A Fast Rotation Estimator in Geometric Algebra

Mauricio Cele Lopez Belon

Abstract. We present a fast estimator of the best rotation aligning two sets of corresponding vectors (also known as Wahba's problem). The proposed method is among the fastest methods reported in literature, moreover it is robust to noise, accurate and simpler than most other methods. It is based on solving the linear equations derived from the formulation of the problem in Geometric Algebra.

Mathematics Subject Classification (2010). Parallel algorithms 68W10; Clifford algebras, spinors 15A66.

Keywords. Geometric Algebra, Rotation Estimation, Wahba Problem.

1. Introduction

Wahba's problem has been studied for over half a century since 1965 [10]. The problem looks for the optimal rotation between two sets of corresponding vectors. Many effective algorithms have been developed to solve the problem [1, 3, 4, 7, 11, 12] using quaternions, the Singular Value Decomposition (SVD) and recently geometric algebra representation [2, 6]. Applications of Wahba's solutions are diverse from aerospace engineering computation of spacecraft attitude [11] to mesh deformation in computer graphics [8, 9] for accurate motion construction and restoration [5, 6].

In engineering fields for which the accuracy requirements are not high the computation time can be lowered down in more than 50% by resigning a little bit of accuracy. In this work we propose a fast, robust to noise, simple and accurate method for solving the Wahba's problem. It is based on minimizing a convex least squares error formulated in geometric algebra $\mathbb{G}_3$. The proposed method can be applied for aligning vectors and bivectors alike.

Few work has been done on studying this problem using geometric algebra. Geometric algebra rotors are closely related to quaternions (quaternion algebra can be regarded as a geometric algebra defined on a set of imaginary

basis vectors) we find geometric algebra to be a more natural choice for studying this problem since it is defined over a Euclidean vector space $\mathbb{R}^3$, where original data is defined. We derive our algorithm from Wahba's problem defined on bivectos instead of vectors for the sake of mathematical simplicity. By duality our formulation is valid also for vectors. The implementation of our algorithm does not require access to any geometric algebra library, we present an implementation based on standard matrix and quaternion library.

This paper is arranged as follows: Section II introduces the geometric algebra $\mathbb{G}_3$, Section III includes the presentation of our fast rotor estimation algorithm. Section IV demonstrates the experimental results.

2. Geometric Algebra $\mathbb{G}_3$

A geometric algebra $\mathbb{G}_3$ is constructed over a real vector space $\mathbb{R}^3$, with basis vectors $\{e_1, e_2, e_3\}$. The associative geometric product is defined so that the square of any vector is a scalar $aa = a^2 \in \mathbb{R}$. From the vector space $\mathbb{R}^3$, the geometric product generates the geometric algebra $\mathbb{G}_3$ with elements $\{X, R, A...\}$ called multivectors.

For a pair of vectors, a symmetric inner product $a \cdot b = b \cdot a$ and antisymmetric outer product $a \wedge b = -b \wedge a$ can be defined implicitly by the geometric product $ab = a \cdot b + a \wedge b$ and $ba = b \cdot a + b \wedge a$. It is easy to prove that $a \cdot b = \frac{1}{2}(ab + ba)$ is scalar, while the quantity $a \wedge b = \frac{1}{2}(ab - ba)$, called a bivector or 2-vector, is a new algebraic entity that can be visualized as the two-dimensional analogue of a direction, that is, a planar direction. Similar to vectors, bivectors can be decomposed in a bivector basis $\{e_{12}, e_{13}, e_{23}\}$ where $e_{ij} = e_i \wedge e_j$.

The outer product of three vectors $a \wedge b \wedge c$ generates a 3-vector also known as the pseudoscalar, because the trivector basis consist of single element $e_{123} = e_1 \wedge e_2 \wedge e_3$. Similarly, the scalars are regarded as 0-vectors whose basis is the number 1. It follows that the outer product of k-vectors is the completely antisymmetric part of their geometric product: $a_1 \wedge a_2 \wedge ... \wedge a_k = \langle a_1 a_2 ... a_k \rangle_k$ where the angle bracket means k-vector part, and k is its grade. The term grade is used to refer to the number of vectors in any exterior product. This product vanishes if and only if the vectors are linearly dependent. Consequently, the maximal grade for nonzero k-vectors is 3. It follows that every multivector X can be expanded into its k-vector parts and the entire algebra can be decomposed into k-vector subspaces:

$$\mathbb{G}_3 = \sum_{k=0}^{n} \mathbb{G}_3^k = \{X = \sum_{k=0}^{n} \langle X \rangle_k\}$$

This is called a *grading* of the algebra.

Reversing the order of multiplication is called reversion, as expressed by $(a_1 a_2 ... a_k)\tilde{} = a_k ... a_2 a_1$ and $(a_1 \wedge a_2 \wedge ... \wedge a_k)\tilde{} = a_k \wedge ... \wedge a_2 \wedge a_1$, and the reverse of an arbitrary multivector is defined by $\tilde{X} = \sum_{k=0}^{n} \langle \tilde{X} \rangle_k$.

Rotations are even grade multivectors known as rotors. We denote the subalgebra of rotors as $\mathbb{G}_3^+$. A rotor R can be generated as the geometric product of an even number of vectors. A reflection of any k-vector X in a plane with normal n is expressed as the sandwitch product $(-1)^k nXn$. The most basic rotor R is defined as the product of two unit vectors a and b with angle of $\frac{\theta}{2}$. The rotation plane is the bivector $B = \frac{a \wedge b}{\|a \wedge b\|}$.

$$ab = a \cdot b + a \wedge b = \cos\left(\frac{\theta}{2}\right) + B \sin\left(\frac{\theta}{2}\right).$$

Rotors act on all k-vectors using the sandwitch product $X' = RX\tilde{R}$, where $\tilde{R}$ is the reverse of R and can be obtained by reversing the order of all the products of vectors.

3. Geometric Algebra Rotor Estimation

Given two sets of n corresponding bivectors $P = \{p_j\}_{j=1}^n$ and $Q = \{q_j\}_{j=1}^n$, we attempt to minimize the following error function:

$$E(R) = \min_{R \in \mathbb{G}_3^+} \sum_j c_j \|q_j - R p_i \tilde{R}\|^2$$
$$s.t.\ R\tilde{R} = 1$$

where $\{c_j\}_{j=1}^n$ are scalar weights such that $\sum_j^n c_j = 1$. In that form, the minimization is a non-linear least squares problem with a non-linear constraint in R. Notice that the error term $q_j - Rp_i\tilde{R}$ is equivalent to $q_j R - Rp_j$ by multiplying by R on the right and using the fact that $R\tilde{R} = 1$. The equivalent problem is:

$$E(R) = \min_{R \in \mathbb{G}_3^+} \sum_j c_j \|Rp_j - q_j R\|^2$$
$$s.t.\ R\tilde{R} = 1$$

which is a linear least squares problem with a non-linear constraint in R.

Let $\mathcal{K} : \mathbb{G}_3^+ \mapsto \mathbb{R}^4$ be a function that maps a rotor to a 4×1 column matrix.

$$\mathcal{K}(R) = \begin{bmatrix} \langle R \rangle_0 \\ \langle R \rangle_{12} \\ \langle R \rangle_{13} \\ \langle R \rangle_{23} \end{bmatrix}$$

Let define the function F^j representing the linear transformation:

$$F^j = \mathcal{K}(\sqrt{c_j}(Rp_j - q_j R))$$
$$F^j = \sqrt{c_j} M_j R$$

with M_j is defined as the following skew-symmetric 4×4 real matrix:

$$M_j = \begin{bmatrix} 0 & \langle D \rangle_{12} & \langle D \rangle_{13} & \langle D \rangle_{23} \\ -\langle D \rangle_{12} & 0 & -\langle S \rangle_{23} & \langle S \rangle_{13} \\ -\langle D \rangle_{13} & \langle S \rangle_{23} & 0 & -\langle S \rangle_{12} \\ -\langle D \rangle_{23} & -\langle S \rangle_{13} & \langle S \rangle_{12} & 0 \end{bmatrix}$$

$$D = q_j - p_j \quad S = p_j + q_j$$

For simplicity of notation we denote the rotor $R \in \mathbb{G}_3^+$ and vector $\mathcal{K}(R) \in \mathbb{R}^4$ with letter R. Note that $M_j^T = -M_j$ as M_j is representing the non-commutativity of the geometric product as well as the antisymmetry of bivectors.

Let F be a column vector of n stacked functions F^j

$$F = \begin{bmatrix} F^1 \\ \vdots \\ F^n \end{bmatrix}$$

such that the energy $E(R)$ can be expressed as matrix product

$$E(R) = F^T F = \begin{bmatrix} F^{1T} & \cdots & F^{nT} \end{bmatrix} \begin{bmatrix} F^1 \\ \vdots \\ F^n \end{bmatrix}$$

We can express $E(R)$ as the following quadratic form:

$$E(R) = \min_R R^T M R$$
$$s.t.\ R^T R = 1$$

where $M = \sum_j^n c_j M_j^T M_j$. Note that since M_j is skew-symmetric, the product $M_j^T M_j$ is symmetric and positive semi-definite. Consequently the matrix M is also symmetric positive semi-definite. It follows that all eigenvalues of M are real and $\lambda_i \geq 0$.

Using the spectral theorem it is easy to show that the minimizer of $E(R)$ is the eigenvector of M associated with the minimum eigenvalue. Typically it can be obtained with great precision using SVD or QR factorization. Nevertheless, in this work we seek for an efficient solution which allow us to take advantage of nowadays computing hardware. In that regard, the *inverse* power iteration is a more desirable algorithm since we can leverage CPU and GPU vectorization.

As we are looking for the eigenvector associated with minimum eigenvalue the cases having $\lambda_{\min} = 0$ are problematic. The reason for having $\lambda_{\min} = 0$ is that the energy $E(R)$ is non-convex, its shape is similar to a sinusoidal wave with infinitely many points at minimum energy value, each one at rotor $R_i = e^{-(\theta + i\pi)B}$ for $i \in \mathbb{N}$, being B the optimal unique attitude bivector and θ an optimal angle with minimal absolute value. That means that the solution is in general non-unique, in which case H is a singular matrix.

Let us analyze the critical points of the energy $E(R)$. The minimum is attained when the gradient vanishes $\nabla E(R) = 0$. The gradient has the following form:

$$\begin{aligned} g &= \nabla E(R) \\ g &= \nabla (F^T F) \\ g &= 2J^T F \end{aligned}$$

where J is the Jacobian matrix of F. Since the energy $E(R)$ is purely quadratic in R, the solution for $\nabla E(R) = 0$ can be found by solving a linear system of equations. An optimal rotor is in the null space of matrix $J^T F$. Naturally, a solution in the null space of $J^T F$ can be approximated using the Newton's method. Due to linearity of the equation $(J^T F)R = 0$ w.r.t. R the optimal increment for $E(R)$ is given by the Newton formula $\Delta R = H^{-1}(J^T F)R$, provided that the inverse of Hessian matrix H^{-1} exists. Noting that since J does not depend on R, i.e. is constant, the Hessian matrix H takes the simple form:

$$\begin{aligned} H &= \frac{\partial (2J^T F R)}{\partial R} \\ H &= 2J^T \frac{\partial F R}{\partial R} \\ H &= 2J^T J \end{aligned}$$

Where H is independent of R. It seem that an optimal solution can be approximated by solving a series of linear systems.

$$R_{i+1} = R_i - H^{-1}(J^T F)R_i$$

However, even when H is non-singular, the Newton iteration introduced above is not possible. Note that $J^j = M_j$ since:

$$\begin{aligned} J^j &= \frac{\partial (F^j R)}{\partial R} \\ J^j &= \frac{\partial (M_j R)}{\partial R} \\ J^j &= M_j \end{aligned}$$

replacing in Newton's formula we get

$$\begin{aligned} R_{i+1} &= R_i - H^{-1}(J^T F)R_i \\ R_{i+1} &= R_i - (J^T J)^{-1}(J^T J)R_i \end{aligned}$$

which is a consequence of the non-convexity of $E(R)$.

One simple solution is to use the Tikhonov regularization on H to approximate a pseudo-inverse $(H + \epsilon I)^{-1}$. We prefer Tikhonov regularization over the Moore-Penrose pseudo-inverse because Tikhonov regularization makes H symmetric positive definite matrix, making the energy $E(R)$ convex. An optimal solution can then be approximated by solving the following

series of linear systems.

$$R_{i+1} = R_i - (H + \epsilon I)^{-1} H R_i$$

until convergence is reached in the sense that $|R_i - R_{i+1}|_1 < \xi$ for a small ξ. Where we use $R_0 = [1\ 0\ 0\ 0]^T$. This algorithm has a number of advantages. Since pseudo-inverse Hessian $(H + \epsilon I)^{-1}$ is constant, it can be precomputed. Also the Hessian matrix H is constant, so it can be precomputed as well. Although the computation is relatively cheap and this approach is fast, we can design a faster algorithm by adding a new regularization term to $R(R)$.

4. Fast Geometric Algebra Rotor Estimation

The non-convex energy $E(R)$ can be made convex by adding a new term to it. We introduce a new rotor R_i within a new function $D(R, R_i) = \sqrt{\epsilon}(R - R_i)$. The new term can be expressed as matrix product:

$$D^T D = \epsilon \|R - R_i\|^2$$

with Jacobian

$$J_D = \frac{\partial D}{\partial R} = \sqrt{\epsilon} I$$

where I is the 4×4 identity matrix. The convex energy now looks like this:

$$E_2(R, R_i) = F^T F + D^T D$$
$$E_2(R, R_i) = \sum_j c_j \|R p_j - q_j R\|^2 + \epsilon \|R - R_i\|^2$$

Where $0 < \epsilon < 1$ is constant but small. The regularization term $D^T D = \epsilon \|R - R_i\|^2$ helps on finding a solution *shifted* by a small amount towards the direction $\delta D = \epsilon(R - R_i)$, so one can interpret the rotor R_i as a target of displacement. Also note that using $R^T R = 1$ and $R_i^T R_i = 1$ we get $D^T D = \epsilon(1 - R^T R_i)$ which can be seen as a soft normalization constraint.

The regularization term is perturbing the gradient and Hessian in the following way:

$$g_2 = \nabla(F^T F + D^T D)$$
$$g_2 = \frac{\partial (F^T F)}{\partial R} + \frac{\partial (D^T D)}{\partial R}$$
$$g_2 = 2(J^T F + J_D^T D)$$
$$g_2 = 2(J^T J + \epsilon D)$$
$$g_2 = 2((H + \epsilon I)R - \epsilon R_i)$$

$$H_2 = 2\frac{\partial(J^T F + \sqrt{\epsilon}D)}{\partial R}$$
$$H_2 = 2J^T\frac{\partial(F)}{\partial R} + 2\sqrt{\epsilon}\frac{\partial(D)}{\partial R}$$
$$H_2 = 2(J^T J + \epsilon I)$$
$$H_2 = 2(H + \epsilon I)$$

It is useful to express g_2 also as $g_2 = (H+\epsilon I)R - R_i$ and H_2 as $H_2 = H + \epsilon I$. Replacing the above definitions in Newton's formula we get:

$$R_{i+1} = R_0 - (H + \epsilon I)^{-1}((H + \epsilon I)R_0 - \epsilon R_i)$$
$$R_{i+1} = \epsilon(H + \epsilon I)^{-1} R_i$$

With this perturbation on the Hessian, the Tikhonov regularization emerges by itself. The perturbation on the gradient $g(R_0)$ make it disappear, cancelling it out with the initial guess, forcing R_{i+1} to align towards the eigenvector of $(H + \epsilon I)^{-1}$ corresponding with the maximum eigenvalue. This is in fact a version of the inverse power iteration algorithm, which happen to coincide with Newton's method for minimizing a convex energy functional.

Newton's method convergence rate is quadratic. However our method's convergence rate is linear or *geometric*, depending on how the ratio of the largest eigenvalue to the second largest eigenvalue $\lambda_2/\lambda_{\max}$ is close to zero i.e., the method converges slowly if there is an eigenvalue close in magnitude to the dominant eigenvalue.

Several mathematicians have proved bounds on eigenvalues in terms of the value of the matrix. A simple test by Alfred Brauer, 1946, is based on the absolute values of the elements. Let a_{ij} be the elements, $R_i = \sum_j \|a_{ij}\|$ be the sum of absolute values along the ith row and $C_j = \sum_i \|a_{ij}\|$ the sum down the jth column. Let $\mathcal{R}$ be the largest R_i and $\mathcal{C}$ the largest C_j. Then for all eigenvalues $|\lambda| \leq \min(\mathcal{R}, \mathcal{C})$.

5. Optimal Computation of Jacobians

The differential of F^j defined as $J^j = \frac{\partial F^j}{\partial R}$ is a 4×4 matrix which four columns are the directional derivatives of $F^j = \sqrt{c_j}(Rp_j - q_j R)$ w.r.t rotor components on the basis $\mathbb{G}_3^+$:

$$J^j = \begin{bmatrix} \frac{\partial F^j}{\partial w} & \frac{\partial F^j}{\partial e_{12}} & \frac{\partial F^j}{\partial e_{13}} & \frac{\partial F^j}{\partial e_{23}} \end{bmatrix}$$

$$\frac{\partial F^j}{\partial w} = \sqrt{c_j}(p_j - q_j)$$
$$\frac{\partial F^j}{\partial w} = \sqrt{c_j}\begin{bmatrix} (p_j - q_j)\cdot e_1 \\ (p_j - q_j)\cdot e_2 \\ (p_j - q_j)\cdot e_3 \\ 0 \end{bmatrix}$$

$$\frac{\partial F^j}{\partial e_{12}} = \sqrt{c_j}(-(p_j+q_j)\cdot e_{12} + (p_j-q_j)\wedge e_{12})$$

$$\frac{\partial F^j}{\partial e_{12}} = \sqrt{c_j}\begin{bmatrix} (p_j+q_j)\cdot e_2 \\ -(p_j+q_j)\cdot e_1 \\ 0 \\ (p_j-q_j)\cdot e_3 \end{bmatrix}$$

$$\frac{\partial F^j}{\partial e_{13}} = \sqrt{c_j}(-(p_j+q_j)\cdot e_{13} + (p_j-q_j)\wedge e_{13})$$

$$\frac{\partial F^j}{\partial e_{13}} = \sqrt{c_j}\begin{bmatrix} (p_j+q_j)\cdot e_3 \\ 0 \\ -(p_j+q_j)\cdot e_1 \\ -(p_j-q_j)\cdot e_2 \end{bmatrix}$$

$$\frac{\partial F^j}{\partial e_{23}} = \sqrt{c_j}(-(p_j+q_j)\cdot e_{23} + (p_j-q_j)\wedge e_{23})$$

$$\frac{\partial F^j}{\partial e_{23}} = \sqrt{c_j}\begin{bmatrix} 0 \\ (p_j+q_j)\cdot e_3 \\ -(p_j+q_j)\cdot e_2 \\ (p_j-q_j)\cdot e_1 \end{bmatrix}$$

So the full expression of J^j is:

$$J^j = \sqrt{c_j}\begin{bmatrix} (p_j-q_j)\cdot e_1 & (p_j+q_j)\cdot e_2 & (p_j+q_j)\cdot e_3 & 0 \\ (p_j-q_j)\cdot e_2 & -(p_j+q_j)\cdot e_1 & 0 & (p_j+q_j)\cdot e_3 \\ (p_j-q_j)\cdot e_3 & 0 & -(p_j+q_j)\cdot e_1 & -(p_j+q_j)\cdot e_2 \\ 0 & (p_j-q_j)\cdot e_3 & -(p_j-q_j)\cdot e_2 & (p_j-q_j)\cdot e_1 \end{bmatrix}$$

The gradient of $E(R)$ amounts to $g = \sum_j J^{jT}F^j$ and the symmetric matrix $H^j = J^{jT}J^j$ has a simple form:

$$S_1 = (q_j+p_j)\cdot e_1\ ,\ S_2 = (q_j+p_j)\cdot e_2\ ,\ S_3 = (q_j+p_j)\cdot e_3$$
$$D_1 = (p_j-q_j)\cdot e_1\ ,\ D_2 = (p_j-q_j)\cdot e_2\ ,\ D_3 = (p_j-q_j)\cdot e_3$$

$$H^j = J^{jT}J^j = c_{ij}\begin{bmatrix} D_1^2+D_2^2+D_3^2 & D_1S_2-D_2S_1 & D_1S_3-D_3S_1 & D_2S_3-D_3S_2 \\ D_1S_2-D_2S_1 & S_2^2+S_1^2+D_3^2 & S_2S_3-D_3D_2 & D_3D_1-S_1S_3 \\ D_1S_3-D_3S_1 & S_2S_3-D_3D_2 & S_3^2+S_1^2+D_2^2 & S_1S_2-D_2D_1 \\ D_2S_3-D_3S_2 & D_3D_1-S_1S_3 & S_1S_2-D_2D_1 & S_3^2+S_2^2+D_1^2 \end{bmatrix}$$

Since H^j is symmetric only 10 out of 16 elements need to be actually computed. The Hessian of $E(R)$ amounts to $H = \sum_j H^j$ and is constant.

We now can proceed to optimize our method. We incorporate the fixed initial guess $R_0 = 1$ into the gradient $g = \sum_j g^j = J^T F$.

$$F^j = \sqrt{c_j}(R_0 p_j - q_j R_0)$$
$$F^j = \sqrt{c_j}(p_j - q_j)$$

$$F^j = \sqrt{c_{ij}} \begin{bmatrix} (p_j - q_j) \cdot e_1 \\ (p_j - q_j) \cdot e_2 \\ (p_j - q_j) \cdot e_3 \\ 0 \end{bmatrix}$$

$$g^j = J^{jT} F^j = c_{ij} \begin{bmatrix} D_1^2 + D_2^2 + D_3^2 \\ D_1 S_2 - D_2 S_1 \\ D_1 S_3 - D_3 S_1 \\ D_2 S_3 - D_3 S_2 \end{bmatrix}$$

Notice that, with this replacement, the gradient $g = \sum_j g^j$ is constant. Notice also that the first column of H^j is equal to g^j, so $g = \sum_j g^j$ does not need to be explicitly calculated. Now we proceed to replace R_0 on the regularization term D, which look like this:

$$D = \sqrt{\epsilon}(R_0 - R_i)$$
$$D = \sqrt{\epsilon}(1 - R_i)$$

$$\sqrt{\epsilon} D = J_D^T D = \epsilon \begin{bmatrix} 1 - \langle R_i \rangle_0 \\ -R_i \cdot e_{12} \\ -R_i \cdot e_{13} \\ -R_i \cdot e_{23} \end{bmatrix}$$

With the above replacements, notice that all terms are constant except D which depends only on the previous iteration.:

$$R_{i+1} = R_0 - (H + \epsilon I)^{-1}(g + \sqrt{\epsilon} D(R_i))$$

the optimization loop amounts to compute a cheap matrix-vector multiplication.

6. Algorithms

There are at least two ways to obtain R_i. A *standalone* algorithm will always initialize $R_i = R_0$ and update R_i with rotor R_{i+1} calculated in the optimization loop. In a simulation, we also can take R_i from the outside and perform only one iteration of the optimization to quickly yield a non-optimal rotor for the next simulation step. We call the later the *incremental* algorithm. We have used both choices in our experiments, with excellent results in both cases. In the later case, our experiments indicates that our algorithm preserves the sense of successive rotations.

The *standalone* method is shown in Algorithm 1.

The C++ code using the Eigen library can be found in Listing 1.

The *incremental* algorithm is best suited if we know that the two sets of vectors are almost aligned. Let us suppose that we already have suboptimal rotor estimation R_{prev}, perhaps calculated for a previous simulation step, that almost align the two sets of vectors. We can use R_{prev} to form a direction $\delta D = \epsilon(R - R_{prev})$ which we integrate in the regularization term. So the new rotor estimation is given by $R_{i+1} = R_0 - (H + \epsilon I)^{-1}(g + \epsilon(R_{prev} - R_0))$.

Algorithm 1 Fast Rotor Estimation

Require: $P = \{p_j\}_{j=1}^{n}, Q = \{q_j\}_{j=1}^{n}, C = \{c_j\}_{j=1}^{n}$

$R_0 = [1,0,0,0]^T, R_1 = [1,0,0,0]^T, H = 0_{4\times4}$

for $j = 1$ **to** n **do**

$S_1 = (q_j + p_j) \cdot e_1\ ,\ S_2 = (q_j + p_j) \cdot e_2\ ,\ S_3 = (q_j + p_j) \cdot e_3$

$D_1 = (p_j - q_j) \cdot e_1\ ,\ D_2 = (p_j - q_j) \cdot e_2\ ,\ D_3 = (p_j - q_j) \cdot e_3$

$H = H +$

$$c_j \begin{bmatrix} D_1^2 + D_2^2 + D_3^2 & D_1S_2 - D_2S_1 & D_1S_3 - D_3S_1 & D_2S_3 - D_3S_2 \\ D_1S_2 - D_2S_1 & S_2^2 + S_1^2 + D_3^2 & S_2S_3 - D_3D_2 & D_3D_1 - S_1S_3 \\ D_1S_3 - D_3S_1 & S_2S_3 - D_3D_2 & S_3^2 + S_1^2 + D_2^2 & S_1S_2 - D_2D_1 \\ D_2S_3 - D_3S_2 & D_3D_1 - S_1S_3 & S_1S_2 - D_2D_1 & S_3^2 + S_2^2 + D_1^2 \end{bmatrix}$$

end for

$g = -\begin{bmatrix} H(0,0) & H(1,0) & H(2,0) & H(3,0) \end{bmatrix}^T$

$i = 0$

repeat

$i = i + 1$

$R_{i+1} = normalize(R_0 + (H + \epsilon I)^{-1}(g + \epsilon(R_i - R_0)))$

until $|R_i - R_{i+1}| < \xi$

return $R_{i+1}(0) + R_{i+1}(1)e_{12} + R_{i+1}(2)e_{13} + R_{i+1}(3)e_{23}$

This can be seen as moving the loop from inside the *standalone* algorithm to the outside. When a simulation provides a temporal coherence between consecutive simulation steps, the incremental estimation of rotations is a good choice, since incremental algorithm only requires to solve a linear system for computing the next best rotor. Provided that the simulation converge to stable sets of corresponding vectors, then incremental rotor estimation will converge to R^*. The Algorithm 2 shows the complete algorithm.

Algorithm 2 Incremental Fast Rotor Estimation

Require: $P = \{p_j\}_{j=1}^{n}, Q = \{q_j\}_{j=1}^{n}, C = \{c_j\}_{j=1}^{n}, R_{prev}$

$R_0 = [1,0,0,0]^T, H = 0_{4\times4}$

for $j = 1$ **to** n **do**

$S_1 = (q_j + p_j) \cdot e_1\ ,\ S_2 = (q_j + p_j) \cdot e_2\ ,\ S_3 = (q_j + p_j) \cdot e_3$

$D_1 = (p_j - q_j) \cdot e_1\ ,\ D_2 = (p_j - q_j) \cdot e_2\ ,\ D_3 = (p_j - q_j) \cdot e_3$

$H = H +$

$$c_j \begin{bmatrix} D_1^2 + D_2^2 + D_3^2 & D_1S_2 - D_2S_1 & D_1S_3 - D_3S_1 & D_2S_3 - D_3S_2 \\ D_1S_2 - D_2S_1 & S_2^2 + S_1^2 + D_3^2 & S_2S_3 - D_3D_2 & D_3D_1 - S_1S_3 \\ D_1S_3 - D_3S_1 & S_2S_3 - D_3D_2 & S_3^2 + S_1^2 + D_2^2 & S_1S_2 - D_2D_1 \\ D_2S_3 - D_3S_2 & D_3D_1 - S_1S_3 & S_1S_2 - D_2D_1 & S_3^2 + S_2^2 + D_1^2 \end{bmatrix}$$

end for

$g = -\begin{bmatrix} H(0,0) & H(1,0) & H(2,0) & H(3,0) \end{bmatrix}^T$

$\Delta R = (H + \epsilon I)^{-1}(g + \epsilon(R_{prev} - R_0))$

$R = normalize(R_0 + \Delta R)$

return $R(0) + R(1)e_{12} + R(2)e_{13} + R(3)e_{23}$

7. Comparisons

We select several representative methods e.g. FLAE [11], SVD [3] and QUEST [7] to implement the algorithm. The Eigen library is employed to implement the SVD and QUEST. Another version of QUEST using the Newton iteration is also added for comparison. The tests are ran on a MacBook Pro 13' 2017 computer with the CPU of Intel 3.1GHz 4-core i5. The Visual Studio 2015 C++ software is used to evaluate the algorithms. The results are listed below (Table 1, 2, 3):

TABLE 1. Roll RMSE (deg)

Case	Proposed	SVD Horn 1987	FLAE 2017	QUEST Eigen 1981	QUEST Newton 1981
1	4.9426×10^{-04}	4.3068×10^{-05}	4.3068×10^{-05}	4.3068×10^{-05}	4.3068×10^{-05}
2	3.3673×10^{-04}	5.9781×10^{-05}	5.9781×10^{-05}	5.9781×10^{-05}	5.9781×10^{-05}
3	$2.4912 \times 10^{+00}$	4.3420×10^{-01}	4.3420×10^{-01}	4.3420×10^{-01}	4.3420×10^{-01}
4	$3.4014 \times 10^{+00}$	5.9978×10^{-01}	5.9978×10^{-01}	5.9978×10^{-01}	5.9978×10^{-01}
5	$3.9992 \times 10^{+00}$	4.3545×10^{-01}	$2.1560 \times 10^{+01}$	4.3545×10^{-01}	4.3649×10^{-01}
6	4.9857×10^{-02}	4.9792×10^{-03}	4.9792×10^{-03}	4.9792×10^{-03}	4.9792×10^{-03}
7	8.7731×10^{-02}	8.1146×10^{-03}	8.1146×10^{-03}	8.1146×10^{-03}	8.1146×10^{-03}
8	$3.4008 \times 10^{+02}$	$5.9402 \times 10^{+01}$	$5.9402 \times 10^{+01}$	$5.9402 \times 10^{+01}$	$5.9402 \times 10^{+01}$
9	$5.3870 \times 10^{+02}$	$7.7637 \times 10^{+01}$	$7.7638 \times 10^{+01}$	$7.7637 \times 10^{+01}$	$7.7637 \times 10^{+01}$
10	$1.3513 \times 10^{+01}$	$1.4641 \times 10^{+00}$	$1.5824 \times 10^{+00}$	$1.4641 \times 10^{+00}$	$1.4641 \times 10^{+00}$
11	$1.9366 \times 10^{+01}$	$2.0427 \times 10^{+00}$	$2.9300 \times 10^{+00}$	$2.0427 \times 10^{+00}$	$2.0427 \times 10^{+00}$
12	$1.2517 \times 10^{+01}$	$2.0712 \times 10^{+00}$	$1.7328 \times 10^{+01}$	$2.0712 \times 10^{+00}$	$3.3679 \times 10^{+01}$

TABLE 2. Pitch RMSE (deg)

Case	Proposed	SVD Horn 1987	FLAE 2017	QUEST Eigen 1981	QUEST Newton 1981
1	4.0745×10^{-04}	4.0500×10^{-05}	4.0500×10^{-05}	4.0500×10^{-05}	4.0500×10^{-05}
2	4.1322×10^{-04}	5.3106×10^{-05}	5.3106×10^{-05}	5.3106×10^{-05}	5.3106×10^{-05}
3	$4.0623 \times 10^{+00}$	4.0205×10^{-01}	4.0205×10^{-01}	4.0205×10^{-01}	4.0205×10^{-01}
4	$6.0567 \times 10^{+00}$	5.2600×10^{-01}	5.2600×10^{-01}	5.2600×10^{-01}	5.2600×10^{-01}
5	$2.4391 \times 10^{+00}$	3.9376×10^{-01}	$1.1799 \times 10^{+01}$	3.9376×10^{-01}	3.9469×10^{-01}
6	3.6311×10^{-04}	4.0168×10^{-05}	4.0168×10^{-05}	4.0168×10^{-05}	4.0168×10^{-05}
7	6.2653×10^{-04}	5.3300×10^{-05}	5.3300×10^{-05}	5.3300×10^{-05}	5.3300×10^{-05}
8	$4.3060 \times 10^{+00}$	3.6817×10^{-01}	3.6817×10^{-01}	3.6817×10^{-01}	3.6817×10^{-01}
9	$5.9480 \times 10^{+00}$	4.5583×10^{-01}	4.5583×10^{-01}	4.5583×10^{-01}	4.5583×10^{-01}
10	4.3860×10^{-04}	5.8427×10^{-05}	5.8427×10^{-05}	5.8427×10^{-05}	5.8432×10^{-05}
11	4.3787×10^{-04}	5.7149×10^{-05}	5.7150×10^{-05}	5.7149×10^{-05}	5.7146×10^{-05}
12	$2.6414 \times 10^{+00}$	4.8955×10^{-01}	$3.9323 \times 10^{+00}$	4.8955×10^{-01}	$4.8405 \times 10^{+00}$

TABLE 3. Yaw RMSE (deg)

Case	Proposed	SVD Horn 1987	FLAE 2017	QUEST Eigen 1981	QUEST Newton 1981
1	8.3214×10^{-04}	4.2976×10^{-05}	4.2976×10^{-05}	4.2976×10^{-05}	4.2976×10^{-05}
2	4.6383×10^{-04}	4.8484×10^{-05}	4.8484×10^{-05}	4.8484×10^{-05}	4.8484×10^{-05}
3	$3.9084 \times 10^{+00}$	4.3773×10^{-01}	4.3773×10^{-01}	4.3773×10^{-01}	4.3773×10^{-01}
4	$2.4989 \times 10^{+00}$	4.8366×10^{-01}	4.8366×10^{-01}	4.8366×10^{-01}	4.8366×10^{-01}
5	$1.9721 \times 10^{+00}$	2.5312×10^{-01}	$1.5349 \times 10^{+01}$	2.5312×10^{-01}	2.5373×10^{-01}
6	2.2129×10^{-04}	3.6325×10^{-05}	3.6325×10^{-05}	3.6325×10^{-05}	3.6325×10^{-05}
7	3.5209×10^{-04}	4.9660×10^{-05}	4.9660×10^{-05}	4.9660×10^{-05}	4.9660×10^{-05}
8	$2.7013 \times 10^{+00}$	3.9770×10^{-01}	3.9770×10^{-01}	3.9770×10^{-01}	3.9770×10^{-01}
9	$4.9757 \times 10^{+00}$	4.9710×10^{-01}	4.9710×10^{-01}	4.9710×10^{-01}	4.9710×10^{-01}
10	5.5605×10^{-04}	6.1117×10^{-05}	6.1114×10^{-05}	6.1117×10^{-05}	6.1118×10^{-05}
11	4.7411×10^{-04}	6.0821×10^{-05}	6.0831×10^{-05}	6.0821×10^{-05}	6.0820×10^{-05}
12	$3.3917 \times 10^{+00}$	3.1503×10^{-01}	$2.8467 \times 10^{+00}$	3.1503×10^{-01}	$6.1232 \times 10^{+00}$

In these tables, the Euler angles i.e. roll, pitch and yaw angles are listed for comparisons. The root mean-squared error (RMSE) is presented to describe the difference of accuracy. It is observed that the proposed algorithm has worse accuracy than that from the conventional Wahba's solutions. The time consumption is summarized in Figure 1. We can also see that the loss of accuracy well worth the computation time. The computation time here from the proposed approach is almost 50% of that of the known best algorithm FLAE.

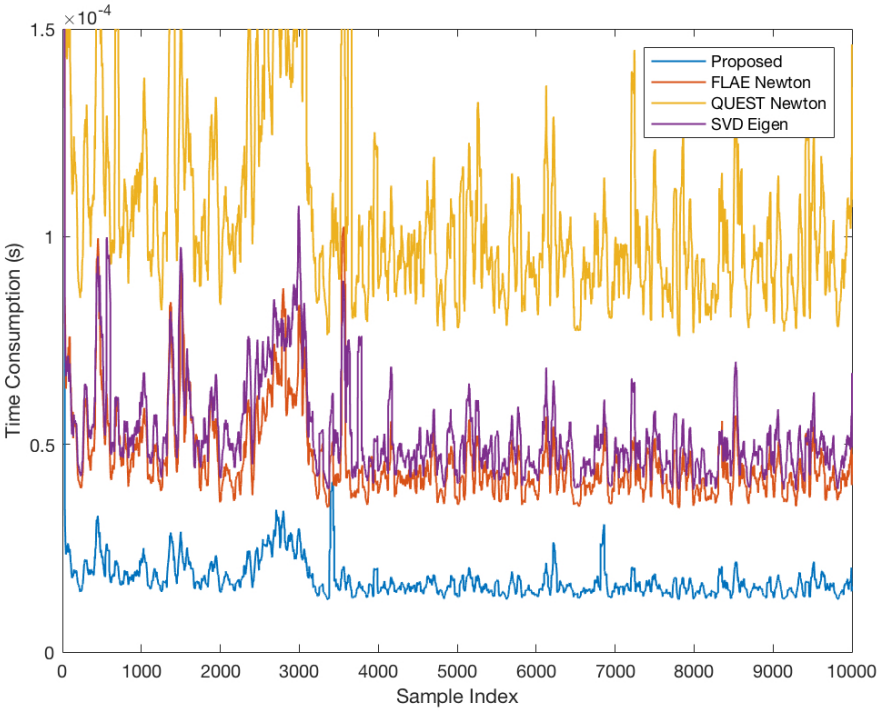

FIGURE 1. Execution time consumption of various algorithms.

8. Conclusion

In this paper, we solve the problem of estimating the best rotation for the alignment of two sets of corresponding 3D vectors. It is based on solving the linear equations derived from the formulation of the problem in Geometric Algebra. The method is fast, robust to noise, accurate and simpler than most other methods. Experimental validation of the performance of the proposed algorithm is presented. The results show that the slightly losing accuracy of the proposed method is well worth the huge advance in execution time consumption. For many applications the vector datasets are very huge i.e. the computation time would be more important than slight change of accuracy.

Therefore, we hope that the proposed algorithm would be of benefit to such applications.

9. Acknowledgement

This research is supported by National Natural Science Foundation of China (NSFC) under the grant of No. 11701496.

10. Conflict of Interests

The authors declare no conflict of interests regarding the publication of this paper.

11. C++ code

Listing 1. C++ code for rotor estimation

```
Quaterniond FastRotorEstimator(
        vector<Vector3d>& P, vector<Vector3d>& Q, vector<double>& weights)
{
        Matrix4d H;
        Vector4d g, R, R_i;
        Vector3d S, D;
        const double epsilon = 1e-6;
        double wj;
        const size_t N = P.size();

        H.setZero();
        for (size_t j = 0; j < N; ++j) {
                wj = weights[j];
                S = Q[j] + P[j];
                D = P[j] - Q[j];
                H(0, 0) += wj*(D[0]*D[0] + D[1]*D[1] + D[2]*D[2]);
                H(0, 1) += wj*(D[0]*S[1] - D[1]*S[0]);
                H(0, 2) += wj*(D[0]*S[2] - D[2]*S[0]);
                H(0, 3) += wj*(D[1]*S[2] - D[2]*S[1]);
                H(1, 1) += wj*(S[1]*S[1] + S[0]*S[0] + D[2]*D[2]);
                H(1, 2) += wj*(S[1]*S[2] - D[2]*D[1]);
                H(1, 3) += wj*(D[2]*D[0] - S[0]*S[2]);
                H(2, 2) += wj*(S[2]*S[2] + S[0]*S[0] + D[1]*D[1]);
                H(2, 3) += wj*(S[0]*S[1] - D[1]*D[0]);
                H(3, 3) += wj*(S[2]*S[2] + S[1]*S[1] + D[0]*D[0]);
        }
        H(1, 0) = H(0, 1);
        H(2, 0) = H(0, 2); H(2, 1) = H(1, 2);
        H(3, 0) = H(0, 3); H(3, 1) = H(1, 3); H(3, 2) = H(2, 3);
        g = -H.col(0);
        H(0, 0) += epsilon; H(1, 1) += epsilon;
        H(2, 2) += epsilon; H(3, 3) += epsilon;
        H = H.inverse();
        R(0) = 1; R(1) = 0; R(2) = 0; R(3) = 0;
        do {
                R_i = R;
                R(0) -= 1.0;
                R.noalias() = H * (g + epsilon * R);
                R(0) += 1.0;
                R /= sqrt(R.dot(R));
        } while ((R_i - R).dot(R_i - R) > 1e-6);
        return Quaterniond(R(0), -R(3), R(2), -R(1));
```

}

References

[1] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(5):698–700, Sep. 1987.

[2] L. Dorst and J. Lasenby, editors. *Guide to Geometric Algebra in Practice*. Springer, 2011.

[3] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, 1987.

[4] D. Mortari. EULER-q algorithm for attitude determination from vector observations. *Journal of Guidance, Control, and Dynamics*, 21(2):328–334, 1998.

[5] J. R. Nieto and A. Susín. Cage based deformations: A survey. *Deformation Models: Tracking, Animation and Applications*, 7:75, 2012.

[6] C. B. U. Perwass. *Geometric algebra with applications in engineering*, volume 4 of *Geometry and Computing*. Springer, Berlin; Heidelberg, 2009.

[7] M. D. Shuster and S. D. Oh. Three-axis attitude determination from vector observations. *Journal of Guidance, Control, and Dynamics*, 4(1):70–77, 1981.

[8] F. Sin, D. Schroeder, and J. Barbic. Vega: Non-linear fem deformable object simulator. *Comput. Graph. Forum*, 32(1):36–48, 2013.

[9] O. Sorkine and M. Alexa. As-rigid-as-possible surface modeling. In *Proceedings of the fifth Eurographics symposium on Geometry processing*, SGP '07, pages 109–116, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[10] G. Wahba. A Least Squares Estimate of Satellite Attitude, 1965.

[11] J. Wu, Z. Zhou, B. Gao, R. Li, Y. Cheng, and H. Fourati. Fast Linear Quaternion Attitude Estimator Using Vector Observations. *IEEE Transactions on Automation Science and Engineering*, X(X):1–13, 2017.

[12] Y. Yang. Attitude determination using Newton's method on Riemannian manifold. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 229(14):2737–2742, 2015.

Mauricio Cele Lopez Belon
Buenos Aires, Argentina
e-mail: `mclopez@outlook.com`