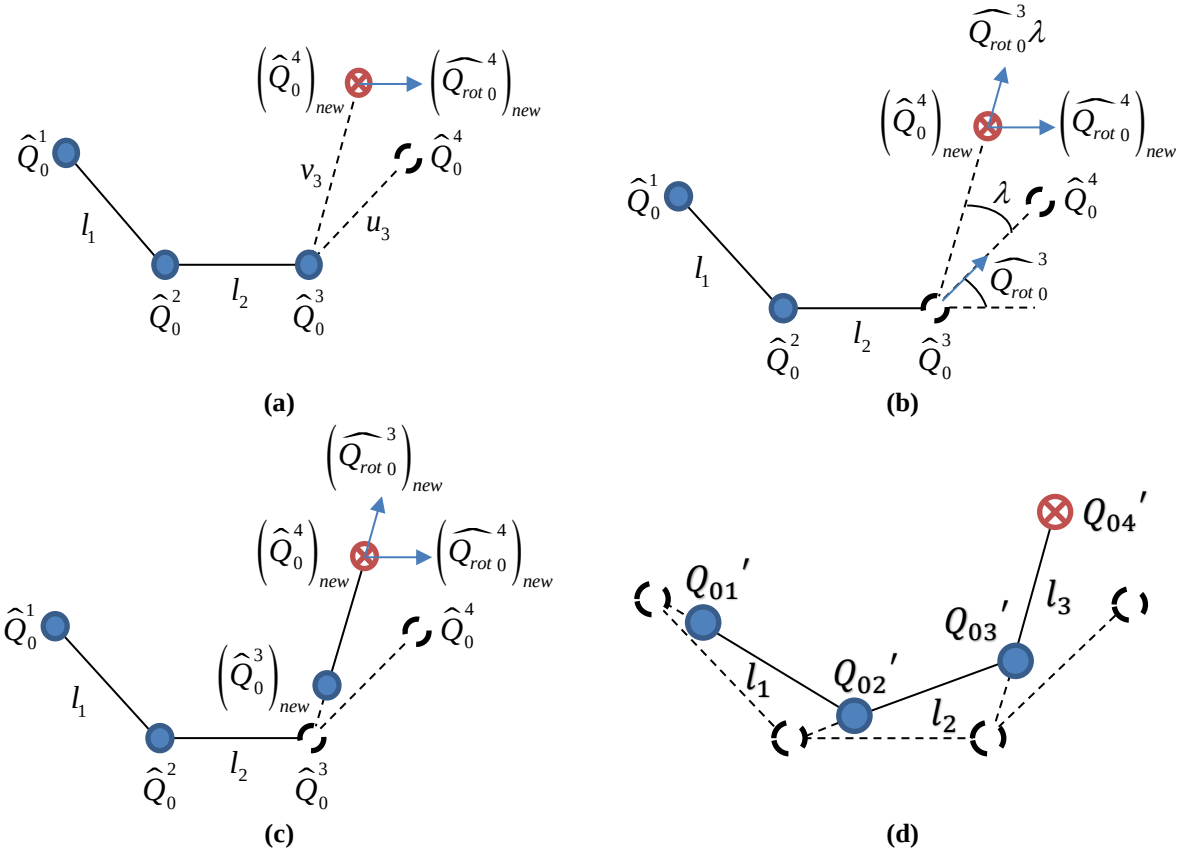


FIGURE 2. Example of Stage 1 Dual Quaternion FABRIK. (a) reposition the new end effector on the target, (b) find the rotation difference and the Dual Quaternion pure rotation to reorient joint k, (c) Reposition joint k on the line (d) Complete iteration of the first stage

Backward Reaching (Stage 2)

First of all, during the second stage the first joint is repositioned back to it initial position as

$$\left(\hat{Q}_0^1\right)' = b \tag{28}$$

Then to find the new position and orientation of joint $k+1$ for $k = 1,\ldots n-1$, the process in stage 1 is repeated from equation (22) to (24). From the Dual Quaternion pure rotation $\hat{\lambda}_k$ in equation (24), the rotation on joint configuration k and line vector u_k can be updated as follow

$$\begin{aligned}\left(\hat{q}_k\right)' &= \hat{q}_k \hat{\lambda}_k \\ u_k &= \frac{2q_{d_k}\left(q_k\right)^*}{\left\|2q_{d_k}\left(q_k\right)^*\right\|}\end{aligned} \tag{29}$$

Next, the orientation of joint k can be updated as

$$\left(\hat{Q}_0^k\right)' = \left(\hat{Q}_0^k\right)' \hat{\lambda}_k \tag{30}$$

Finally, the joint $k+1$ is repositioned as

$$\left(\hat{Q}_0^{k+1}\right)' = \left(\hat{Q}_0^k\right)' \hat{q}_k \tag{31}$$

This process is repeated for all joints. The second stage is finished when the position and orientation of joint n are updated.

Post iteration

When all joints have been updated, the position difference between the new end effector and the target can be calculated. The new position of the end effector should be closer to the target. If their difference is more than the error tolerance the main iteration is repeated, otherwise the algorithm is terminated.

Algorithm 1: A full iteration of the Dual Quaternion FABRIK algorithm.

Input : The joint initial positions and orientation $\hat{Q}_0^i$, for $i = 0,\ldots n$ relative to the fixed reference frame and the target position in Dual Quaternion form $\hat{Q}_0^t$.

Output : The new joint position and orientation $\hat{Q}_0^j$ and new joint configuration $\hat{q}_j$ for $j = 1,\ldots n$.

1.1 % Find each joint configuration and first line vector.

1.2 **for** $j = 1,\ldots n$ **do**

1.3 $\hat{q}_j = \left(Q_0^{j-1}\right)^* Q_0^j$

1.4 $u_j = 2q_{d_j} {q_{r_j}}^* \Big/ \left\|2q_{dj} {q_{r_j}}^*\right\|$

1.5 % In case there are more than one direction of rotation.

1.6 $u_j = \begin{bmatrix} 0 & u_{v_j} - \left({u_{v_j}}^T d\right) d \end{bmatrix}^T$

1.7 **end**

1.8 % Set error tolerance.

1.9 $tol = 0.01$

1.10 % Calculate the difference between the initial end effector and the target.

1.11 $\hat{e} = \left(\hat{Q}_0^t\right)^* \hat{Q}_0^n$

1.12 % Calculate the position difference.

1.13 $error = \left\| 2e_d \left(e_r\right)^* \right\|$

1.14 **while** $error \geq tol$ **do**

1.15 % STAGE 1

1.16 % Set the first joint as the base $\hat{b}$.

1.17 $\hat{b} = \hat{Q}_0^1$

1.18 % Set end effector joint as the target.

1.19 $\hat{Q}_0^n = \hat{Q}_0^t$

1.20 **for** $k = n-1,\ldots 1$ **do**

1.21 % Find the difference between joint k and $k+1$.

1.22 $\hat{r}_k = \left(\hat{Q}_0^k\right)^* \hat{Q}_0^{k+1}$

1.23 % Convert the Dual Quaternion difference into vector Quaternion and normalize.

1.24 $v_k = 2r_{d_k} {r_{r_k}}^* \Big/ \left\| 2r_{d_k} {r_{r_k}}^* \right\|$

1.25 % In case there are more than one direction of rotation.

1.26 $v_k = \begin{bmatrix} 0 & v_{v_k} - \left({v_{v_k}}^T d\right) d \end{bmatrix}^T$

1.27 % Find the Dual Quaternion rotation.

1.28 $\hat{\lambda}_k = \begin{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T - u_{k+1} v_k, 0 \end{bmatrix}$

1.29 % Find the Dual Quaternion rotation of joint $k+1$.

1.30 $\hat{\beta}_k = \left(\widehat{q_{rot\ k+1}}\right)^* \left(\hat{\lambda}_k\right)^* \left(\widehat{Q_{rot\ 0}}^k\right)^* \widehat{Q_{rot\ 0}}^{k+1}$

1.31 % Find the new joint k.

1.32 $\hat{Q}_0^k = \hat{Q}_0^{k+1} \left(\hat{\beta}_k\right)^* \left(\hat{q}_{k+1}\right)^*$

1.33 **end**

1.34 % STAGE 2

1.35 % Set the first joint to its initial position.

1.36 $\hat{Q}_0^1 = b$

1.37 **for** $k = 1,\ldots n-1$ **do**

1.38 % Find the difference between joint k and $k+1$.

1.39 $\hat{r}_k = \left(\hat{Q}_0^k\right)^* \hat{Q}_0^{k+1}$

1.40 % Convert the Dual Quaternion difference into vector Quaternion and normalize.

1.41 $v_k = 2r_{d_k} {r_{r_k}}^* \Big/ \left\| 2r_{d_k} {r_{r_k}}^* \right\|$

1.42 % In case there are more than one direction of rotation.

1.43 $v_k = \begin{bmatrix} 0 & v_{v_k} - \left({v_{v_k}}^T d\right) d \end{bmatrix}^T$

1.44 % Find the Dual Quaternion rotation.

1.45 $\hat{\lambda}_k = \left[\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T - u_{k+1} v_k, 0 \right]$

1.46 % Update the rotation for joint configuration k.

1.47 $\hat{q}_k = \hat{q}_k \hat{\lambda}_k$

1.48 % Update joint k.

1.49 $\hat{Q}_0^k = \hat{Q}_0^k \hat{\lambda}_k$

1.50 % Find new joint $k+1$.

1.51 $\hat{Q}_0^{k+1} = \hat{Q}_0^k \hat{q}_{k+1}$

1.52 **end**

1.53 % Calculate the difference between the new end effector and target.

1.54 $\hat{e} = \left(\hat{Q}_0^t \right)^* \hat{Q}_0^n$

1.55 % Calculate the position error.

1.56 $error = \left\| 2 e_d \left(e_r \right)^* \right\|$

1.57 **end**

Joint limit

Joint limit is the rotation limit on each joint configuration as the human knee or elbow. This limitation may occur due to the manipulator design or the limitation in the electric motor used. To ensure the joint configuration does not exceed the rotational limit, the algorithm is applied in both stages.

Joint Limit on Stage 1

Since the first stage starts from the joint n, the joint configuration $k+1$ must be ensured rotated within their limit. Therefor, joint k will be repositioned, for $k = n-1, \ldots 1$. In this article, if the joint configuration exceeds the limit, the joint configuration is set to their maximum or minimum limit. In the first stage, the rotation of joint configuration $k+1$ can be calculated by finding the rotation difference between joint k and new joint $k+1$ as

$$\widehat{q_{rot\ k+1}} = \left(\widehat{Q_{rot\ 0}}^k \right)^* \left(\widehat{Q_{rot\ 0}}^{k+1} \right) \tag{32}$$

In case the rotation on joint configuration $k+1$ exceed their limit, joint k will be repositioned as follow

$$\begin{aligned} \hat{\beta}_k &= \left(\widehat{q_{rot\ k+1}} \right)^* \left(\left(\widehat{\lim}_{k+1} \right)^* \widehat{q_{rot\ k+1}} \right)^* \left(\widehat{Q_{rot\ 0}}^k \right)^* \widehat{Q_{rot\ 0}}^{k+1} \\ \hat{Q}_0^k &= \hat{Q}_0^{k+1} \left(\hat{\beta}_k \right)^* \left(\hat{q}_{k+1} \right)^* \end{aligned} \tag{33}$$

Where $\widehat{\lim}_{k+1}$ is the joint configuration $k+1$ rotation limit in pure Dual Quaternion rotation.

Joint Limit on Stage 2

During the second stage, the joint $k+1$ is repositioned so that the rotation on joint configuration k does not exceed their limit. After the joint configuration k is updated as in equation (29), the rotation can be checked. In case the rotation exceeds their limit, the rotation on joint configuration k can be set on the maximum or minimum rotation limit.

Orientation control

In the previous section, the basic concept of the proposed algorithm is presented. This section explains the orientation control algorithm in Dual Quaternion form. The orientation control uses the swing and twist approach. In this algorithm, joint $n-1$ is assumed responsible for the orientation control and end effector joint n are fixed. Should be noted, that the orientation control algorithm is applied separately from the position control algorithm. First of all, the target orientation can be defined as

$$\widehat{Q_{img}}_0^t = \hat{Q}_0^{n-1}\left(\widehat{Q_{rot}}_0^{n-1}\right)^*\left(\widehat{Q_{rot}}_0^t\right)\left(\hat{Q}_0^{n-1}\right)^*\left(\hat{Q}_0^n\right) \tag{34}$$

In case the joint $n-1$ and joint n located in the same position, an imaginary distance should be applied. Joint j , which is responsible for the orientation control can be rotated so that the end effector is pointed in the same direction as the target. Using equation (17), (18) and (19) the first line vector from joint j to the current end effector can be found as follow

$$\begin{aligned}\hat{p} &= \left(\hat{Q}_0^j\right)^*\hat{Q}_0^n \\ u &= \frac{2p_d\left(p_r\right)^*}{\left\|2p_d\left(p_r\right)^*\right\|}\end{aligned} \tag{35}$$

The second line vector, from joint j to the target orientation can be defined as follow

$$\begin{aligned}\hat{r} &= \left(\hat{Q}_0^j\right)^*\widehat{Q_{img}}_0^t \\ v &= \frac{2r_d\left(r_r\right)^*}{\left\|2r_d\left(r_r\right)^*\right\|}\end{aligned} \tag{36}$$

It should be noted, that both of the line vector must be mapped on their plane as in equation (5). The Dual Quaternion pure rotation can be found using equation (24). Finally, the joint configuration j can be rotated as

$$\hat{q}_j = \hat{q}_j\lambda \tag{37}$$

Rotation around it own axis

In some case the joint only rotate around it own axis (i.e. twist). This case occurs when $\|u\| = 0$. In this situation, the Dual Quaternion pure rotation can be defined as the orientation difference between the joint j and target.

$$\hat{\lambda} = \left(\widehat{Q_{rot\,0}}^{\,j}\right)^{*} \widehat{Q_{rot\,0}}^{\,t} \tag{38}$$

Algorithm 2: A full iteration of the orientation control in Dual Quaternion form.

Input : The joint that responsible for orientation control $\hat{Q}_0^j$, the joint configuration $\hat{q}_j$, the current end effector $\hat{Q}_0^n$, the joint $\hat{Q}_0^{n-1}$ and the target position $\hat{Q}_0^t$.

Output : The new joint configuration $\hat{q}_j$.

2.0 % Find the difference between the last two joints and check their distance.

2.1 $\hat{k} = \left(\hat{Q}_0^{n-1}\right)^{*} \left(\hat{Q}_0^{n}\right)$

2.2 $d = 2k_d k_r$

2.3 **if** $\|d\| = 0$

2.4 $\hat{k} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 \end{bmatrix}^T$

2.5 **end**

2.6 % Find the target orientation direction.

2.7 $\widehat{Q_{img\,0}}^{\,t} = \hat{Q}_0^{n-1} \left(\widehat{Q_{rot\,0}}^{\,n-1}\right)^{*} \widehat{Q_{rot\,0}}^{\,t} \hat{k}$

2.8 % Find the first line vector.

2.9 $\hat{p} = \left(\hat{Q}_0^{j}\right)^{*} \hat{Q}_0^{n}$

2.10 $u = 2p_d \left(p_r\right)^{*} \Big/ \left\| 2p_d \left(p_r\right)^{*} \right\|$

2.11 % Map the line vector on the plane.

2.12 $u = \begin{bmatrix} 0 & u_v - \left(u_v{}^T d\right) d \end{bmatrix}^T$

2.13 **if** $\|u\| = 0$

2.14 $\hat{\lambda} = \left(\widehat{Q_{rot\,0}}^{\,j}\right)^{*} \widehat{Q_{rot\,0}}^{\,t}$

2.15 **else**

2.16 % Find the second line vector.

2.17 $\hat{r} = \left(\hat{Q}_0^{j}\right)^{*} \widehat{Q_{img\,0}}^{\,t}$

2.18 $v = 2r_d \left(r_r\right)^{*} \Big/ \left\| 2r_d \left(r_r\right)^{*} \right\|$

2.19 % Map the line vector on the plane.

2.20 $v = \begin{bmatrix} 0 & v_v - \left(v_v{}^T d\right) d \end{bmatrix}^T$

2.21 % Find the Dual Quaternion pure rotation.

2.22 $\hat{\lambda} = \begin{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T - uv, 0 \end{bmatrix}$

2.23 **end**

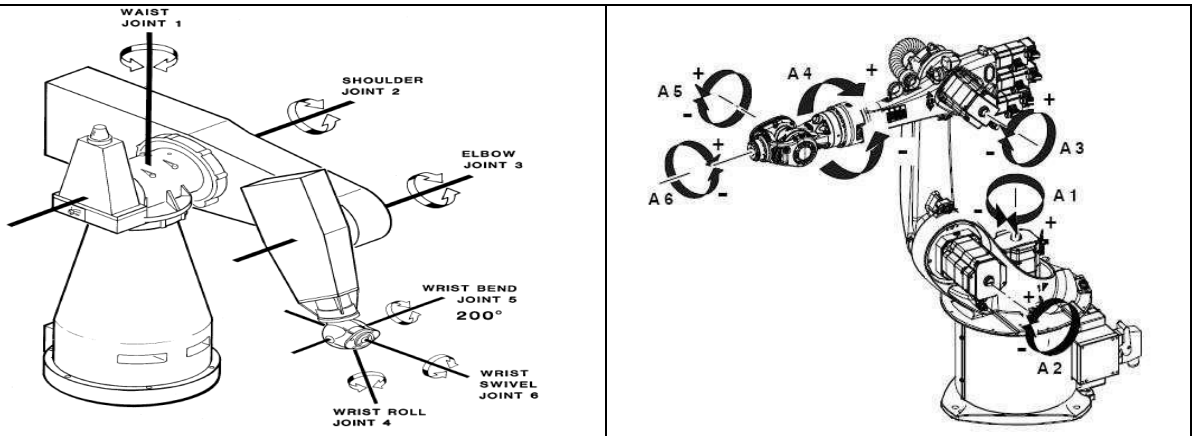
2.24 $\hat{q}_j = \hat{q}_j \hat{\lambda}$

MULTIPLE DIRECTION OF ROTATION

This section, explain how to apply the proposed algorithm for a manipulator with multiple directions of rotation. It should be noted, for a manipulator with multiple directions of rotation, each joint should only rotate with respect to its direction of rotation while moving the end-effector position and orientation closer to the target. To apply the algorithm in multiple direction of rotation, the following step are applied.

1. Define each joint direction of rotation.
2. Define the joint responsible for position control.
3. Define the joint responsible for orientation control.
4. Group the position control joint based on their direction of rotation.
5. Run the position control algorithm for all the direction of rotation from Step 4.
6. Run the orientation control for the joint defined in Step 2.
7. Check for system error.
8. If the error is greater than the stated acceptable tolerance, repeat step 5 through 7. Otherwise, the process end.

To explain further step 1 to 4, PUMA and KUKA manipulator with 6 degree of freedom as in FIGURE 3 is used. The first three joints (i.e. joint 1,2 and 3) are responsible for the position control and the last three joints (i.e. joint 4,5 and 6) are responsible for the orientation control. For position control, the joints are divided into two groups. The first group, is for joint 1 that is rotated on the z-axis. The second group is for joint 2 and joint 3 which is rotated on the y-axis.



(a) **(b)**

FIGURE 3. Most used type of manipulator model. (a) PUMA manipulator, (b) KUKA manipulator

EXPERIMENTAL RESULTS

A target database consisting of 7200 points has been created for the validation and testing of the proposed Inverse Kinematics algorithm. The database consists of only reachable targets and targets with different distances from the end effectors for model in FIGURE 1. The error tolerance for these experiments were set to be less than 0.01.

This article compares the proposed algorithm with Dual Quaternion Cyclic Coordinate Descent (CCD) algorithm. They were compared with respect to their processing time, computational cost, number of iterations needed to reach the target and the error convergence. The runtimes were recorded in milliseconds and were measured with custom MATLAB code on an Intel Core i5 2.5 GHz. No optimizations were used for both methods reported in TABLE 1.

The Dual Quaternion FABRIK algorithm requires on average 7 iterations with 62.16 milliseconds to reach the target. However, Dual Quaternion CCD requires in average 12 iterations with 66.03 milliseconds. FIGURE 4 is an

example of one of the errors plotted against the number of iterations for both methods. The proposed algorithm error converges at a faster rate compared to the Dual Quaternion CCD. In the beginning the error converges accelerated then gradually decelerate as it reaches the target.

One of the advantages that has been observed from the experiment is that, the proposed algorithm produces better movement in each iteration as shown in FIGURE 5 (a). Dual Quaternion CCD tend to produce an excessive movement at the beginning before converging to the target as in FIGURE 5 (b). Although, when the target is closer to the base the proposed algorithm has been observed to require a higher number of iterations. The reason for this behavior is not analyzed in this article.

TABLE 1. To format a table caption, use the Microsoft Word template style: Table Caption. The text

	Number of Iteration	**Matlab exe. Time (ms)**	**Time Per Iteration (ms)**
DQ FABRIK	6.94850	62.16	8.685
DQ CCD	11.66467	66.03	5.3075

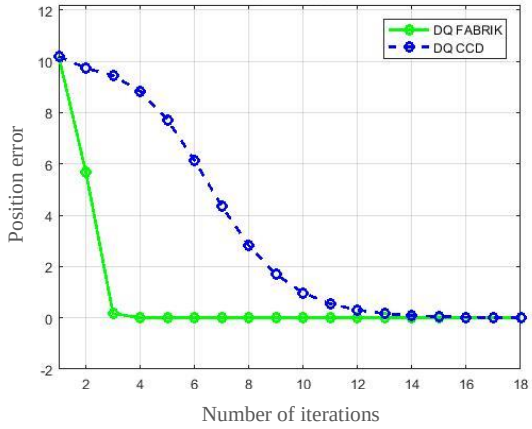
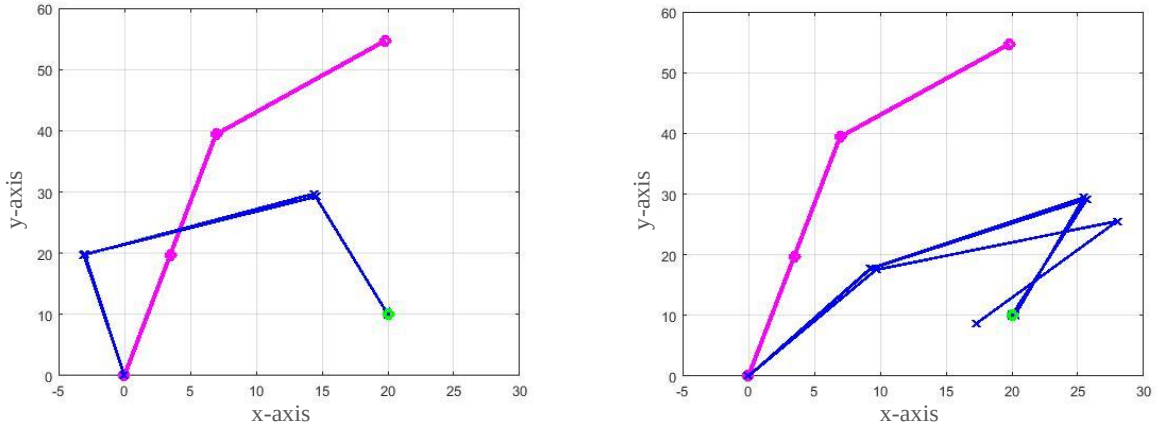


FIGURE 4. Number of iteration against the position error



(a) **(b)**

FIGURE 5. An example of the poses for both Dual Quaternion algorithm in each iteration. (a) Dual Quaternion FABRIK result, (b) Dual Quaternion CCD result

Moreover, the experiment tested both algorithms to move along a continues point on a circle. The end effector trajectory for both algorithms are plotted in FIGURE 6. From the results in FIGURE 6 (a) and (b), it can be concluded that when the distances between the points are closer to each other, both methods perform equally well. However, when the distances between the points are further apart the proposed algorithm perform better than Dual Quaternion CCD.

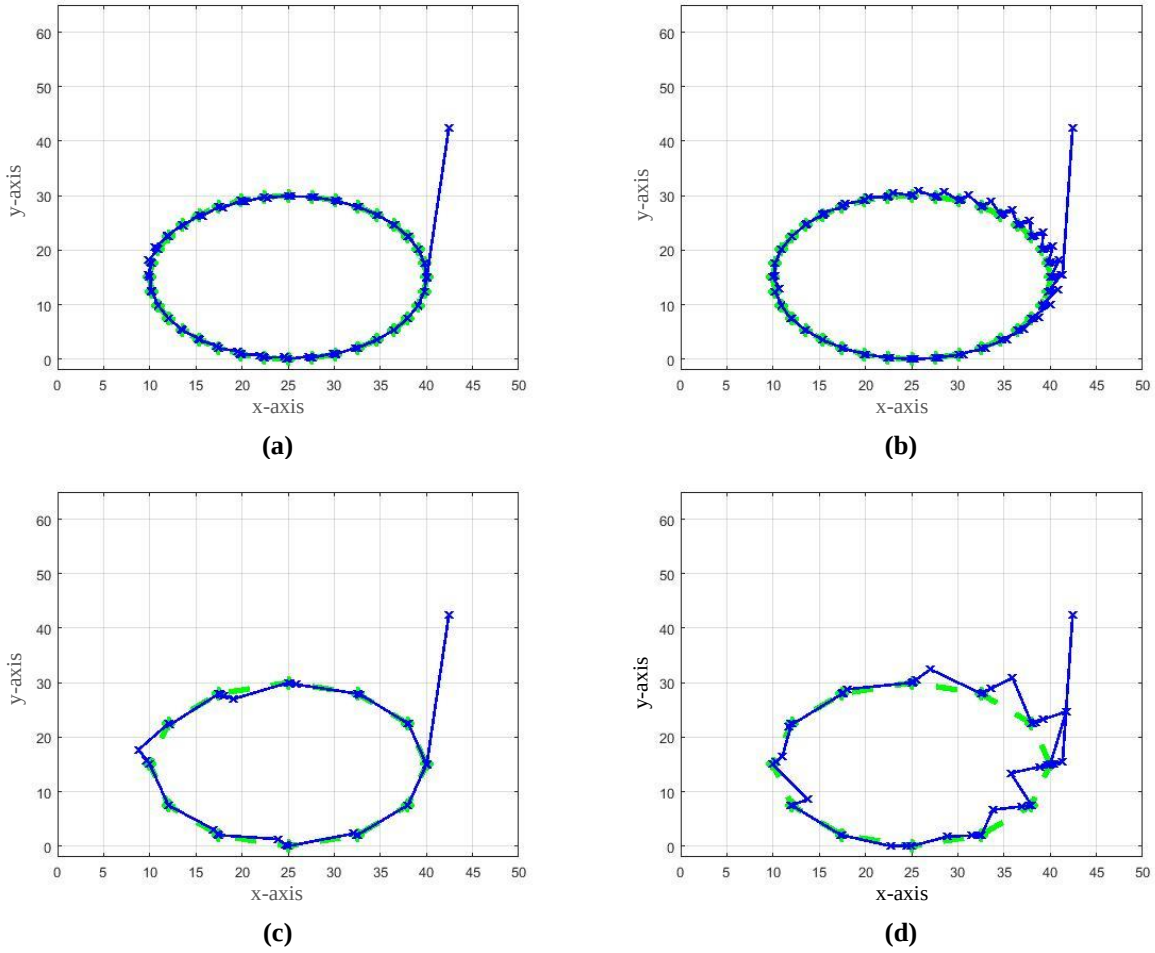


FIGURE 6. Example of the end effector trajectory for both algorithms following a continuous point. (a) Dual Quaternion FABRIK end effector trajectory with points 10 degree apart, (b) Dual Quaternion CCD end effector trajectory with points 10 degree apart, (c) Dual Quaternion FABRIK end effector trajectory with points 30 degree apart, (d) Dual Quaternion CCD end effector trajectory with points 30 degree apart.

This article test the proposed algorithm with different type of manipulator model such as KUKA and PUMA manipulators as shown in FIGURE 3. FIGURE 7 (a) represents a KUKA manipulator with z-axis rotation on joint 5. FIGURE 7 (c) represent KUKA manipulator with y-axis rotation on joint 5. While FIGURE 7 (e) represent the PUMA manipulator. Based on the result, the final position and orientation of the manipulator produces the same poses. However, each manipulator model requires a different number of iterations to reach the target's position and orientation, depending on their configuration. As in FIGURE 7 (d) the number of iterations were the highest. The orientation control joint 5, rotate on the y-axis. However, our target orientation is on the z-axis on the 90 degrees. In this experiment, it is obvious that the orientation control causes the manipulator to require a higher number of iterations. If the orientation control is ignored all manipulator models will reach the target at a faster rate with similar results. In conclusion from this experiment, the Dual Quaternion Inverse Kinematics algorithm based on FABRIK methodology has proven to work for all models. There are definitely some room for improvement on the orientation control.

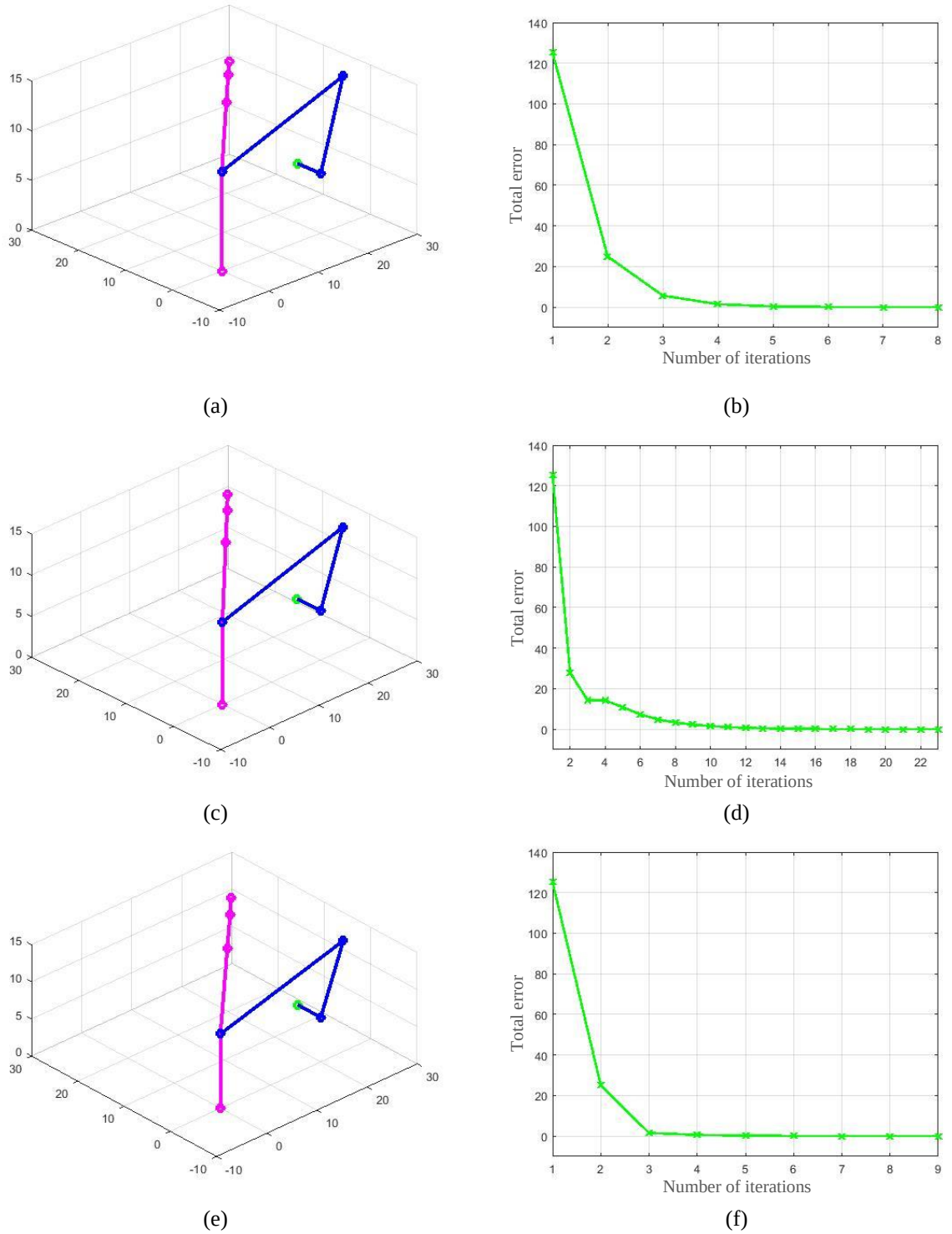


FIGURE 7. Example of the result poses for Dual Quaternion FABRIK algorithm with different manipulator models. (a) KUKA manipulator with joint 5 rotate on z-axis, (b) Number of iterations against orientation and position error for KUKA manipulator with joint 5 rotate on z-axis (c) KUKA manipulator with joint 5 rotate on y-axis, (d) Number of iterations against orientation and position error for KUKA manipulator with joint 5 rotate on y-axis, (e) PUMA manipulator, (f) Number of iterations against orientation and position error for PUMA manipulator.

CONCLUSION AND FUTURE WORK

This article described an alternative solution to solve inverse kinematics problems in the Dual Quaternion Form. Based on the results from the experiment, the proposed algorithm on average required less number of iterations compared to Dual Quaternion CCD. Moreover, with the concept of updating the joint in a forward and backward iteration mode, the proposed algorithm provided smooth poses in each iteration and does not cause any excessive movement. This approach caused the algorithm to accelerate at the beginning and gradually decelerate when as it reached the target.

Furthermore, the same algorithm has been proven to be applicable to different model of manipulators. This algorithm required minimal changes in their directions of rotation. This is because the proposed algorithm considered the direction of rotation in their iteration process. Besides, no additional task is required to be applied to the joint's limit algorithm as the joint rotation information is available at all time.

However, during the experiment, it is observed that the number of iterations increases significantly as the target approaches the base. In the future, it is suggested to test the algorithm with coefficient on the joint rotation. In addition, the orientation control algorithm was designed as a separate algorithm. This causes more iterations as the error for orientation and position move inconsistently. In some cases, the position error tends to converge faster than the orientation error or vice versa. Because of this, in the future it is suggested to reformulate the orientation control algorithm and combine it in the main position algorithm.

Moreover, during the experiment, it has been observed that the algorithm suffers from a minor singularity issues. This is seen in traditional FABRIK and CCD [5]. To avoid these issues, it is best to find a solution to identify when the singularity may occur during the iteration process. Lastly, it is suggested to design a solution that would be able to check if the target orientation and position are reachable in Dual Quaternion form.

ACKNOWLEDGMENTS

The authors would like to thank the editors, reviewers and all the members of the Department of System Analysis and Control for their support and insightful feedback that has contributed towards making this article more clear, concise, and correct.

REFERENCES

1. E. Sariyildiz and H. Temeltas, Turkish Journal of Electrical Engineering and Computer Sciences **20**, 607 (2012).
2. B. Kenwright, Journal of Graphics Tools **16**, 177 (2012).
3. B. Kenwright, International Journal on Advances in Intelligent Systems **6**, 53 (2013).
4. A. Aristidou and J. Lasenby, Graphical Models **73**, 243 (2011).
5. A. Aristidou, Y. Chrysanthou, and J. Lasenby, Comp. Anim. Virtual Worlds **27**, 35 (2016).
6. B. Kenwright, in *20th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2012, Plzen, 2012*, edited by V. Skala (Union Agency, Plzen, 2012), pp. 1-10.
7. S. Akhramovich, V. Malyshev, and A. Starkov, All-Russian Scientific-Technical Journal "Polyot" ("Flight") **4**, 9 (2018).
8. E. Sariyildiz, E. Cakiray, and H. Temeltas, International Journal of Advanced Robotic Systems **8**, 64 (2011).
9. L. Josuet, B. Carlos, L. Hsien-I, H. Te-Sheng, and W. Chun-Sheng, in *2016 International Automatic Control Conference (CACS), Taichung, 2016* (IEEE, 2017), pp. 77–82.
10. R. Bai and Z. Liu, in *2017 International Automatic Control Conference (CACS), Pingtung, 2017* (IEEE, 2018), pp. 1–6.
11. E. Sariyildiz and H. Temeltas, in *2009 International Conference on Mechatronics and Automation, Changchun, 2009* (IEEE, 2009), pp. 26–31.
12. E. Sariyildiz and H. Temeltas, in *2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Singapore, 2009* (IEEE, 2009), pp. 338–343.
13. L. Dorst and J. Lasenby, *Guide to Geometric Algebra in Practice* (Springer Science & Business Media, 2011), pp. 47–62.

14. A. V. Salazar, “Dynamic modeling and control of spacecraft robotic systems using dual quaternion,” Ph.D. thesis, Georgia Institute of Technology, 2018.
15. R. Mukundan, International Journal of Computer Applications in Technology **34**, 303 (2009).
16. E. Özgür and Y. Mezouar, Robotics and Autonomous Systems **77**, 66 (2016).
17. M. Gouasmi, M. Ouali, and F. Brahim, International Journal of Robotics and Automation (IJRA) **1**, 13 (2012).
18. M. Rodelo, J.L. Villa, J. Duque, and E. Yime, in *2018 IEEE 2nd Colombian Conference on Robotics and Automation (CCRA), Barranquilla, 2018* (IEEE, 2018), pp. 1–6.
19. J. Chen, D. Han, H. Nie, and M. Cheng, Journal of Vibroengineering **16**, 2813 (2014).
20. L. Guillaume, L. Philippe, and B. Gunnar, Frontiers in Behavioral Neuroscience **7**, 7 (2013).
21. Q. Chen, S. Zhu, and X. Zhang, International Journal of Advanced Robotic Systems **12**, 140 (2015).
22. I. Mas and C. Kitts, J Intell Robot Syst **87**, 643 (2017).
23. S. Payandeh and A.A. Goldenberg, Journal of Robotic Systems **4**, 771 (1987).