A High-Performance Singularity-Free Rotation Estimator in Geometric Algebra

Mauricio Cele Lopez Belon

Abstract. Well known methods for solving the Wahba's problem are formulated in the linear algebra framework, mainly using the SVD or polar decomposition, or in quaternion's framework which leads to solve a max-eigenvalue problem. Those methods are based on well established numerical algorithms which on the other hand are iterative and computationally expensive. Recently, closed form solutions has been proposed based on an analytic formula for solving the roots of the quartic polynomial associated with the eigenvalue problem. Closed form methods are fast but they have singularities i.e., they completely fail on certain input data. In this paper we propose a high-performance, singularity-free estimator of the best quaternion aligning two sets of corresponding vectors i.e., equivalent to Wahba's problem. It is based on a new formulation of the problem in the Geometric Algebra representation of rotations which is isomorphic to Hamilton's quaternions. The proposed method converges in a fixed number of iterations, its performance is competitive with closed form solutions reported in literature and at the same time is free of singularities, robust to noise and simple to implement.

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 [15]. The problem looks for the optimal rotation between two sets of corresponding vectors. Many effective methods have been developed to solving the problem [1, 6, 9, 12, 20] mainly by formulating it in the quaternions framework, which lead us to solve an eigenvalue problem, or by formulating it in the linear algebra framework and using the Singular Value Decomposition (SVD) and recently by formulating it in the Geometric Algebra framework [3, 11] which at some point casts the solution into linear algebra framework and SVD due

to the lack of geometric algebra numerical algorithms. More recently, closed form solutions for computing the optimal quaternion has been proposed [16, 17, 19, 21] based on solving the roots of the quartic polynomial associated with the eigenvalue problem with an analytic formula.

Accuracy and speed of most methods have been compared in previous works [4, 8, 19] where the trade-off between performance and robustness is assessed. In particular SVD based methods exhibit very good accuracy but low performance and quaternion based methods are in the other end depending on the implementation. Regarding the later methods, the proposed closed form solutions exhibit the best trade-off so far but they have singularities i.e., they completely fail on certain input data.

In this paper we propose a high-performance, singularity-free estimator of the best quaternion aligning two sets of corresponding vectors i.e., equivalent to Wahba's problem. It is based on a new formulation of the problem in the Geometric Algebra representation of rotations which is isomorphic to Hamilton's quaternions. The proposed method converge in a fixed number of iterations being is as fast as the closed form solutions reported in literature and at the same time is free of singularities, robust to noise and simple to implement.

In this work we propose a high-performance, singuarity-free method for solving the Wahba's problem. It is based on a maximizing a convex quadratic energy functional, which is equivalent to Wahba's problem, formulated in geometric algebra framework $\mathbb{G}_3$. The proposed method converges in a fixed number of iterations being is as fast as the closed form solutions reported in literature and at the same time is free of singularities and simple to implement.

Applications of Wahba's solutions are diverse from aerospace engineering computation of spacecraft attitude [19] to mesh deformation in computer graphics [13, 14] for accurate motion construction and restoration [10, 11]. An important characteristic of our method is that it is based almost entirely on symmetric-matrix multiplication so it can be hardware-accelerated using vector instructions or graphics hardware without effort. In engineering fields requiring high-performance vector alignment our method can lower down the time by taking advantage of the AVX (Advanced Vector Extensions) support available in modern microprocessors. Modern matrix packages such as Eigen [5] can produce apropriate AVX instructions automatically. In computer graphics applications our method can lower down the time even more by taking advantage of GPU vector instructions obtaining a performance boost not present in other numerical algorithms.

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. In contrast to previous works we base our

formulation on bivectos instead of vectors for the sake of mathematical convenience. Due to mathematical duality of vector and bivectors in $\mathbb{G}_3$ our formulation is also valid for vectors. The implementation of our algorithm does not require 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 $B = \frac{a \wedge b}{\|a \wedge b\|}$.

$$ab = a \cdot b + a \wedge b = \cos\left(\frac{\theta}{2}\right) + B \sin\left(\frac{\theta}{2}\right). \tag{2.1}$$

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 maximize the following energy function:

$$E(R) = \max_{R \in \mathbb{G}_3^+} \sum_j c_j \|p_j + \tilde{R} q_i R\|^2 \tag{3.1}$$

$$s.t.\ R\tilde{R} = 1 \tag{3.2}$$

where $\{c_j\}_{j=1}^n$ are scalar weights such that $\sum_j^n c_j = 1$. It is a non-linear quadratic maximization problem with a non-linear constraint in R. Notice that the term $\|p_j + \tilde{R} q_i R\|^2$ is dual to the traditional least squares error $\|q_j - R p_j \tilde{R}\|^2$. Duality in the sense that the optimal R is a critical point of both energies. Notice also that $p_j + \tilde{R} q_i R$ is equivalent to $R p_j + q_j R$ by multiplying by R on the left and using the fact that $R\tilde{R} = 1$. The equivalent problem is:

$$E(R) = \max_{R \in \mathbb{G}_3^+} \sum_j c_j \|R p_j + q_j R\|^2 \tag{3.3}$$

$$s.t.\ R\tilde{R} = 1 \tag{3.4}$$

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

We define the commutator product of two bivectors p_j and q_j as $p_j \times q_j = \frac{1}{2}(p_j q_j - q_j p_j)$. The commutator product of bivectors in $\mathbb{G}_3$ can be interpreted as a cross-product of bivectors in the sense that the resulting bivector $B = p_j \times q_j$ is orthogonal to both p_j and q_j. The commutator product allow us to define the geometric product of two bivectors as $AB = A \cdot B + A \times B$. Recall that the inner product of bivectors is not the same as the inner product of vectors, since the square of bivectors is negative, the inner product of bivectors has the opposite sign e.g., $(ae_{12} + be_{13} + ce_{23}) \cdot (de_{12} + ee_{13} + fe_{23}) = -ad - be - cf$.

For a pair of unit bivectors, A and B, the rotor R aligning them can be defined as $R = A\frac{(A+B)}{\|A+B\|}$. Defining the unit bivector $C = \frac{A+B}{\|A+B\|}$ the rotor R is also $R = AC = w + L$ where $w = A \cdot C = \cos(\theta/2)$ and $L = A \times C = \sin(\theta/2)\frac{A \times B}{\|A \times B\|}$.

The constraint $Rp_j + q_jR$ can be rewriten as $(w + L)p_j + q_j(w + L)$. Expanding the geometric product of bivectors in terms of the inner product and the communtator product we get:

$$w(p_j + q_j) + L \cdot (p_j + q_j) + (q_j - p_j) \times L \tag{3.5}$$

In matrix language we can define the following matrix system $M_jR = 0$:

$$M_jR = \begin{bmatrix} 0 & -s_j^T \\ s_j & [d_j]_\times \end{bmatrix} \begin{bmatrix} w \\ L \end{bmatrix} = \begin{bmatrix} -s_j^T L \\ ws_j + d_j \times L \end{bmatrix} \tag{3.6}$$

$$d_j = q_j - p_j \quad s_j = p_j + q_j \tag{3.7}$$

where d_j and s_j are 3×1 column vectors holding bivector's coefficients, M_j is a skew-symmetric 4×4 real matrix, so that $M_j^T = -M_j$. The rotor R is represented as 4×1 column vector made of the scalar w and the 3×1 column vector L holding the bivector's components. The 3×3 matrix $[d_j]_\times$ is representing the skew-symmetric cross-product matrix as usually defined for vectors in $\mathbb{R}^3$.

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

$$F^j = \sqrt{c_j} M_j R$$

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:

$$E(R) = \max_R R^T M R \tag{3.8}$$

$$s.t.\ R^T R = 1 \tag{3.9}$$

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 $\lambda_i \geq 0$.

$$M_j^T M_j = \begin{bmatrix} \|s_j\|^2 & (s_j \times d_j)^T \\ s_j \times d_j & s_j s_j^T - [d_j]_\times^2 \end{bmatrix} \tag{3.10}$$

$$d_j = q_j - p_j \quad s_j = p_j + q_j \tag{3.11}$$

By the spectral theorem the maximizer of $E(R)$ is the eigenvector of M associated with the largest eigenvalue which is a positive number.

4. Convexity

The convexity of the energy $E(R)$ can be proof by showing that its Hessian matrix of second partial derivatives is positive semi-definite. The Hessian matrix of $E(R)$ is $\frac{\partial^2 E(R)}{\partial R} = \sum_j^n c_j M_j^T M_j$ which is symmetric, moreover since M_j is skew-symmetric matrix, the product $M_j^T M_j$ is symmetric positive semi-definite. Then follows that $\sum_j^n c_j M_j^T M_j$ is positive semi-definite and therefore convex, provided that the sum of weights is convex i.e., $\sum_j^n c_j = 1$.

5. Optimal Quaternion using 4D Geometric Algebra

The most time consuming task of the estimation is to compute the eigenvector of M associated with the greatest eigenvalue. Sound numerical algorithms for computing eigenvectors of a real symmetric matrix are SVD, QR factorization and Jacobi iteration among others. However those algorithms finds all eigenvectors of a matrix. In this section we show how to find the largest eigenvalue and its corresponding eigenvector i.e., the required quaternion, using the 4D Geometric Algebra $\mathbb{G}_4$ which is efficient, accurate and suitable for high performance applications.

Let us define four vectors $\mathbf{m}_1$, $\mathbf{m}_2$, $\mathbf{m}_3$, $\mathbf{m}_4$ corresponding to the columns of matrix M:

$$M = \sum_j^n c_j M_j^T M_j = \begin{bmatrix} \mathbf{m}_1 & \mathbf{m}_2 & \mathbf{m}_3 & \mathbf{m}_4 \end{bmatrix}$$

The matrix system that we want to solve is $MR = \lambda R$ for λ corresponding to the largest eigenvalue of M and the corresponding R. We can write the system in the following form:

$$\begin{bmatrix} \mathbf{m}_1 - \lambda e_1 & \mathbf{m}_2 - \lambda e_2 & \mathbf{m}_3 - \lambda e_3 & \mathbf{m}_4 - \lambda e_4 \end{bmatrix} \begin{bmatrix} w \\ L_1 \\ L_2 \\ L_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (5.1)$$

The matrix in equation 5.1, called *characteristic matrix*, is singular ie., the vectors $(\mathbf{m}_1 - \lambda e_1)$, $(\mathbf{m}_2 - \lambda e_2)$, $()\mathbf{m}_3 - \lambda e_3)$ and $()\mathbf{m}_4 - \lambda e_4)$ are linearly dependent (the rank of the matrix is 3). It follows that its outer product must be zero:

$$(\mathbf{m}_1 - \lambda e_1) \wedge (\mathbf{m}_2 - \lambda e_2) \wedge (\mathbf{m}_3 - \lambda e_3) \wedge (\mathbf{m}_4 - \lambda e_4) = 0 \quad (5.2)$$

Which is called the *charactetistic outer porduct* and is equivalent to the characteristic polynomial $P(\lambda) = det(M - \lambda I)$. We can find the largest root of the polynomial using Newton-Raphson method $\lambda_{i+1} = \lambda_i - P(\lambda)/P'(\lambda)$.

For the sake of completeness we show the first derivative of 5.2:

$$\begin{aligned} P'(\lambda) = &-e_1 \wedge (\mathbf{m}_2 - \lambda e_2) \wedge (\mathbf{m}_3 - \lambda e_3) \wedge (\mathbf{m}_4 - \lambda e_4) \\ &-(\mathbf{m}_1 - \lambda e_1) \wedge e_2 \wedge (\mathbf{m}_3 - \lambda e_3) \wedge (\mathbf{m}_4 - \lambda e_4) \\ &-(\mathbf{m}_1 - \lambda e_1) \wedge (\mathbf{m}_2 - \lambda e_2) \wedge e_3 \wedge (\mathbf{m}_4 - \lambda e_4) \\ &-(\mathbf{m}_1 - \lambda e_1) \wedge (\mathbf{m}_2 - \lambda e_2) \wedge (\mathbf{m}_3 - \lambda e_3) \wedge e_4 \end{aligned}$$

We found that $Trace(M)$ is a robust guess for Newton-Raphson iteration because it is larger than the largest eigenvalue and is also close enough to converge in few iterations.

Following [2] we solve the system 5.1 using outer products. For a linear system $A\mathbf{x} = b$ the authors in [2] defines N linear equations of the form $A_j^T \mathbf{x} = b_j$ each of which corresponds to a dual hyper-plane of the form $\mathbf{a}_j \equiv A_j - b_j e_0$ where the solution $\mathbf{x}$ must lie on. The e_0 is an *homogeneous* basis vector needed for enabling projective geometry, enlarging the base space to $N+1$, which interpretation is to be the offset of the hyper-plane. So solution of the linear system is the intersection of N dual hyper-planes, which is given by its outer product.

$$\alpha(\mathbf{x} + e_0)^* = \mathbf{a}_1 \wedge \mathbf{a}_2 \wedge ... \wedge \mathbf{a}_N \tag{5.3}$$

Where the symbol * is the dual operator of the $N+1$ space and α is a weight factor. After taking the dual of $\alpha(\mathbf{x} + e_0)^*$ and divide by the coefficient of e_0 (which is α), the solution $\mathbf{x}$ can be read off the coefficients of the 1-vector.

We know that the null space of $(M - \lambda I)$ is of rank one. Algebraically this means that one of its column vectors is redundant i.e., it can be written in term of the others. In linear algebra this means that the system has infinitely many solutions. The geometric interpretation is that all hyper-planes intersect in a line passing through the origin. Since all solutions lie on the same line and they only differ by a scaling term, a particular solution can be found by fixing the scale. Actually, it is enough to constrain the scale of a single hyper-plane. The homogeneous component e_0 can be interpreted as scale of solution (instead of hyper-plane's offset as in [2]) since it affects only that aspect of the solution. In linear algebra language it is equivalent to set one value at the right hand side of 5.1, however that system can't be solved in linear algebra because it requires to invert a singular matrix.

We define the dual hyper-planes passing through the origin as:

$$\mathbf{a}_i \equiv \mathbf{m}_i - \lambda e_i \tag{5.4}$$

Although in most cases it is enough to set the scale of a single hyper-plane go get a solution it is inconvenient to do so, as will be explained in Section 6. It is more robust to set the scale of all hyper-planes to some $\gamma \neq 0$ which is a scalar value. Intersection can then be found by taking the outer product as:

$$\alpha(\mathbf{x} + 1)^* = (\mathbf{a}_1 + \gamma) \wedge (\mathbf{a}_2 + \gamma) \wedge (\mathbf{a}_3 + \gamma) \wedge (\mathbf{a}_4 + \gamma) \tag{5.5}$$

Distributing the outer product and keeping the terms of grade-3 we get:

$$\alpha(\mathbf{x}+1)^* = \gamma(\mathbf{a}_1 \wedge \mathbf{a}_3 \wedge \mathbf{a}_4 + \mathbf{a}_1 \wedge \mathbf{a}_2 \wedge \mathbf{a}_4 + \mathbf{a}_1 \wedge \mathbf{a}_2 \wedge \mathbf{a}_3 + \mathbf{a}_2 \wedge \mathbf{a}_3 \wedge \mathbf{a}_4) \quad (5.6)$$

Here the symbol * is the dual operator of the 4D space (not 5D as in [2] which allow us to be more efficient) and α is a weight factor. After taking the dual of $\alpha(\mathbf{x}+1)^*$ the eigenvector $\mathbf{x}$ can be read off the coefficients of the 1-vector. Notice that solution needs to be normalized.

6. Robustness and Singularities

As stated before, setting the scale of a single hyperplane is enough to get a valid eigenvector in most cases. However, the choice of which hyper-plane to constraint is problematic. For instance, assuming input vectors without noise, constraining the hyperplane corresponding to the w component of the rotor won't work when the angle of rotation is π because $w = \cos(\pi/2)$ in that case. Similarly, constraining one of the hyper-planes corresponding to L_1, L_4 or L_3 won't work when the angle of rotation is 0 or 4π, etc. So it is complex task to avoid all problematic situations. By constraining all hyper-planes as in 5.5 our geometric algebra method does not suffer from any of those singularities.

Some of the above mentioned singularities are present in classic methods such as QUEST [12] and FOAM [7] but also on methods derived from those, including recent descendants based on analytic formulas [16–19,21]. All those methods are based on finding the eigenvector corresponding to largest eigenvalue of Davenport's matrix. Since it is an indefinite matrix, some eigenvalues are positive and some negative, the Newton-Raphson can fail to find the max eigenvalue (which can be negative). So significant effort has been put on finding fast and robust analytic solutions to the quartic polynomial but no advances has been made on improving robustness on finding the associated eigen-vector.

7. Optimal Computation of M

The symmetric matrix $M_j^T M_j$ has a simple form:

$$M_j^T M_j = \begin{bmatrix} \|s_j\|^2 & (s_j \times d_j)^T \\ s_j \times d_j & s_j s_j^T - d_j d_j^T + \|d_j\|^2 I \end{bmatrix}$$
$$d_j = q_j - p_j \quad s_j = p_j + q_j$$

In terms of p_j and q_j it is:

$$M_j^T M_j = 2 \begin{bmatrix} p_j^T q_j & (p_j \times q_j)^T \\ p_j \times q_j & p_j q_j^T + q_j p_j^T - p_j^T q_j I_{3\times 3} \end{bmatrix} + (\|p_j\|^2 + \|q_j\|^2) I_{4\times 4}$$

All terms of the matrix $M_j^T M_j$ can be derived from the matrix $B = p_j q_j^T$ plus the quantiry $\|p_j\|^2 + \|q_j\|^2$. Since $M_j^T M_j$ is symmetric, only 10 out of 16 elements need to be actually computed. Notice that the trace of $M_j^T M_j$ is $Tr(M_j^T M_j) = 4\|p_j\|^2 + 4\|q_j\|^2$ and $Tr(H) = 4\sum_j (\|p_j\|^2 + \|q_j\|^2)$.

8. Algorithms

The pseudo-code of proposed method is shown in Algorithm 1.

Algorithm 1 Fast Rotor Estimation

Require: $P = \{p_j\}_{j=1}^n, Q = \{q_j\}_{j=1}^n, C = \{c_j\}_{j=1}^n$
$S = B = 0, \gamma = 1$
for $j = 1$ **to** n **do**
  $S = S + c_j(p_j \cdot p_j + q_j \cdot q_j)$
  $B = B + c_j p_j q_j^T$
end for
$\mathbf{m}_1 = (S + Tr(B))e_1 + (B_{12} - B_{21})e_2 + (B_{20} - B_{02})e_3 + (B_{01} - B_{10})e_4$
$\mathbf{m}_2 = (B_{12} - B_{21})e_1 + 2(B_{00} + S - Tr(B))e_2 + (B_{01} + B_{10})e_3 + (B_{20} + B_{02})e_4$
$\mathbf{m}_3 = (B_{20} - B_{02})e_1 + (B_{01} + B_{10})e_2 + 2(B_{11} + S - Tr(B))e_3 + (B_{12} + B_{21})e_4$
$\mathbf{m}_3 = (B_{01} - B_{10})e_1 + (B_{20} + B_{02})e_2 + (B_{12} + B_{21})e_3 + 2(B_{22} + S - Tr(B))e_4$
{Newton-Raphson}
$\lambda_0 = 7S - 3Tr(B)$
repeat
  $\lambda_{i+1} = \lambda_i - P(\lambda_i)/P'(\lambda_i)$
until $\|\lambda_{i+1} - \lambda_i\| < \epsilon$
$\mathbf{a}_1 = \mathbf{m}_1 - \lambda e_1$
$\mathbf{a}_2 = \mathbf{m}_2 - \lambda e_2$
$\mathbf{a}_3 = \mathbf{m}_3 - \lambda e_3$
$\mathbf{a}_4 = \mathbf{m}_4 - \lambda e_4$
$\mathbf{X} = \gamma(\mathbf{a}_1 \wedge \mathbf{a}_3 \wedge \mathbf{a}_4 + \mathbf{a}_1 \wedge \mathbf{a}_2 \wedge \mathbf{a}_4 + \mathbf{a}_1 \wedge \mathbf{a}_2 \wedge \mathbf{a}_3 + \mathbf{a}_2 \wedge \mathbf{a}_3 \wedge \mathbf{a}_4)$
$R = normalize(\langle \mathbf{X}^* \rangle_1)$
return $R(0) + R(1)e_{12} + R(2)e_{13} + R(3)e_{23}$

An optimized C++ code using the Eigen library [5] can be found in Listing 1.

9. Comparisons

We select several representative methods e.g. FLAE [19], SVD [6] and QUEST [12] to implement the algorithm. The Eigen library [5] 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 2019 C++ compiler is used in our experiments.

10. 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.

11. C++ code

Listing 1. C++ code for rotor estimation

```
Quaterniond GAFastRotorEstimator(
    const vector<Vector3d>& P,
    const vector<Vector3d>& Q,
    const vector<double>& w)
{
  array<double, 10> H;
  Matrix3d Sx;
  Vector3d S;
  double wj;
  const size_t N = P.size();
  Sx.setZero();
  S.setZero();
  for (size_t j = 0; j < N; ++j) {
    wj = w[j];
    const Vector3d& Qj = Q[j];
    const Vector3d& Pj = P[j];
    S.noalias() += wj * (Pj + Qj);
    Sx.noalias() += (wj * Pj) * Qj.transpose();
  }
  wj = S.dot(S);
  H[0] = wj + 2.0 * Sx.trace();          // (3,3)
  wj = wj - 2.0 * Sx.trace();
  H[1] = 2.0 * (Sx(1, 2) - Sx(2, 1)); // (3,0)
  H[2] = 2.0 * (Sx(2, 0) - Sx(0, 2)); // (3,1)
  H[3] = 2.0 * (Sx(0, 1) - Sx(1, 0)); // (3,2)
  H[4] = 4.0 * Sx(0, 0) + wj;          // (0,0)
  H[5] = 2.0 * (Sx(0, 1) + Sx(1, 0)); // (1,0)
  H[6] = 2.0 * (Sx(2, 0) + Sx(0, 2)); // (2,0)
  H[7] = 4.0 * Sx(1, 1) + wj;          // (1,1)
  H[8] = 2.0 * (Sx(1, 2) + Sx(2, 1)); // (2,1)
  H[9] = 4.0 * Sx(2, 2) + wj;          // (2,2)

  double lambda = H[0]+H[4]+H[7]+H[9];
  double lambda_prev = 0;
  while(std::abs(lambda - lambda_prev) > 1e-5) {
    lambda_prev = lambda;
    lambda = lambda - characteristic(H, lambda) / deriv_characteristic(H, lambda);
  };

  Quaterniond R( hyperplanes_intersection(H, lambda) );
  R.normalize();
  return R;
```

```
}

/*
a = H4 * e1 + H5 * e2 + H[6] * e3 + H1 * e4;
b = H5 * e1 + H7 * e2 + H8 * e3 + H2 * e4;
c = H6 * e1 + H8 * e2 + H9 * e3 + H3 * e4;
d = H1 * e1 + H2 * e2 + H3 * e3 + H0 * e4;

? det = (a - lambda * e1)^(b - lambda * e2)^(c - lambda * e3)^(d - lambda * e4);
*/
double characteristic(const array<double, 10>& H, double lambda)
{
  return (((H[4] - lambda) * (H[7] - lambda) + (-(H[5] * H[5]))) * (H[9] - lambda)
  + (-(((H[4] - lambda) * H[8] + (-(H[6] * H[5]))) * H[8]))
  + (H[5] * H[8] + (-(H[6] * (H[7] - lambda)))) * H[6]) * (H[0] - lambda)
  + (-((((H[4] - lambda) * (H[7] - lambda) + (-(H[5] * H[5]))) * H[3] + (-(((H[4]
  - lambda) * H[2] + (-(H[1] * H[5]))) * H[8])) + (H[5] * H[2] + (-(H[1] * (H[7]
  - lambda)))) * H[6]) * H[3])) + (((H[4] - lambda) * H[8] + (-(H[6] * H[5]))) * H[3]
  + (-(((H[4] - lambda) * H[2] + (-(H[1] * H[5]))) * (H[9] - lambda))) + (H[6] * H[2]
  + (-(H[1] * H[8]))) * H[6]) * H[2] + (-(((H[5] * H[8] + (-(H[6] * (H[7] - lambda))))
  * H[3] + (-((H[5] * H[2] + (-(H[1] * (H[7] - lambda)))) * (H[9] - lambda)))
  + (H[6] * H[2] + (-(H[1] * H[8]))) * H[8]) * H[1])); // e1^e2^e3^e4
}

/*
a = H4 * e1 + H5 * e2 + H6 * e3 + H1 * e4;
b = H5 * e1 + H7 * e2 + H8 * e3 + H2 * e4;
c = H6 * e1 + H8 * e2 + H9 * e3 + H3 * e4;
d = H1 * e1 + H2 * e2 + H3 * e3 + H0 * e4;

? ddet = ((-1 * e1)^(b - lambda * e2)^(c - lambda * e3)^(d - lambda * e4)) +
((a - lambda * e1)^(-1 * e2)^(c - lambda * e3)^(d - lambda * e4)) +
((a - lambda * e1)^(b - lambda * e2)^(-1 * e3)^(d - lambda * e4)) +
((a - lambda * e1)^(b - lambda * e2)^(c - lambda * e3)^(-1 * e4));
*/
double deriv_characteristic(const array<double, 10>& H, double lambda)
{
  return ((-(H[7] - lambda)) * (H[9] - lambda) + (-((-H[8]) * H[8]))) * (H[0]
  - lambda) + (-(((-(H[7] - lambda)) * H[3] + (-((-H[2]) * H[8]))) * H[3])) + ((-H[8])
  * H[3] + (-((-H[2]) * (H[9] - lambda)))) * H[2] + ((-(H[4] - lambda)) * (H[9]
  - lambda) + H[6] * H[6]) * (H[0] - lambda) + (-(((-(H[4] - lambda)) * H[3] + H[1]
  * H[6]) * H[3])) + (-((H[6] * H[3] + (-(H[1] * (H[9] - lambda)))) * H[1]))
  + (-((H[4] - lambda) * (H[7] - lambda) + (-(H[5] * H[5])))) * (H[0] - lambda)
  + ((H[4] - lambda) * H[2] + (-(H[1] * H[5]))) * H[2] + (-((H[5] * H[2] + (-(H[1]
  * (H[7] - lambda)))) * H[1])) + (-(((H[4] - lambda) * (H[7] - lambda) + (-(H[5]
  * H[5]))) * (H[9] - lambda) + (-(((H[4] - lambda) * H[8] + (-(H[6] * H[5])))
  * H[8])) + (H[5] * H[8] + (-(H[6] * (H[7] - lambda)))) * H[6])); // e1^e2^e3^e4
}

/*
a = H4 * e1 + H5 * e2 + H6 * e3 + H1 * e4;
b = H5 * e1 + H7 * e2 + H8 * e3 + H2 * e4;
c = H6 * e1 + H8 * e2 + H9 * e3 + H3 * e4;
d = H1 * e1 + H2 * e2 + H3 * e3 + H0 * e4;

? R0 = ((a - lambda * e1)^(b - lambda * e2)^(c - lambda * e3 + 1)^(d - lambda * e4)) . (
*/
Vector4d hyperplanes_intersection(const array<double, 10>& H, double lambda)
{
  Vector4d R0;
  R0[0] = (H[5] * H[8] + (-(H[6] * (H[7] - lambda)))) * (H[0] - lambda)
  + (-((H[5] * H[2] + (-(H[1] * (H[7] - lambda)))) * H[3])) + (H[6] * H[2]
  + (-(H[1] * H[8]))) * H[2]; // e1
  R0[1] = (-(((H[4] - lambda) * H[8] + (-(H[6] * H[5]))) * (H[0] - lambda)
  + (-(((H[4] - lambda) * H[2] + (-(H[1] * H[5]))) * H[3])) + (H[6] * H[2]
  + (-(H[1] * H[8]))) * H[1])); // e2
  R0[2] = ((H[4] - lambda) * (H[7] - lambda) + (-(H[5] * H[5]))) * (H[0]
  - lambda) + (-(((H[4] - lambda) * H[2] + (-(H[1] * H[5]))) * H[2]))
```

```
  + (H[5] * H[2] + (-(H[1] * (H[7] - lambda)))) * H[1]; // e3
  R0[3] = (-(((H[4] - lambda) * (H[7] - lambda) + (-(H[5] * H[5]))) * H[3]
  + (-(((H[4] - lambda) * H[8] + (-(H[6] * H[5]))) * H[2])) + (H[5] * H[8]
  + (-(H[6] * (H[7] - lambda)))) * H[1])); // e4
  return R0;
}
```

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] S. De Keninck and L. Dorst. Geometric algebra levenberg-marquardt. In M. Gavrilova, J. Chang, N. M. Thalmann, E. Hitzer, and H. Ishikawa, editors, *Advances in Computer Graphics*, pages 511–522, Cham, 2019. Springer International Publishing.

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

[4] D. Eggert, A. Lorusso, and R. Fisher. Estimating 3-d rigid body transformations: a comparison of four major algorithms. *Machine Vision and Applications*, 9(5):272–290, Mar 1997.

[5] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[6] 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.

[7] L. Markley. Attitude determination from vector observations: A fast optimal matrix algorithm. *Journal of the Astronautical Sciences*, 41, 07 1993.

[8] L. Markley. 30 years of wahba's problem. 02 1999.

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

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

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

[12] 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.

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

[14] 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.

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

[16] J. Wu, M. Liu, Z. Zhou, and R. Li. Fast rigid 3d registration solution: A simple method free of svd and eigen-decomposition, 2018.

[17] J. Wu, M. Liu, Z. Zhou, and R. Li. Fast symbolic 3d registration solution, 2018.

[18] J. Wu, Z. Zhou, J. Chen, H. Fourati, and R. Li. Fast Complementary Filter for Attitude Estimation Using Low-Cost MARG Sensors. *IEEE Sensors Journal*, 16(18):6997–7007, 2016.

[19] 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.

[20] 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.

[21] Y. Yang and Z. Zhou. An analytic solution to wahba's problem. *Aerospace Science and Technology*, 30(1):46–49, Oct 2013.

Mauricio Cele Lopez Belon
Madrid, España
e-mail: `mclopez@outlook.com`