Dual Iterative Displacement Analysis of Spatial Mechanisms Using the C^H Programming Language

Harry H. Cheng
Sean Thompson
Integration Engineering Laboratory
Department of Mechanical and Aeronautical Engineering
University of California
Davis, CA 95616
Email: hhcheng@ucdavis.edu
World Wide Web: http://iel.ucdavis.edu

Abstract

A dual iterative method for displacement analysis of spatial mechanisms is presented in this paper. The algorithm and formulation based upon 3x3 dual transformation matrices are succinct. They can be implemented concisely in the C^H programming language, a superset of C, which handles dual, dual double, and dual computational arrays polymorphically. The algorithm is numerically verified by dual iterative displacement analysis of spatial mechanisms. Comparison studies on stability, convergence, performance near singularities, and CPU time of the dual iterative method versus the real iterative method are conducted. It is shown that both dual and real iterative algorithms are stable with a comparable convergence rate even near singularities. But, the computational speed of the dual iterative formulation is about 2.18 times faster than that of the real iterative formulation when formulations are translated into C^H programs for numerical calculation.

1 INTRODUCTION

Displacement analysis of spatial mechanisms can be performed using a variety of displacement operators or coordinate transformation forms such as 3x3 dual matrices (Soni and Harrisberger, 1968; Yang, 1969; Pennock and Yang, 1985; McCarthy, 1990), 4x4 Denavit-Hartenberg (DH) matrices (Denavit and Hartenberg, 1955), 4x4 screw matrices (Suh and Radcliffe, 1978), 6x6 line coordinate transformation matrices (Yuan, 1970), 2x2 dual matrices (Denavit, 1958), dual quaternions (Hamilton, 1899; Yang and Freudenstein, 1964), etc. A 3x3 dual transformation matrix is orthogonal and similar to a 3x3 pure rotational matrix with real elements. Because of their compact nature, the 3x3 dual matrices are widely used for analytical treatment of spatial mechanisms. However, programming with dual numbers is cumbersome in traditional programming languages. When numerical computations are needed, dual formulas may have to be partitioned into separate real and dual parts. The partition process is tedious and error-prone, the conciseness of dual formulas diminishes. Therefore, the use of 4x4 DH matrices and 4x4 screw matrices is more popular for numerical computations.

The C^H programming language merges C and FORTRAN with many additional extensions (Cheng, 1993; 1995a). Dual number in C^H is treated as a basic built-in data type like real and complex numbers (Cheng, 1994). Computation of dual formulas and dual matrices in C^H is just as efficient and

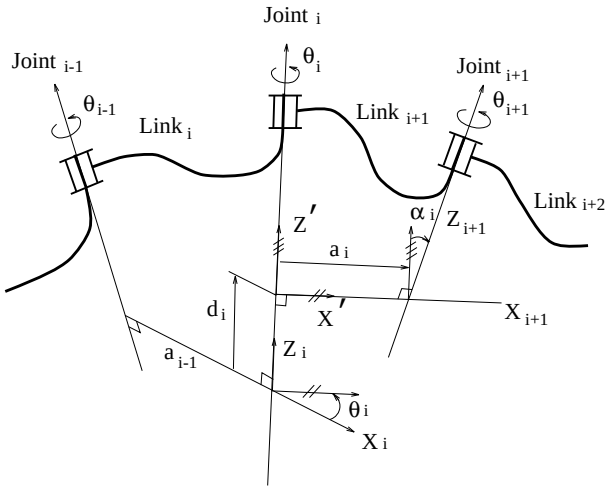

FIGURE 1: THE DENAVIT-HARTENBERG PARAMETERS FOR ADJACENT LINKS.

convenient as computation of real or complex ones. The C$^{\mathrm{H}}$ programming language has been used for displacement analysis of spatial mechanisms with analytical solutions (Cheng, 1994; Cheng and Thompson, 1995). Formulation and programming with dual numbers and dual arrays under the C$^{\mathrm{H}}$ programming paradigm are quite different from those in FORTRAN, C, or other programming languages. This paper demonstrates dual iterative formulation and programming techniques with dual numbers and dual computational arrays under the C$^{\mathrm{H}}$ programming paradigm through dual iterative displacement analysis of spatial mechanisms.

A real iterative method based on 4x4 DH matrices for displacement analysis of spatial mechanisms was first presented by Uicker, et al. (1964). The algorithm has been extended for inverse kinematics of robot manipulators (Gupta and Kazerounian, 1985). Using the principle of transference, Fischer (1988) presented a semi-dual iterative method for evaluation of the translational displacements of spatial mechanisms. We will present a dual iterative method based on 3x3 dual matrices for displacement analysis of spatial mechanisms in this paper. The dual formulations intended for implementation in the C$^{\mathrm{H}}$ programming language are concise. They can be conveniently implemented with only one page of code. These formulations are quite different from those (Fischer, 1988) intended for implementation in FORTRAN or other languages. Extensive numerical studies on error stability, convergence rate, performance near singularities, and speed in CPU time of the dual iterative algorithm in comparison with the real iterative algorithm in (Uicker, et al., 1964) will also be conducted in this paper. To narrow the historical gap and for comparison studies, notations similar to those in (Uicker, et al., 1964) will be used in this paper.

2 DUAL FORMULATION FOR ITERATIVE DISPLACEMENT ANALYSIS

A 3x3 dual matrix can be used to describe the transformation between the coordinate systems $X_iY_iZ_iO_i$ and $X_{i-1}Y_{i-1}Z_{i-1}O_{i-1}$ shown in Figure 1. This dual transformation matrix can be expressed as (Soni and Harrisberger, 1968; Yang, 1969; Cheng, 1994)

$$\widehat{\mathbf{A}}_i(\widehat{\theta}_i) = \begin{bmatrix} \cos\widehat{\theta}_i & -\sin\widehat{\theta}_i\cos\widehat{\alpha}_i & \sin\widehat{\theta}_i\sin\widehat{\alpha}_i \\ \sin\widehat{\theta}_i & \cos\widehat{\theta}_i\cos\widehat{\alpha}_i & -\cos\widehat{\theta}_i\sin\widehat{\alpha}_i \\ 0 & \sin\widehat{\alpha}_i & \cos\widehat{\alpha}_i \end{bmatrix}, \tag{1}$$

where dual angles $\widehat{\theta}_i$ and $\widehat{\alpha}_i$ are defined as $\theta_i + \varepsilon d_i$ and $\alpha_i + \varepsilon a_i$, respectively. The loop-closure equation for mechanisms of n links becomes

$$\widehat{\mathbf{A}}_1(\widehat{\theta}_1)\widehat{\mathbf{A}}_2(\widehat{\theta}_2)\ldots\widehat{\mathbf{A}}_n(\widehat{\theta}_n) = \mathbf{I}, \tag{2}$$

where $\mathbf{I}$ is a 3x3 identity matrix and $\widehat{\mathbf{A}}_i(\widehat{\theta}_i)$ is given in equation (1).

To derive the dual formula for iterative displacement analysis of spatial mechanisms, a dual joint angle is defined as

$$\widehat{\theta}_i = \overline{\widehat{\theta}}_i + d\widehat{\theta}_i, \tag{3}$$

where $\widehat{\theta}_i$ is the exact dual angle, $\overline{\widehat{\theta}}_i$ is the estimate of the dual angle, and $d\widehat{\theta}_i$ is an infinitesimal small dual number between the exact and estimate dual angles. Assuming that the input angle θ_1 is given, the loop closure equation (2) becomes

$$\widehat{\mathbf{A}}_1(\widehat{\theta}_1)\widehat{\mathbf{A}}_2(\overline{\widehat{\theta}}_2 + d\theta_2)\ldots\widehat{\mathbf{A}}_n(\overline{\widehat{\theta}}_n + d\widehat{\theta}_n) = \mathbf{I}. \tag{4}$$

And the transformation matrix $\widehat{\mathbf{A}}_i$ in equation (1) can be expressed in terms of $\overline{\widehat{\theta}}_i$ and $d\widehat{\theta}_i$ as

$$\widehat{\mathbf{A}}_i(\overline{\widehat{\theta}}_i + d\widehat{\theta}_i) = \begin{bmatrix} \cos(\overline{\widehat{\theta}}_i + d\widehat{\theta}_i) & -\sin(\overline{\widehat{\theta}}_i + d\widehat{\theta}_i)\cos\widehat{\alpha}_i & \sin(\overline{\widehat{\theta}}_i + d\widehat{\theta}_i)\sin\widehat{\alpha}_i \\ \sin(\overline{\widehat{\theta}}_i + d\widehat{\theta}_i) & \cos(\overline{\widehat{\theta}}_i + d\widehat{\theta}_i)\cos\widehat{\alpha}_i & -\cos(\overline{\widehat{\theta}}_i + d\widehat{\theta}_i)\sin\widehat{\alpha}_i \\ 0 & \sin\widehat{\alpha}_i & \cos\widehat{\alpha}_i \end{bmatrix} \tag{5}$$

$$= \begin{bmatrix} \cos\overline{\widehat{\theta}}_i & -\sin(\overline{\widehat{\theta}}_i)\cos\widehat{\alpha}_i & \sin\overline{\widehat{\theta}}_i\sin\widehat{\alpha}_i \\ \sin\overline{\widehat{\theta}}_i & \cos(\overline{\widehat{\theta}}_i)\cos\widehat{\alpha}_i & -\cos\overline{\widehat{\theta}}_i\sin\widehat{\alpha}_i \\ 0 & \sin\widehat{\alpha}_i & \cos\widehat{\alpha}_i \end{bmatrix} + \begin{bmatrix} -\sin\overline{\widehat{\theta}}_i & -\cos\overline{\widehat{\theta}}_i\cos\widehat{\alpha}_i & \cos\overline{\widehat{\theta}}_i\sin\widehat{\alpha}_i \\ \cos\overline{\widehat{\theta}}_i & -\sin\overline{\widehat{\theta}}_i\cos\widehat{\alpha}_i & \sin\overline{\widehat{\theta}}_i\sin\widehat{\alpha}_i \\ 0 & \sin\widehat{\alpha}_i & \cos\widehat{\alpha}_i \end{bmatrix} d\widehat{\theta}_i. \tag{6}$$

The form of the first matrix in equation (6) is the same as equation (1). The second matrix in equation (6) is the first partial derivative of $\widehat{\mathbf{A}}_i$ with respect to $\widehat{\theta}_i$ evaluated at $\overline{\widehat{\theta}}_i$. Equation (6) can be rewritten as

$$\begin{aligned} \widehat{\mathbf{A}}_i(\overline{\widehat{\theta}}_i + d\overline{\widehat{\theta}}_i) &= \widehat{\mathbf{A}}_i(\overline{\widehat{\theta}}_i) + \left[\frac{\partial\widehat{\mathbf{A}}_i(\widehat{\theta}_i)}{\partial\widehat{\theta}_i}\bigg|_{\widehat{\theta}_i=\overline{\widehat{\theta}}_i}\right] d\widehat{\theta}_i \\ &= \overline{\widehat{\mathbf{A}}}_i + \left[\frac{\partial\widehat{\mathbf{A}}_i(\widehat{\theta}_i)}{\partial\widehat{\theta}_i}\bigg|_{\widehat{\theta}_i=\overline{\widehat{\theta}}_i}\right] d\widehat{\theta}_i. \end{aligned} \tag{7}$$

For clarity, $\widehat{\mathbf{A}}_i(\overline{\widehat{\theta}}_i)$ is represented as $\overline{\widehat{\mathbf{A}}}_i$ in equation (7). The first partial derivative of $\widehat{\mathbf{A}}_i$ with respect to $\widehat{\theta}_i$ evaluated at $\overline{\widehat{\theta}}_i$ can be achieved by multiplying $\overline{\widehat{\mathbf{A}}}_i$ with an appropriate $\widehat{\mathbf{Q}}$ matrix, that is

$$\frac{\partial\widehat{\mathbf{A}}_i(\widehat{\theta}_i)}{\partial\widehat{\theta}_i}\bigg|_{\widehat{\theta}_i=\overline{\widehat{\theta}}_i} = \widehat{\mathbf{Q}}\,\overline{\widehat{\mathbf{A}}}_i. \tag{8}$$

Matrix $\widehat{\mathbf{Q}}$ is determined to be

$$\widehat{\mathbf{Q}} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{9}$$

Therefore, equation (7) can be written as

$$\begin{aligned}\widehat{\mathbf{A}}_i(\overline{\widehat{\theta}}_i + d\widehat{\theta}_i) &= \overline{\widehat{\mathbf{A}}}_i + \widehat{\mathbf{Q}}\overline{\widehat{\mathbf{A}}}_i d\widehat{\theta}_i \\ &= (\mathbf{I} + \widehat{\mathbf{Q}}d\widehat{\theta}_i)\overline{\widehat{\mathbf{A}}}_i. \end{aligned} \tag{10}$$

Substituting equation (10) into equation (4), we get

$$\widehat{\mathbf{A}}_1(\mathbf{I} + \widehat{\mathbf{Q}}d\widehat{\theta}_2)\overline{\widehat{\mathbf{A}}}_2(\mathbf{I} + \widehat{\mathbf{Q}}d\widehat{\theta}_3)\overline{\widehat{\mathbf{A}}}_3 \ldots (\mathbf{I} + \widehat{\mathbf{Q}}d\widehat{\theta}_i)\overline{\widehat{\mathbf{A}}}_i \ldots (\mathbf{I} + \widehat{\mathbf{Q}}d\widehat{\theta}_n)\overline{\widehat{\mathbf{A}}}_n = \mathbf{I}. \tag{11}$$

Since an iterative process will be used to solve this equation, all higher order terms may be set to zero. Ignoring higher order terms, equation (11) becomes

$$\begin{aligned}(\widehat{\mathbf{A}}_1\widehat{\mathbf{Q}}\overline{\widehat{\mathbf{A}}}_2\overline{\widehat{\mathbf{A}}}_3 \ldots \overline{\widehat{\mathbf{A}}}_n)d\widehat{\theta}_2 + (\widehat{\mathbf{A}}_1\overline{\widehat{\mathbf{A}}}_2\widehat{\mathbf{Q}}\overline{\widehat{\mathbf{A}}}_3 \ldots \overline{\widehat{\mathbf{A}}}_n)d\widehat{\theta}_3 + \ldots + (\widehat{\mathbf{A}}_1\overline{\widehat{\mathbf{A}}}_2\overline{\widehat{\mathbf{A}}}_3 \ldots \overline{\widehat{\mathbf{A}}}_{n-1}\widehat{\mathbf{Q}}\overline{\widehat{\mathbf{A}}}_n)d\widehat{\theta}_n \\ \cong \mathbf{I} - (\widehat{\mathbf{A}}_1\overline{\widehat{\mathbf{A}}}_2\overline{\widehat{\mathbf{A}}}_3 \ldots \overline{\widehat{\mathbf{A}}}_n). \end{aligned} \tag{12}$$

Equation (12) can be written symbolically as the matrix equation

$$\widehat{\mathbf{B}}_2 + \widehat{\mathbf{B}}_3 + \ldots + \widehat{\mathbf{B}}_n \cong \mathbf{I} - \widehat{\mathbf{B}}_1, \tag{13}$$

where

$$\widehat{\mathbf{B}}_1 = \widehat{\mathbf{A}}_1\overline{\widehat{\mathbf{A}}}_2\overline{\widehat{\mathbf{A}}}_3 \ldots \overline{\widehat{\mathbf{A}}}_n \tag{14}$$

is defined as

$$\widehat{\mathbf{B}}_1 = \begin{bmatrix} \widehat{B}_{111} & \widehat{B}_{112} & \widehat{B}_{113} \\ \widehat{B}_{121} & \widehat{B}_{122} & \widehat{B}_{123} \\ \widehat{B}_{131} & \widehat{B}_{132} & \widehat{B}_{133} \end{bmatrix} \tag{15}$$

and

$$\widehat{\mathbf{B}}_i = (\widehat{\mathbf{A}}_1\overline{\widehat{\mathbf{A}}}_2 \ldots \overline{\widehat{\mathbf{A}}}_{i-1}\widehat{\mathbf{Q}}\overline{\widehat{\mathbf{A}}}_i \ldots \overline{\widehat{\mathbf{A}}}_n)d\widehat{\theta}_i \tag{16}$$

for $i = 2$ to n is defined as

$$\widehat{\mathbf{B}}_i = \begin{bmatrix} \widehat{B}_{i11} & \widehat{B}_{i12} & \widehat{B}_{i13} \\ \widehat{B}_{i21} & \widehat{B}_{i22} & \widehat{B}_{i23} \\ \widehat{B}_{i31} & \widehat{B}_{i32} & \widehat{B}_{i33} \end{bmatrix} d\widehat{\theta}_i. \tag{17}$$

If we define a 3x3 dual matrix $\widehat{\mathbf{E}}$ with its element as

$$\widehat{E}_{jk} = \sum_{i=2}^{n} \widehat{B}_{ijk}d\widehat{\theta}_i, \tag{18}$$

equation (13) becomes

$$\widehat{\mathbf{E}} \cong \mathbf{I} - \widehat{\mathbf{B}}_1 \tag{19}$$

or in the following dual matrix form

$$\begin{bmatrix} \widehat{E}_{11} & \widehat{E}_{12} & \widehat{E}_{13} \\ \widehat{E}_{21} & \widehat{E}_{22} & \widehat{E}_{23} \\ \widehat{E}_{31} & \widehat{E}_{32} & \widehat{E}_{33} \end{bmatrix} \cong \begin{bmatrix} 1 - \widehat{B}_{111} & -\widehat{B}_{112} & -\widehat{B}_{113} \\ -\widehat{B}_{121} & 1 - \widehat{B}_{122} & -\widehat{B}_{123} \\ -\widehat{B}_{131} & -\widehat{B}_{132} & 1 - \widehat{B}_{133} \end{bmatrix}. \tag{20}$$

Matrix $\hat{\mathbf{B}}_1$ in equation (15) is dual orthogonal because it is formed by the products of orthogonal matrices. Matrix $\hat{\mathbf{Q}}$ in equation (9) is dual antisymmetric because $\hat{Q}_{jk} = -\hat{Q}_{kj}$ and $\hat{Q}_{jj} = 0$. A useful theorem proved in the Appendix states that if $\hat{\mathbf{Q}}$ is dual antisymmetric and $\hat{\mathbf{H}}^T$ is the transpose of $\hat{\mathbf{H}}$, then

$$\hat{\mathbf{L}} = \hat{\mathbf{H}}\hat{\mathbf{Q}}\hat{\mathbf{H}}^T \tag{21}$$

is also dual antisymmetric.

Equation (16) can be written in a form where equation (21) can be applied by first inserting an identity matrix

$$\mathbf{I} = (\hat{\mathbf{A}}_1\overline{\hat{\mathbf{A}}}_2 \ldots \overline{\hat{\mathbf{A}}}_{i-1})^{-1}(\hat{\mathbf{A}}_1\overline{\hat{\mathbf{A}}}_2 \ldots \overline{\hat{\mathbf{A}}}_{i-1}) \tag{22}$$

into the equation. Equation (16) then becomes

$$\hat{\mathbf{B}}_i = (\hat{\mathbf{A}}_1\overline{\hat{\mathbf{A}}}_2 \ldots \overline{\hat{\mathbf{A}}}_{i-1})\hat{\mathbf{Q}}[(\hat{\mathbf{A}}_1\overline{\hat{\mathbf{A}}}_2 \ldots \overline{\hat{\mathbf{A}}}_{i-1})^{-1}(\hat{\mathbf{A}}_1\overline{\hat{\mathbf{A}}}_2 \ldots \overline{\hat{\mathbf{A}}}_{i-1})](\hat{\mathbf{A}}_i\overline{\hat{\mathbf{A}}}_2 \ldots \overline{\hat{\mathbf{A}}}_n)d\hat{\theta}_i \tag{23}$$

which can be reduced to

$$\hat{\mathbf{B}}_i = (\hat{\mathbf{A}}_1\overline{\hat{\mathbf{A}}}_2 \ldots \overline{\hat{\mathbf{A}}}_{i-1})\hat{\mathbf{Q}}(\hat{\mathbf{A}}_1\overline{\hat{\mathbf{A}}}_2 \ldots \overline{\hat{\mathbf{A}}}_{i-1})^{-1}\hat{\mathbf{B}}_1 d\hat{\theta}_i. \tag{24}$$

Because matrices $\overline{\hat{\mathbf{A}}}_i$ are dual orthogonal, equation (24) can be written as

$$\hat{\mathbf{B}}_i = (\hat{\mathbf{A}}_1\overline{\hat{\mathbf{A}}}_2 \ldots \overline{\hat{\mathbf{A}}}_{i-1})\hat{\mathbf{Q}}(\hat{\mathbf{A}}_1\overline{\hat{\mathbf{A}}}_2 \ldots \overline{\hat{\mathbf{A}}}_{i-1})^T\hat{\mathbf{B}}_1 d\hat{\theta}_i. \tag{25}$$

And finally, substituting this equation into equation (13) or (19), we get

$$\begin{aligned}\hat{\mathbf{E}} &= \sum_{i=2}^{n}(\hat{\mathbf{A}}_1\overline{\hat{\mathbf{A}}}_2 \ldots \overline{\hat{\mathbf{A}}}_{i-1})\hat{\mathbf{Q}}(\hat{\mathbf{A}}_1\overline{\hat{\mathbf{A}}}_2 \ldots \overline{\hat{\mathbf{A}}}_{i-1})^T\hat{\mathbf{B}}_1 d\hat{\theta}_i \\ &= \mathbf{I} - \hat{\mathbf{B}}_1.\end{aligned} \tag{26}$$

Using equation (21), an antisymmetric matrix $\hat{\mathbf{Q}}_1$ can be defined as

$$\begin{aligned}\hat{\mathbf{Q}}_1 &= \sum_{i=2}^{n}(\hat{\mathbf{A}}_1\overline{\hat{\mathbf{A}}}_2 \ldots \overline{\hat{\mathbf{A}}}_{i-1})\hat{\mathbf{Q}}(\hat{\mathbf{A}}_1\overline{\hat{\mathbf{A}}}_2 \ldots \overline{\hat{\mathbf{A}}}_{i-1})^T d\hat{\theta}_i \\ &= \begin{bmatrix} 0 & -\hat{Q}_{122} & -\hat{Q}_{132} \\ \hat{Q}_{122} & 0 & -\hat{Q}_{133} \\ \hat{Q}_{132} & \hat{Q}_{133} & 0 \end{bmatrix}.\end{aligned} \tag{27}$$

Equation (26) becomes

$$\hat{\mathbf{E}} = \hat{\mathbf{Q}}_1\hat{\mathbf{B}}_1 = \begin{bmatrix} 0 & -\hat{Q}_{122} & -\hat{Q}_{132} \\ \hat{Q}_{122} & 0 & -\hat{Q}_{133} \\ \hat{Q}_{132} & \hat{Q}_{133} & 0 \end{bmatrix}\begin{bmatrix} \hat{B}_{111} & \hat{B}_{112} & \hat{B}_{113} \\ \hat{B}_{121} & \hat{B}_{122} & \hat{B}_{123} \\ \hat{B}_{131} & \hat{B}_{132} & \hat{B}_{133} \end{bmatrix} \cong \mathbf{I} - \hat{\mathbf{B}}_1 \tag{28}$$

or

$$\begin{aligned}\hat{\mathbf{E}} &= \begin{bmatrix} -\hat{Q}_{122}\hat{B}_{121} - \hat{Q}_{132}\hat{B}_{131} & -\hat{Q}_{122}\hat{B}_{122} - \hat{Q}_{132}\hat{B}_{132} & -\hat{Q}_{122}\hat{B}_{123} - \hat{Q}_{132}\hat{B}_{133} \\ \hat{Q}_{122}\hat{B}_{111} - \hat{Q}_{133}\hat{B}_{131} & \hat{Q}_{122}B_{112} - \hat{Q}_{133}\hat{B}_{132} & \hat{Q}_{122}\hat{B}_{113} - \hat{Q}_{133}\hat{B}_{133} \\ \hat{Q}_{132}\hat{B}_{111} + \hat{Q}_{133}\hat{B}_{121} & \hat{Q}_{132}\hat{B}_{112} + \hat{Q}_{133}\hat{B}_{122} & \hat{Q}_{132}\hat{B}_{113} + \hat{Q}_{133}\hat{B}_{123} \end{bmatrix} \\ &\cong \begin{bmatrix} 1 - \hat{B}_{111} & -\hat{B}_{112} & -\hat{B}_{113} \\ -\hat{B}_{121} & 1 - \hat{B}_{122} & -\hat{B}_{123} \\ -\hat{B}_{131} & -\hat{B}_{132} & 1 - \hat{B}_{133} \end{bmatrix}.\end{aligned} \tag{29}$$

Since the initial estimates are given for $\widehat{\mathbf{B}}_1$ in equation (14), all of the $\widehat{B}_{1jk}$ elements are known quantities in this equation. Only the values of $\widehat{Q}_{122}$, $\widehat{Q}_{132}$, and $\widehat{Q}_{133}$ contain unknown quantities $d\widehat{\theta}_i$. The values can be found by choosing either the three upper-right or lower-left triangular equations. The three lower left triangular equations

$$\widehat{E}_{21} = \widehat{Q}_{122}\widehat{B}_{111} - \widehat{Q}_{133}\widehat{B}_{131} \cong -\widehat{B}_{121} \tag{30}$$

$$\widehat{E}_{31} = \widehat{Q}_{132}\widehat{B}_{111} + \widehat{Q}_{133}\widehat{B}_{121} \cong -\widehat{B}_{131} \tag{31}$$

$$\widehat{E}_{32} = \widehat{Q}_{132}\widehat{B}_{112} + \widehat{Q}_{133}\widehat{B}_{132} \cong -\widehat{B}_{132} \tag{32}$$

are chosen in our software implementation. The diagonal equations will also be used to make sure that the system converges to the correct solution. Six equations are the minimum number of equations necessary to get the system to converge. The effects of adding more equations on the stability, convergence rate, and speed of the algorithm will be discussed in section 4.3.

When all values of $\widehat{\theta}_i$ have converged, $\widehat{\mathbf{B}}_1$ becomes an identity matrix. This is true due to the fact that $\widehat{\mathbf{B}}_1$ in equation (14) equals equation (2) when all values of $\widehat{\theta}_i$ are exact values. The six dual equations, three from off-diagonal and three from diagonal elements, selected from dual matrix equation (12) or (19) give the following matrix equation in terms of the unknown quantities $d\widehat{\theta}_i$ for i from 2 to n.

$$\begin{bmatrix} \widehat{B}_{211} & \widehat{B}_{311} & \dots & \widehat{B}_{n11} \\ \widehat{B}_{221} & \widehat{B}_{321} & \dots & \widehat{B}_{n21} \\ \widehat{B}_{231} & \widehat{B}_{331} & \dots & \widehat{B}_{n31} \\ \widehat{B}_{222} & \widehat{B}_{322} & \dots & \widehat{B}_{n22} \\ \widehat{B}_{232} & \widehat{B}_{332} & \dots & \widehat{B}_{n32} \\ \widehat{B}_{233} & \widehat{B}_{333} & \dots & \widehat{B}_{n33} \end{bmatrix} \begin{bmatrix} d\widehat{\theta}_2 \\ d\widehat{\theta}_3 \\ . \\ . \\ . \\ d\widehat{\theta}_n \end{bmatrix} \cong \begin{bmatrix} 1 - \widehat{B}_{111} \\ -\widehat{B}_{121} \\ -\widehat{B}_{131} \\ 1 - \widehat{B}_{122} \\ -\widehat{B}_{132} \\ 1 - \widehat{B}_{133} \end{bmatrix} \tag{33}$$

This matrix equation can be expressed as

$$\widehat{\mathbf{M}}\widehat{\mathbf{d}} = \widehat{\mathbf{v}}, \tag{34}$$

where $\widehat{\mathbf{M}}$ is a 6x(n-1) dual matrix, $\widehat{\mathbf{d}}$ is a dual vector of (n-1) elements, and $\widehat{\mathbf{v}}$ is a dual vector of 6 elements.

3 NUMERICAL SOLUTION FOR DUAL FORMULATION

The dual matrix $\widehat{\mathbf{M}}$ of equation (34) is not necessarily a square matrix. This matrix equation cannot be solved by multiplying the inverse of $\widehat{\mathbf{M}}$ to each side of the equation. Therefore, it is necessary to use other techniques to find the solutions for $d\widehat{\theta}_i$ which minimize residual errors of the matrix equation. One quick way to solve a dual matrix equation is by using the dual least-squares method. The following is a derivation of the dual least-squares method for solving equation (34).

3.1 DUAL LEAST-SQUARES METHOD

Assuming $\widehat{\mathbf{M}}$ is an mxn dual matrix, the ith dual linear equation in matrix equation (34) can be expressed as the dual function

$$\widehat{F}_i = \sum_{j=1}^{n} \widehat{M}_{ij}\widehat{d}_j - \widehat{v}_i = 0, \tag{35}$$

for $i = 1$ to m, where n is the total number of unknowns and m is the total number of equations. The sum of the square of each residual from equation (35) must be taken to get the solution to equation (34) using

the dual least squares method. The error function is written as

$$\widehat{E} = \sum_{i=1}^{m} \widehat{F}_i^2. \tag{36}$$

The error function must be minimized with respect to n independent coefficients for m equations from i equal 1 to m. Equation (36) becomes

$$\begin{aligned} \widehat{E} &= \sum_{i=1}^{m} \left(\widehat{\mathbf{M}}_i^T \widehat{\mathbf{d}} - \widehat{v}_i \right)^T \left(\widehat{\mathbf{M}}_i^T \widehat{\mathbf{d}} - \widehat{v}_i \right) && (37) \\ &= \widehat{\mathbf{d}}^T \left(\sum_{i=1}^{m} \widehat{\mathbf{M}}_i \widehat{\mathbf{M}}_i^T \right) \widehat{\mathbf{d}} - 2\widehat{\mathbf{d}}^T \left(\sum_{i=1}^{m} \widehat{\mathbf{M}}_i \widehat{v}_i \right) + \sum_{i=1}^{m} \widehat{v}_i^2, && (38) \end{aligned}$$

where dual vector $\widehat{\mathbf{M}}_i$ is defined as

$$\widehat{\mathbf{M}}_i = (\widehat{M}_{i1}, \widehat{M}_{i2}, \cdots, \widehat{M}_{in})^T \tag{39}$$

and dual inner product $\widehat{\mathbf{M}}_i^T \widehat{\mathbf{d}}$ is defined as

$$\widehat{\mathbf{M}}_i^T \widehat{\mathbf{d}} = \sum_{j=1}^{n} \widehat{M}_{ij} \widehat{d}_j. \tag{40}$$

Define dual matrix $\widehat{\mathbf{N}}$ and dual vector $\widehat{\mathbf{u}}$ as

$$\widehat{\mathbf{N}} = \sum_{i=1}^{m} \widehat{\mathbf{M}}_i \widehat{\mathbf{M}}_i^T = \widehat{\mathbf{M}}^T \widehat{\mathbf{M}} \tag{41}$$

and

$$\widehat{\mathbf{u}} = \sum_{i=1}^{m} \widehat{\mathbf{M}}_i \widehat{v}_i = \widehat{\mathbf{M}}^T \widehat{\mathbf{v}} \tag{42}$$

respectively, and substituting these equations into equation (38) gives

$$\widehat{E} = \widehat{\mathbf{d}}^T \widehat{\mathbf{N}} \widehat{\mathbf{d}} - 2\widehat{\mathbf{d}}^T \widehat{\mathbf{u}} + \sum_{i=1}^{m} \widehat{v}_i^2. \tag{43}$$

The necessary condition for $\widehat{E}$ to be stationary is

$$\frac{\partial \widehat{E}}{\partial \widehat{\mathbf{d}}} = 0. \tag{44}$$

The derivatives of function $\widehat{E}$ with respect to $\widehat{d}_j$'s with j from 1 to n can be compactly organized as a dual vector as follows

$$\frac{\partial \widehat{E}}{\partial \widehat{\mathbf{d}}} = \left(\frac{\partial \widehat{E}}{\partial \widehat{d}_1}, \frac{\partial \widehat{E}}{\partial \widehat{d}_2}, \cdots, \frac{\partial \widehat{E}}{\partial \widehat{d}_n} \right)^T \tag{45}$$

After applying the stationary condition (44) to equation (43), the resultant equation gives

$$\widehat{\mathbf{N}} \widehat{\mathbf{d}} = \widehat{\mathbf{u}} \tag{46}$$

Alternatively, equation (46) can be written as the dual matrix equation

$$\widehat{\mathbf{M}}^T \widehat{\mathbf{M}} \widehat{\mathbf{d}} = \widehat{\mathbf{M}}^T \widehat{\mathbf{v}}, \tag{47}$$

where $\widehat{\mathbf{M}}$ is defined in equation (34). Equation (46) can be solved as a dual linear system. In summary, the above derivation shows that to solve equation (34) using the closest approximation of a solution in the least-square sense, we must premultiply the transpose of the $\widehat{\mathbf{M}}$ matrix to each side of the equation (34) to get equation (47).

3.2 SOLUTION OF A DUAL LINEAR SYSTEM OF EQUATIONS

The internal implementaion for solutions of a dual linear system (46) is described in this section. The unknown dual vector $\widehat{\mathbf{d}}$ in equation (46) can be determined by separating equation into its real and dual parts. For convenience of symbolic manipulations, a dual matrix topped with an accent grave "`" stands for the real part of the dual matrix and a dual matrix topped with an accent acute "´" stands for the dual part of the dual matrix. Let

$$\widehat{\mathbf{N}} = \grave{\mathbf{N}} + \varepsilon \acute{\mathbf{N}}, \quad \widehat{\mathbf{d}} = \grave{\mathbf{d}} + \varepsilon \acute{\mathbf{d}}, \quad \text{and} \quad \widehat{\mathbf{u}} = \grave{\mathbf{u}} + \varepsilon \acute{\mathbf{u}}, \tag{48}$$

equation (46) can be written as

$$\grave{\mathbf{N}}\grave{\mathbf{d}} + \varepsilon(\grave{\mathbf{N}}\acute{\mathbf{d}} + \acute{\mathbf{N}}\grave{\mathbf{d}}) = \grave{\mathbf{u}} + \varepsilon \acute{\mathbf{u}}. \tag{49}$$

The real part $\grave{\mathbf{d}}$ of dual vector $\widehat{\mathbf{d}}$ can be obtained by solving the following systems of linear equations

$$\grave{\mathbf{N}}\grave{\mathbf{d}} = \grave{\mathbf{u}}. \tag{50}$$

$$\grave{\mathbf{N}}\acute{\mathbf{d}} = \acute{\mathbf{u}} - \acute{\mathbf{N}}\grave{\mathbf{d}}. \tag{51}$$

After solving the real part $\grave{\mathbf{d}}$ in equation (50), the dual part $\acute{\mathbf{d}}$ can be obtained from the real matrix equation (51).

3.3 ITERATIVE SCHEME

To iteratively calculate a new joint position, initial guesses for $\overline{\overline{\theta}}_i$ are made. The correction terms $d\widehat{\theta}_i$ that are calculated using equations (50) and (51) are then added to $\overline{\overline{\theta}}_i$ according to equation (3). Using the updated $\widehat{\theta}_i$, the correction terms $d\widehat{\theta}_i$ will be computed again. This iterative process continues until the weighted sum δ of the error terms $d\widehat{\theta}_i$ is smaller than the preset convergence criterion. The weighted sum δ is defined as

$$\delta = \sum_{i=2}^{n} \left(l_i \left| d\theta_i \right| + \left| dd_i \right| \right), \tag{52}$$

where $d\theta_i$ and dd_i are real and dual parts of $d\widehat{\theta}_i$, respectively, and l_i is a normalizing factor. The value of l_i is set to 1 in our numerical implementation, To find the next position of the mechanism when the input angle θ_1 is incremented, the previous joint angles and displacements are used as initial guesses. In this paper, a convergence criterion of 0.00001 is used in all numerical examples.

3.4 SOFTWARE IMPLEMENTATION IN C^H

The above concise dual iterative formualtions can be conveniently implemented in the C^H programming language. The conciseness of the algorithm is retained in the C^H program as shown in Program 1. Program 1 can perform iterative displacement analysis of the RCCC spatial mechanism with specifications given as follows (Cheng, 1994): $\alpha_1 = 30^\circ$, $\alpha_2 = 55^\circ$, $\alpha_3 = 45^\circ$, $\alpha_4 = 60^\circ$, $a_1 = 2''$, $a_2 = 4''$, $a_3 = 3''$ $a_4 = 5''$, and $d_1 = 0$. The values of $\widehat{\theta}_2, \widehat{\theta}_3$, and $\widehat{\theta}_4$ are solved for each θ_1 ranging from 0 to 360° with an incremental step size of 90° degrees. For cross comparison, equation numbers for formulas derived in the previous sections are given as comments at the end of the corresponding programming statements in Program 1. The output from execution of Program 1 is given in Figure 2. The program will be further explained in section 4.1.

4 NUMERICAL EXAMPLES

To make comparison studies of stability, convergence rate, and speed of the dual iterative method, a program based upon the real iterative method (Uicker, et al., 1964) has also been coded in C^H. The error stability is defined as a continual reduction in error size after each iteration. And the convergence rate is defined as the amount of iterations necessary for convergence. In this section, we will first study the performance of both real and dual iterative algorithms with the minimum number of required equations. In section 4.3 we will then give numerical results with redundant equations. Unless specified otherwise in section 4.2 for solutions near singularities, all numerical results are obtained using float data type with 32 bits for a floating-point number. We have tested the efficacy of the dual iterative algorithm for displacement analysis of RCCC, RCRCR, and RSSR spatial mechanisms. For comparison purpose, only numerical experiments with RCCC mechanisms will be presented in this paper. Numerical examples with RCRCR and RSSR will be reported in the forthcoming MS thesis of the second author (Thompson, 1995).

4.1 COMPARISON WITH REAL ITERATION METHOD

We performed the displacement analysis using both real and dual iterative algorithms for an RCCC spatial mechanism with specifications given in section 3.4. There are two branches of solutions for a given input angle θ_1 for both dual and real iterative programs for θ_2, d_2, θ_3, d_3, θ_4, and d_4. In our numerical implementation, if the convergence criterion of the algorithm, defined as δ in equation (52) is less than a preset tolerance of 0.00001, the algorithm is considered to be convergent. If δ is larger than 100000, the algorithm is considered to be divergent as implemented in Program 1. To obtain solutions of a specific branch for both programs, the initial guesses for θ_i's must be sufficiently close to the solutions of that branch. If the initial guesses are not close to the solutions for a specific branch, the programs will still converge, but may not converge to the desired branch. For this RCCC spatial mechanism with two branches, only rough initial guesses to the desired solution are necessary to get convergence. For example, when the angular solutions for angles θ_2, θ_3, and θ_4 are in the range of -180° to 180°, the solutions in one branch of the RCCC mechanism lie between 0° and 180° at input angle $\theta_1 = 0^\circ$. To get a convergence to this branch, we chose 100° as the initial guess for all unknown angles θ_i's as implemented in Program 1. The solutions to the other branch are between 0° and -180° at input angle $\theta_1 = 0^\circ$. To get a convergence to this branch, we chose -100° as the initial guess for the unknown angles θ_i's. Experiment results show that the closeness of the initial guesses of output displacements d_2, d_3, and d_4 to the correct solutions for a desired branch is not an important factor in causing the program to converge to the desired branch. In fact, we can use zeros as the initial guesses of all unknown displacement values. After obtaining the first

TABLE 1: CONVERGENCE RATE AND ERROR δ WHEN $\theta = 20^\circ$ AND 40° USING DUAL ITERATIVE METHOD.

θ_1 (deg.)	Iteration number	$d\widehat{\theta}_2$ (radian, inch)	$d\widehat{\theta}_3$ (radian, inch)	$d\widehat{\theta}_4$ (radian, inch)	Error δ
20	1	(0.281170, 0.811668)	(0.000000, 0.000000)	(0.162334, 0.562341)	1.817514
	2	(-0.016173,-0.186685)	(0.062836, 0.195100)	(-0.034871,-0.247725)	0.743391
	3	(0.001186, 0.018256)	(-0.001494,-0.014047)	(0.001521, 0.020812)	0.057315
	4	(0.000001, 0.000030)	(-0.000001,-0.000014)	(0.000001, 0.000023)	0.000070
	5	(-0.000000, 0.000001)	(-0.000000,-0.000001)	(0.000000, 0.000001)	0.000003
40	1	(0.216086, 0.401169)	(0.109874, 0.299758)	(0.078998, 0.081556)	1.187442
	2	(-0.009430,-0.110008)	(0.052094, 0.068301)	(-0.031419,-0.159279)	0.430531
	3	(0.000677, 0.007224)	(-0.000700,-0.002550)	(0.000777, 0.007590)	0.019519
	4	(0.000000, 0.000004)	(-0.000000, 0.000001)	(0.000000, 0.000001)	0.000007
	5	(0.000000,-0.000000)	(0.000000,-0.000000)	(0.000000, 0.000000)	0.000001

set of solutions by the iterative process, the solutions are then used as initial guesses for the next new position as implemented in Program 1. For the above specifications of the RCCC mechanism, both dual and real iterative programs can converge to solutions on the same branch even for large increments of θ_1. For example, even if increments of 90° are used for θ_1, both methods can converge with solutions of the same branch they started with, as implemented in Program 1.

TABLE 2: CONVERGENCE RATE AND ERROR δ WHEN $\theta = 20^\circ$ AND 40° USING REAL ITERATION METHOD.

θ_1 (deg.)	Iteration number	$d\theta_2$ (radians)	dd_2 (inches)	$d\theta_3$ (radians)	dd_3 (inches)	$d\theta_4$ (radians)	dd_4 (inches)	Error δ
20	1	-0.811668	0.000011	-0.562342	-0.281169	-0.000003	-0.162331	1.655189
	2	0.188575	-0.196051	0.244952	0.016171	-0.062834	0.034869	0.645748
	3	-0.020153	0.014986	-0.018035	-0.001186	0.001494	-0.001521	0.054359
	4	-0.000024	0.000017	-0.000028	-0.000001	0.000001	-0.000001	0.000071
	5	0.000001	-0.000002	0.000002	0.000000	0.000000	-0.000000	0.000006
	6	-0.000001	0.000001	-0.000001	0.000000	-0.000000	-0.000000	0.000002
40	1	-0.401170	-0.299756	-0.081558	-0.216086	-0.109874	-0.078998	0.998571
	2	0.111895	-0.070407	0.158140	0.009430	-0.052093	0.031418	0.349872
	3	-0.009114	0.004656	-0.006449	-0.000677	0.000700	-0.000776	0.020895
	4	-0.000003	0.000000	-0.000004	-0.000000	0.000000	-0.000000	0.000007
	5	0.000001	-0.000002	0.000001	-0.000000	0.000000	0.000000	0.000004

In a numerical experimentation, input angle θ_1 varies from 0° to 360° with a step size of 20°. It has been observed that no more than five iterations are necessary for convergence of this iterative displacement analysis of the RCCC mechanism in both dual and real programs. Tables 1 and 2 give the convergence rates of the dual and real iterative programs, respectively. The numerical data in these tables show the typical convergence rates of the dual and real iterative algorithms when the input angle θ_1 is at 20° and 40° given an initial guess of 100° for θ_2, θ_3, and θ_4 and $0''$ for d_2, d_3 and d_4 at input angle $\theta_1 = 0^\circ$.

Numerical experiment results show that, although the rate of convergence of the dual iterative method is comparable with that of the real iterative method, the computational speed of the dual iterative method is faster. For example, for a complete RCCC displacement solution when θ_1 varies from 0 to 360° with an incremental step size of 20°, using zero as the initial guess for each of the unknown output displacements d_2, d_3, and d_4 and 100° for the initial guess for each of the unknown output angles θ_2, θ_3, and θ_4, the dual

TABLE 3: COMPARISON OF ERROR STABILITY OF REAL AND DUAL ITERATION METHODS WHEN $\theta_1 = 300^\circ$ USING VALUES AT $\theta_1 = 280^\circ$ AS INITIAL GUESSES WHEN USING FLOAT AND DOUBLE DATA TYPES.

Iteration number	Float data type Error δ		Double data type Error δ	
	Dual	Real	Dual	Real
1	3.498003	2.666386	3.498003	2.666387
2	2.322846	1.846846	2.322844	1.846858
3	0.797244	0.542110	0.797243	0.542095
4	0.286706	0.180110	0.286709	0.180144
5	0.138128	0.086053	0.138144	0.086099
6	0.068676	0.042861	0.068649	0.042849
7	0.034154	0.021059	0.034180	0.021340
8	0.016997	0.010573	0.016873	0.010523
9	0.008078	0.005234	0.008034	0.005007
10	0.003701	0.003164	0.003334	0.002087
11	0.017814	0.000543	0.000875	0.000555
12	0.005767	0.006010	0.000070	0.000045
13	0.045382	0.011300	0.000000	0.000000

iterative program takes an average of 15.81 seconds whereas the real iterative program takes an average of 34.47 seconds CPU time on a Motorola MVME167 VME board with a Motorola M68040 microprocessor of 25 MHz clock rate under a real-time Unix operating system. The speedup 2.18 of the dual iterative program over the real iterative program in C^H can be explained as follows. One of the reasons that the real iterative method uses more CPU time is due to the fact that the real iterative method uses more floating-point operations than the dual iterative method. In order to get displacement solutions of spatial mechanisms by an iterative numerical scheme, at least six dual equations out of the 3x3 dual matrix equation (29) are required. Similarly, at least nine equations out of twelve valid displacement equations are necessary for the real iterative method (Uicker, et al. 1964). The iterative program in the above example uses a 6x3 dual matrix $\widehat{\mathbf{M}}$, a 3-dimensional dual vector $\widehat{\mathbf{d}}$, and a 6-dimensional dual vector $\hat{\mathbf{v}}$ in equation (34). But, the real iterative program has a similar matrix equation containing a 9x6 real matrix $\mathbf{M}$, a 6-dimensional real vector $\mathbf{d}$, and a 9-dimensional real vector $\mathbf{v}$ (Uicker, et al., 1964). To solve the matrix equation using the least-squares method, a square matrix $\mathbf{M}^T\mathbf{M}$ is formed. This matrix multiplication takes 324 float multiplications and 288 float additions for the real iterative program. And the matrix multiplication of $\widehat{\mathbf{M}}^T\widehat{\mathbf{M}}$ in equation (47) takes 54 dual multiplications and 48 dual additions, which is equivalent to the total of 162 floating-point multiplications and 204 additions. The dual iterative program spends most CPU time on solving two linear systems of 3 equations (50) and (51) whereas the real iterative program spends most CPU time on solving a linear system of 6 equations. Another reason why the real iterative method uses more CPU time is the implementation of the dual data type in the C^H programming language. Built-in dual arithmetic operations and dual functions in C^H are very efficient. If conventional computer programming languages such as C and Fortran are used for implementation of the algorithms, one may observe less significant speedup because of the overhead in handling dual numbers. It should be mentioned that both real and dual iterative programs in C^H are implemented similarly.

TABLE 4: TOTAL ITERATIONS AND AVERAGE CPU TIMES USING DIFFERENT NUMBER OF EQUATIONS FOR BOTH DUAL AND REAL ITERATION METHODS.

Method	Number of equations	Total iterations	CPU time (seconds)
dual	6	84	15.81
dual	7	83	15.82
dual	8	84	16.36
dual	9	83	16.48
real	9	88	34.47
real	10	86	34.43
real	11	88	35.85
real	12	87	35.68

4.2 PERFORMANCE NEAR SINGULARITIES OF MECHANISMS

There is no singular position specified for the RCCC mechanism in the previous section. To test the rate of convergence and error stability of the real and dual iterative algorithms near a singularity position requires an alternate set of linkage parameters. We performed a displacement analysis for an RCCC spatial mechanism using the following linkage parameters (Sugimoto, et al., 1982): $\alpha_1 = 90^\circ$, $\alpha_2 = 60^\circ$, $\alpha_3 = 60^\circ$, $\alpha_4 = 90^\circ$, $a_1 = 2''$, $a_2 = 1.5''$, $a_3 = 1''$ $a_4 = 3''$, and $d_1 = 2.5''$. For this mechanism, the input crank will be stationary at $\theta_1 = 60^\circ$ and $\theta_1 = 300^\circ$, and the translational displacements will have infinite values at $\theta_1 = 180^\circ$. Graphical solutions to this RCCC spatial mechanism are shown in Figure 3. These graphs show that there are singularity positions at points A, B, and C corresponding to $\theta_1 = 60^\circ$, 180°, and 300°, respectively. These singularity positions are known to exist because at these positions there are two possible paths to take as the input angle θ_1 of the mechanism is incremented through its entire range of motion (Litvin, et al., 1986). The singularity at $\theta_1 = 180^\circ$ is also due to the fact that both displacements d_2 and d_4 go to infinity at this position. Therefore, it is not possible for either dual or real iterative programs to converge at $\theta_1 = 180^\circ$. Table 3 gives the performance of both real and dual iterative methods near a singularity at $\theta_1 = 300^\circ$ when values of $\theta_1 = 280^\circ$ are used as initial guesses. Columns 2 and 3 are the residual errors δ at different iteration steps for both dual and real iterative methods using float data types, respectively. Residual errors δ using double data types are given in columns 4 and 5 in Table 3. A dual double is represented by a double data type for both the real and dual parts of a dual number. As one can see, double data type has to be used and it takes more iteration steps to converge at a singular position. The residual error δ defined in equation (52) uniformly decreases to zero during the convergence process for both real and dual iterative programs. The singularity only causes a reduction in convergence rate. Both real and dual iterative algorithms show good stability at this singularity position.

4.3 ITERATIVE ALGORITHMS USING REDUNDANT EQUATIONS

Up to three additional equations could be used from equation (29) to form equation (34) in the dual iterative formulation. By experimentation, it was found that at non-singular mechanism positions only negligible differences in convergence rate and speed can be detected when additional equations are used to produce matrix equation (34). The experiment used the same convergence criterion of 0.00001. A range of 6 to 9 equations were used to form the matrix equation (34) for the dual iterative method and

a range of 9 to 12 equations were used for the real iterative method in this experiment. Table 4 gives total iteration numbers and average CPU times with different number of equations for both dual and real iterative methods when θ_1 of the RCCC mechanism described in section 4.1 is moved from 0 to 360° with a step size of 20°. Most of the CPU time is spent on solving two linear systems of 3 equations and one linear system of 6 equations for the dual and real iterative methods, respectively. Therefore, the CPU time of an iterative method is closely related to the total number of iterations as shown in Table 4. The total number of iterations of the dual iteration program is 84, 83, 84, and 83 when the number of equations used to form the dual matrix equation (34) is 6, 7, 8, and 9, respectively. These total number of iterations are comparable to each other, so are the corresponding total CPU times. This result holds true for the real iterative algorithm as well. A similar experiment was also conducted at the singularity position with θ_1 equal to 300° in the mechanism described in section 4.2. The experiment results showed that there is no significant difference in convergence rate when using more equations. It took 13 iteration steps to converge at each position for both real and dual algorithms regardless of the number of equations used.

5 CONCLUSIONS

Dual, dual double, and dual computational arrays are built into the C^H programming language, which significantly changes the way formulas for design and analysis of spatial mechanisms using dual numbers are formulated. Dual formulas no longer have to be painfully partitioned into separate real and dual parts for numerical calculation. A dual iterative method for displacement analysis of spatial mechanisms under the C^H programming paradigm has been developed in this paper. Because the 3x3 dual matrix is similar to the 3x3 pure rotational matrix of real elements, the dual iterative formulations are very concise. The conciseness of these dual equations and dual matrix equations can be retained in C^H programs. As shown in the paper, only one page of C^H code is needed for complete iterative displacement analysis of spatial mechanisms.

For comparison studies, a similar C^H program has also been developed for displacement analysis based upon the real iterative algorithm. Numerical results indicate that dual and real iterative algorithms are comparable in stability and convergence rate. Both algorithms can converge at mechanism singularity positions when double data type is used for floating-point calculations. But, the algorithms will fail at singularities if float data type is used. Iterative formulations are based upon infinitesimal displacements relations such as equation (29) for the dual iterative algorithm. Six out of nine dual equations are needed for the dual iterative formulation (three from diagonal part and three from the off-diagonal part) whereas nine out of twelve real equations for the real iterative formulation (three from the translational part, three from the diagonal part and three from the off-diagonal part of the rotational submatrix). It appears that using additional equations has little effect on convergence rate, stability, and CPU time of the programs, even near singularities. Numerical results also indicate that the dual iterative program presented in this paper is about 2.18 times faster than the corresponding real iterative program. Two primary factors contribute to this significant speedup. One reason is that the dual iterative program requires less floating-point operations than the real iterative program. For example, the dual iterative program only solves two smaller linear systems of equations whereas the real iterative program solves a large linear system of equations. The other reason of speedup is that, like real and complex numbers, dual number and dual arrays are efficiently implemented in the C^H programming language. The paper demonstrates that dual number is not only desirable for analytical treatment of spatial mechanisms, but also appealing for numerical computations.

6 ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under Grant DMII-9309207, by the University of California, Davis through a faculty research grant and a junior faculty research fellowship, and by Motorola, Inc.

7 REFERENCES

1. Cheng, H. H., Scientific Computing in the C^H Programming Language, *Scientific Programming*, vol. 2, No. 3, Fall 1993, pp. 49-75.

2. Cheng, H. H., Programming with Dual Numbers and its Applications in Mechanisms Design, *Engineering with Computers*, Vol. 10, No. 4, 1994, pp. 212-229.

3. Cheng, H. H., Extending C and FORTRAN for Design Automation, *ASME Trans, Journal of Mechanical Design*, Vol. 117, No. 3, 1995, pp. 390-395.

4. Cheng, H. H., and Thompson, S., Computer-Aided Displacement Analysis of Spatial Mechanisms Using the C^H Programming Language, *Advances in Engineering Software*, 1995b (in press).

5. Denavit, J., Displacement Analysis of Mechanisms Based on (2x2) Matrices of Dual Numbers, VDI-Berichte, Vol. 29, 1958, pp. 81-89.

6. Denavit, J. and Hartenberg, R. S., A Kinematic Notation for Lower Pair Mechanisms Based on Matrices, *ASME Trans. J. Applied Mech.*, vol. 22, 1955, pp. 215-221.

7. Fischer, I.S., Application of the Principle of Transference to the Evaluation of Translational Displacements in Spatial Mechanisms, *Proceedings of ASME 20th Biennial Mechanisms Conference*, Kissimmee, FL, Sept. 25-28, 1988, vol. 2, pp. 7-11.

8. Gupta, K. C., Spatial Kinematics. ME 409 Lecture Notes. Department of Mechanical Engineering, University of Illinois at Chicago, 1989.

9. Gupta, K. C., and Kazerounian, S. M. K., Improved Numerical Solutions of Inverse Kinematics of Robots, *International Conf. on Robotics and Automation*, March 25-27, 1985, St. Louis, MO., pp. 743-748;

10. Hamilton, W. R., *Elements of Quaternions*, 2nd ed., C. J. Jolly, ed., Vol. 1, 1899; Vol. 2, 1901, Longmans, Green and Co., London.

11. Litvin, F. L., Fanghella, P., Tan, J., and Zhang, Y., Singularities in Motion and Displacement Functions of Spatial Linkages. *ASME Trans., J. of Mechanisms, Transmissions, and Design Automation*, Vol. 108, 1986, pp. 516-523.

12. McCarthy, M., *Introduction to Theoretical Kinematics*, MIT Press, Cambridge, MA, 1990.

13. Pennock, G. R. and Yang, A. T., Application of Dual-Number Matrices to the Inverse Kinematics Problem of Robot Manipulators, *ASME Trans., J. of Mechanisms, Transmissions, and Design Automation*, Vol.107, June 1985, pp. 201-208.

14. Suh, C. H. and Radcliffe, C. W., *Kinematics and Mechanism Design*, John Wiley and Sons Inc., NY, 1978.

15. Sugimoto, K., Duffy J., and Hunt, K. H., Special Configurations of Spatial Mechanisms and Robot Arms. *Mechanism and Machine Theory*, Vol. 17, No 2, 1982, pp. 119-132.

16. Soni, A. H. and Harrisberger, L., Die Anwendung der 3x3 Schraubungs Matrix auf die kinematische und dynamische Analyze von raumlichen Getriben, V.D.I. *Berichte*, No. 127, 1968.

17. Thompson, S., Computer-Aided Displacement Analysis of Spatial Mechanisms Using the C^H Programming Language, MS thesis, Department of Mechanical and Aeronautical Engineering, UC Davis, 1995.

18. Uicker Jr., J. J., Denavit, J., Hartenberg, R. S. An Iterative Method for the Displacement Analysis of Spatial Mechanisms, *Journal of Applied Mechanics*, June 1964, pp. 309-314.

19. Yang, A. T. and Freudenstein, F., Application of Dual-Number Quaternion Algebra to the Analysis of Spatial Mechanisms, *ASME Journal of Applied Mechanics*, Vol. 31, ASME Trans., Vol. 86, Series E, No. 2, June 1964, pp. 300-308.

20. Yang, A. T., Displacement Analysis of Spatial Five-Link Mechanisms Using (3 X 3) Matrices With Dual-Number Elements, *Journal of Engineering for Industry*, Vol. 91 Feb. 1969, pp. 152-157.

21. Yuan, S. C., Displacement Analysis of the RCRCR Five-Link Spatial Mechanism, *Journal of Mechanisms*, Vol. 6, 1970, pp. 119-134.

8 APPENDIX

Prove that if $\hat{\mathbf{Q}}$ is dual antisymmetric and $\hat{\mathbf{H}}^T$ is the transpose of $\hat{\mathbf{H}}$, then the product

$$\hat{\mathbf{L}} = \hat{\mathbf{H}}\hat{\mathbf{Q}}\hat{\mathbf{H}}^T \tag{53}$$

is also dual antisymmetric. The definition of a dual antisymmetric matrix is $\hat{Q}_{ij} = -\hat{Q}_{ji}$ or $-\hat{Q} = \hat{Q}^T$ which implies $\hat{Q}_{ii} = 0$.

Let $\hat{\mathbf{Q}} = [\hat{Q}_{ij}]$, $\hat{\mathbf{H}} = [\hat{H}_{ij}]$, $\hat{\mathbf{L}} = [\hat{L}_{ij}]$, and $\hat{\mathbf{H}}^T = [\hat{H}_{ij}]^T = [\hat{H}_{ji}]$, where the subscripts ij indicate the element location (i,j). For clarity, define dual matrix $\hat{\mathbf{B}}$ with element $\hat{B}_{ij}$ as

$$\hat{B}_{ij} = \sum_{k=1}^{n} \hat{H}_{ik}\hat{Q}_{kj}. \tag{54}$$

Element $\hat{L}_{ij}$ in equation (53) can be written as

$$\begin{aligned} \hat{L}_{ij} &= \sum_{l=1}^{n} \hat{B}_{il}\hat{H}_{lj}^T \\ &= \sum_{l=1}^{n} \left(\sum_{k=1}^{n} \hat{H}_{ik}\hat{Q}_{kl} \right) \hat{H}_{jl} \end{aligned} \tag{55}$$

From equation (55), we can get

$$\widehat{L}_{ji} = \sum_{l=1}^{n} \left(\sum_{k=1}^{n} \widehat{H}_{jk} \widehat{Q}_{kl} \right) \widehat{H}_{il} \tag{56}$$

After substituting $\widehat{Q}_{kl} = -\widehat{Q}_{lk}$ into equation (56), we get

$$\widehat{L}_{ji} = \sum_{l=1}^{n} \left(\sum_{k=1}^{n} \widehat{H}_{jk} (-\widehat{Q}_{lk}) \right) \widehat{H}_{il}. \tag{57}$$

This equation can be rearranged as

$$\widehat{L}_{ji} = -\sum_{k=1}^{n} \left(\sum_{l=1}^{n} \widehat{H}_{il} \widehat{Q}_{lk} \right) \widehat{H}_{jk}. \tag{58}$$

After switching variables l and k, equation (58) becomes

$$\widehat{L}_{ji} = -\sum_{l=1}^{n} \left(\sum_{k=1}^{n} \widehat{H}_{ik} \widehat{Q}_{kl} \right) \widehat{H}_{jl}. \tag{59}$$

Equations (55) and (59) indicate that

$$\widehat{L}_{ji} = -\widehat{L}_{ij}. \tag{60}$$

PROGRAM 1: A C^H program for dual iterative displacement analysis of an RCCC mechanism

```
/* A dual iterative program in CH for RCCC mechanism */
#include <linkage.h>     /* PI is defined in <linkage.h> */
#define D2R PI/180
#define R2D 180/PI
main(){
  int i, j, theta1, n=4, m=6;
  float error, epsilon=0.00001, MaxError=100000;
  dual theta[n], alpha[n];
  array dual Q[3][3],Bi[3][3],B1[3][3],Ai[3][3],M[m][3],d[3],v[m],N[3][3],u[3];
  void ai(dual theta, alpha, array dual Ai[3][3]){    /* calculates Ai matrix */
    Ai[0][0] = cos(theta);            Ai[0][1] = -sin(theta)*cos(alpha);
    Ai[0][2] = sin(theta)*sin(alpha); Ai[1][0] = sin(theta);
    Ai[1][1] = cos(theta)*cos(alpha); Ai[1][2] = -cos(theta)*sin(alpha);
    Ai[2][0] = 0; Ai[2][1] = sin(alpha); Ai[2][2] = cos(alpha);     /*(1) */
  }

  alpha[0]=dual(30*D2R,2);  alpha[1]=dual(55*D2R,4);
  alpha[2]=dual(45*D2R,3);  alpha[3]=dual(60*D2R,5);  /* link parameters */
  theta[0]=dual(0,0);       theta[1]=dual(100*D2R,0);
  theta[2]=dual(100*D2R,0); theta[3]=dual(100*D2R,0); /* initial values */
  printf("  theta1     s1    theta2      s2     "
         "theta3     s3    theta4      s4\n");        /* print heading */
  Q = 0; Q[0][1] = -1; Q[1][0] = 1,0;                              /*(9) */
  for(theta1=0;theta1<=360;theta1+=90){        /* 90 deg step for theta1 */
    theta[0] = theta1*D2R;                     /* change deg to rad */
    do{
      for (i=1;i<n;i++){
        for (Bi=0,j=0;j<=2;j++) Bi[j][j] = 1;
        for (j=0;j<=i-1;j++,Bi*=Ai) ai(theta[j],alpha[j],Ai);       /*(16)*/
        for (Bi*=Q,j=i;j<n;j++,Bi*=Ai) ai(theta[j],alpha[j],Ai);    /*(16)*/
        M[0][i-1]=Bi[0][0]; M[1][i-1]=Bi[1][1]; M[2][i-1]=Bi[2][2]; /*(33)*/
        M[3][i-1]=Bi[1][0]; M[4][i-1]=Bi[2][0]; M[5][i-1]=Bi[2][1]; /*(34)*/
      }
      ai(theta[0],alpha[0],B1);                                     /*(14)*/
      for (i=1;i<n;i++,B1*=Ai) ai(theta[i],alpha[i],Ai);            /*(14)*/
      v[0]=1-B1[0][0];  v[1]=1-B1[1][1];  v[2]=1-B1[2][2];          /*(33)*/
      v[3]= -B1[1][0];  v[4]= -B1[2][0];  v[5]= -B1[2][1];          /*(34)*/
      N=transpose(M)*M;                                             /*(41)*/
      u=transpose(M)*v;                                             /*(42)*/
      real(d) = linearsolver(real(N), real(u));                     /*(50)*/
      dual(d) = linearsolver(real(N), dual(u) - dual(N)*real(d));   /*(51)*/
      for (error=0,i=0;i<n-1;i++)
          error += abs(real(d[i]))+abs(dual(d[i]));                 /*(52)*/
      for (i=1;i<n;i++) theta[i] += d[i-1]; /* update theta */      /*(3) */
    }while (error >= epsilon && error < MaxError);
    if(error >= MaxError)                  /* diverged */
      printf("Error: the iterative algorithm diverges at theta1 = %d \n", theta1);
    else                                   /* converged, print results */
      printf("%8.2f %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f\n",
      real(theta[0])*R2D,dual(theta[0]),real(theta[1])*R2D,dual(theta[1]),
      real(theta[2])*R2D,dual(theta[2]),real(theta[3])*R2D,dual(theta[3]));
  }
}
```

theta1	s1	theta2	s2	theta3	s3	theta4	s4
0.00	0.000	149.680	-0.210	45.556	-2.693	144.209	-0.115
90.00	0.000	54.512	-3.171	92.715	-1.513	81.114	-2.114
180.00	0.000	-59.093	-0.301	142.649	-1.814	83.700	-0.173
270.00	0.000	-157.692	1.136	92.715	-1.513	148.494	-0.515
360.00	0.000	-210.320	-0.210	45.556	-2.693	144.209	-0.115

FIGURE 2: The output from the dual iterative program.

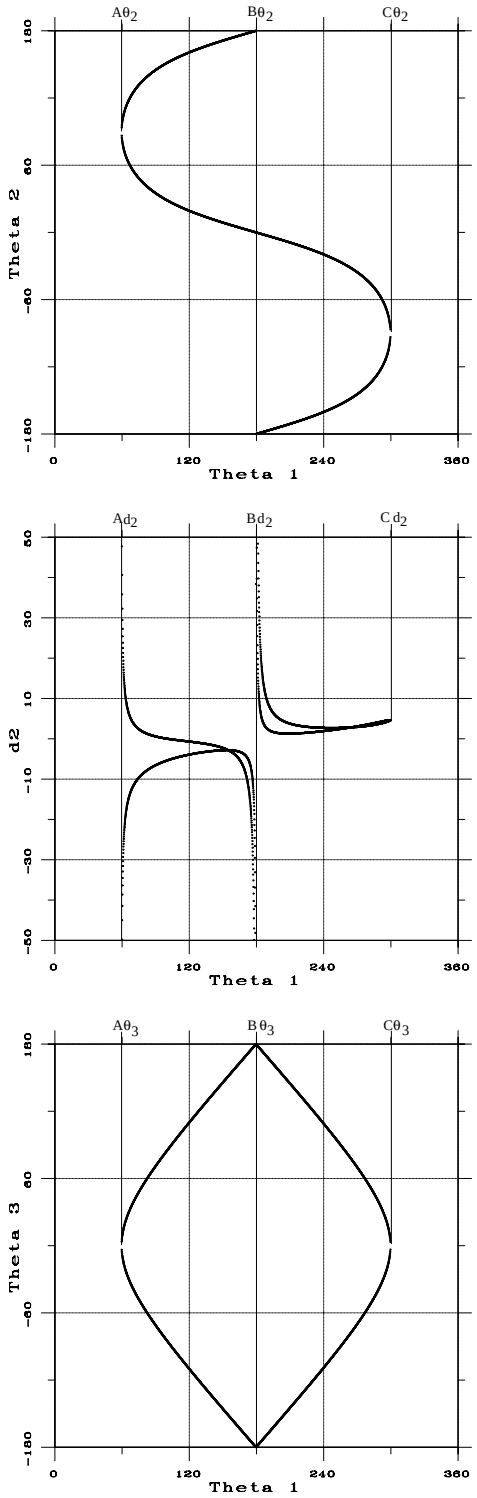

FIGURE 3: GRAPHICAL OUTPUT FOR JOINT VARIABLES θ_2, d_2, and θ_3 VERSUS INPUT θ_1 OF AN RCCC MECHANISM WITH SINGULARITIES.