ICIAM 2019 SEMA SIMAI Springer Series 13

Sebastià Xambó-Descamps *Ed.*

Systems, Patterns and Data Engineering with Geometric Calculi

Springer

SEMA SIMAI Springer Series

ICIAM 2019 SEMA SIMAI Springer Series

Volume 13

This sub-series of the SEMA SIMAI Springer Series aims to publish some of the most relevant results presented at the ICIAM 2019 conference held in Valencia in July 2019.

The sub-series is managed by an independent Editorial Board, and will include peer-reviewed content only, including the Invited Speakers volume as well as books resulting from mini-symposia and collateral workshops.

The series is aimed at providing useful reference material to academic and researchers at an international level.

More information about this subseries at http://www.springer.com/series/16499

Sebastià Xambó-Descamps
Editor

Systems, Patterns and Data Engineering with Geometric Calculi

Springer

Editor
Sebastià Xambó-Descamps
Department of Mathematics
Universitat Politècnica de Catalunya
Barcelona, Spain

ISSN 2199-3041 ISSN 2199-305X (electronic)
SEMA SIMAI Springer Series
ISSN 2662-7183 ISSN 2662-7191 (electronic)
ICIAM 2019 SEMA SIMAI Springer Series
ISBN 978-3-030-74485-4 ISBN 978-3-030-74486-1 (eBook)
https://doi.org/10.1007/978-3-030-74486-1

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

The genesis of this collection is the homonymous mini-symposium (MS) held on July 16, 2019, in the **International Congress of Industrial and Applied Mathematics** (ICIAM-2019) organized by the **Spanish Society of Applied Mathematics** (SEMA) at the University of Valencia (July 15–19, 2019). Its initial idea, however, was born out of discussions I held with Carlile Lavor, Pablo Colapinto and Srđan Lazendić on the occasion of the 7th Conference on *Applied Geometric Algebras in Computer Science and Engineering* (AGACSE 2018), a satellite of ICIAM-2019 held at the University of Campinas, Brazil, July 23rd to 27th, 2018.

The goal of the mini-symposium was envisaged to overview the basic ideas of geometric algebra/calculus, to report on state-of-the-art applications showcasing its advantages, and to explore the bearing of the formalisms in novel contexts, with a particular view to automatic learning. The idea was given further considerations, within the constraints of the ICIAM-2019 MS, and finally the outcome was the following program:

1. *Geometric calculus techniques in science and engineering* (Sebastià Xambó-Descamps, Chair)
2. *Bringing new perspectives to robotics and computer vision* (Isiah Zaplana)
3. *Geometric algebra and distance geometry* (Carlile Lavor)
4. *Embedded Coprocessors for Native Execution of Geometric Algebra Operations* (Salvatore Vitabile)
5. *Hypercomplex algebras for art investigation* (Srđan Lazendić)
6. *Conformal Geometric Algebra for Medical Imaging* (Salvatore Vitabile)
7. *Geometric bio-inspired deep learning* (Eduardo Ulises Moya)
8. *Geometric Calculus meets Deep Learning* (Sebastià Xambó-Descamps)

E. U. Moya had send a paper to the AGACSE 2018 Conference, and although it turned out that he could not travel to Campinas (his work was presented there by Eduardo Bayro-Corrochano, his PhD advisor), he expected to attend ICIAM-2019. In contrast, Pablo Colapinto attended AGACSE 2018, and there he

agreed to deliver a talk at the MS on *The New Geometry of Computer Aided Design* if he could manage to join the MS, but at the end he could not.

The collection of papers in this volume differs from the program in several respects. The talk number 5 did not end in a paper, nor did the tutorial 1. The talks 4 and 6 were merged into a single paper. In addition to the five remaining contributions (perhaps with additional authors and variations in the titles), we invited three additional authors: Leo Dorst, Pablo Colapinto, and Leandro A. F. Fernandes. The final contents are as follows:

1. *New Perspectives on Robotics with Geometric Calculus* (Isiah Zaplana)
2. *Recent advances on oriented conformal geometric algebra applied to molecular distance geometry* (Carlile Lavor and Rafael Alves)
3. *Geometric Calculus Applications to Medical Imaging: Status and Perspectives* (Silvia Franchini and Salvatore Vitabile)
4. *Optimal Combination of Orientation Measurements Under Angle, Axis and Chord Metrics* (Leo Dorst)
5. *Space-Bending Lattices through Conformal Transformation of Principal Contact Elements* (Pablo Colapinto)
6. *Exploring Lazy Evaluation and Compile-Time Simplifications for Efficient Geometric Algebra Computations* (Leandro A. F. Fernandes)
7. *A Quaternion Deterministic Monogenic CNN Layer for Contrast Invariance* (Eduardo U. Moya, S. Xambó-Descamps, Sebastián Salazar Colores, Abraham Sánchez, Ulises Cortés)
8. *Geometric Calculus and Deep Learning–An Overview* (S. Xambó-Descamps and Eduardo U. Moya)

The main idea of this collection is well aligned with the core purposes of the MS. The first three contributions, which correspond to lectures at the MS, offer perspectives on recent advances in the application GC in the areas of robotics, molecular geometry, and medical imaging. The next three, especially invited, hone the expressiveness of GC in orientation measurements under different metrics, the treatment of contact elements, and the investigation of efficient computational methodologies. The last two, which also correspond to lectures at the MS, deal with two aspects of deep learning: a presentation of a concrete quaternionic convolutional neural network layer for image classification that features contrast invariance and a general overview of automatic learning aimed at steering the development of neural networks whose units process elements of a suitable algebra, as, for instance, a geometric algebra.

It is a pleasant duty to thank the organizers of ICIAM-2019, and in particular the organizers of the mini-symposia, the speakers at our MS, and the authors of the papers collected in this volume.

January 2021 Sebastià Xambó-Descamps

Contents

About the Editor

Sebastià Xambó-Descamps is an Emeritus Full Professor of Mathematics at the Universitat Politècnica de Catalunya/Barcelonatech and is currently visiting the Barcelona Supercomputer Center. Previously, he held positions at the Universidad Complutense de Madrid (Full Professor) and at the Universitat de Barcelona (Associate Professor). His main research interests are algebraic geometry (intersection theory and enumerative geometry), information theory (coding theory and code-based post-quantum cryptography), computational methods (computer mathematics and symbolic computation), mathematical models in physics and engineering, geometric calculus and its applications to deep learning. He is author of *Block Error-Correcting Codes, A Computational Primer* and *Real Spinorial Groups, A Short Mathematical Introduction.*

New Perspectives on Robotics with Geometric Calculus

Isiah Zaplana

Abstract One of the most successful applications of geometric calculus to engineering refers to robotics and computer vision. In this line, this chapter presents an overview of the main classical problems in robot kinematics and motion planning and explains how geometric calculus has been used to solve them by exploiting their algebraic and geometric properties (such as, for instance, that every isometry can be compactly represented, the geometric covariance, the properties of the rotor group and the bivector algebra). Besides, it also introduces recent open problems in robotics and explains how geometric calculus can be used to contribute to their solutions.

1 Introduction

Nowadays, robotics is a well-known field. From science-fiction to real industry, during the last years we have thought and talked a lot about robots and their role in our modern society. But, what exactly is a robot? It is a programmable machine able to carry out a complex series of tasks with some level of autonomy [17]. This definition, however, is too broad for the purposes of this chapter since it includes robots like the vacuum Roomba robot and the Lego toy robot. Here, instead, we are going to focus on *serial industrial robots*, which are of fundamental importance in the industry. They perform tasks that human operators cannot, such as carrying heavy objects, painting, grasping, moving and handling large pieces, etc. Either by assisting human operators or by completely replacing them, serial industrial robots have turned out to be an indispensable element of modern industry.

Geometrically speaking, these robots are sequences of rigid-bodies (called *links*) connected by means of motor-actuated kinematic pairs (called *joints*). Every joint provides relative motion between the two consecutive links it connects (see Fig. 1). The most important point is the free end of the last link, the so-called *end-effector*. Its importance relies on the fact that every tool the robot needs to perform its tasks—painting tools, screw-drives, robotic hands, grippers, etc.—is placed at the

I. Zaplana (✉)
Department of Mechanical Engineering, KU Leuven, Leuven, Belgium
e-mail: isiah.zaplana@kuleuven.be

S. Xambó-Descamps (ed.), *Systems, Patterns and Data Engineering with Geometric Calculi*, SEMA SIMAI Springer Series 13, https://doi.org/10.1007/978-3-030-74486-1_1

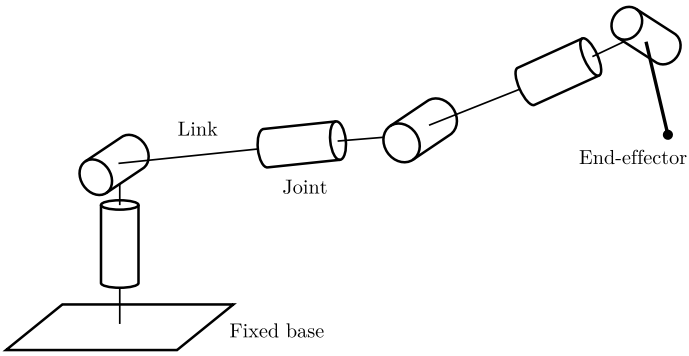


Fig. 1 General scheme of a serial industrial robot

end-effector and, therefore, it is fundamental to know: (1) where the end-effector is at each *configuration* of the entire robot, i.e., its position and orientation in $\mathbb{R}^3$ at each configuration, and (2) how to move the robot so its end-effector arrives in a predefined desired position and orientation. The first problem is known as the *forward kinematics* problem of a serial industrial robot, while the second is known as the *motion planning* problem. Since both problems analyze the motion of a serial industrial robot without considering the dynamics of the system, they are said to be *kinematic* problems or, more precisely, *robotic kinematic* problems.

This chapter provides a formal mathematical introduction to both problems and develops some tools based on conformal geometric algebra to solve them. In particular, we are going to deal with the forward and inverse kinematics, where the latter is a non-trivial kinematic subproblem of the motion planning problem, as well as with the motion planning problem itself. The rest of the chapter is organized as follows: Sect. 2 formulates the forward and inverse kinematics for general serial industrial robots using conformal geometric algebra and shows how to solve them using this mathematical framework, while the same is done with the motion planning problem in Sect. 3. Finally, we present the conclusions and some open problems in the final section, Sect. 4.

2 Forward and Inverse Kinematics

As stated before, robot kinematics is about studying the motion of general robots without considering the dynamics of the system. In particular, for serial industrial robots, it entails the study of two well-differentiated problems: the forward kinematics and the inverse kinematics. Before we formally formulate both problems, we need to introduce some preliminary concepts.

The joints of serial industrial robots are of two types: *revolute*, if their motion is rotational, and *prismatic*, if their motion is translational. The amount of such motion

is known as the *joint variable* and is denoted by q. Then, for every joint $1 \leq i \leq n$, q_i is either an angle, θ_i, if joint i is revolute or a displacement, d_i, if joint i is prismatic.

Definition 1. The vector of all joint variables $\boldsymbol{q} = (q_1, \ldots, q_n)$ is said to be the *configuration* of the robot. The space of all configurations of a robot is called the *configuration* or *joint space* of the robot and is denoted by C.

A frame $\{\boldsymbol{o}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$ is attached to the end-effector of the robot. The three-dimensional point $\boldsymbol{o} \in \mathbb{R}^3$ describes the position of the end-effector, while the right-handed linear frame $\{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$ describes its orientation. We will use this notation through the rest of the chapter to avoid confusion, so a linear frame will be a set of three mutually orthogonal unitary vectors $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$, while a frame will be the pair formed by a three-dimensional point and a linear frame.

Definition 2. The space of all positions and orientations of the end-effector with respect to a reference frame is called the *operational space* of the robot and is denoted by $\mathcal{X}$. Clearly, $\mathcal{X} \subset SE(3)$, where $SE(3)$ denotes the three-dimensional special Euclidean group.

Definition 3. A serial industrial robot is said to have n *degrees of freedom* (DoF) if its configuration is specified by n joint variables.

Given a configuration $\boldsymbol{q} \in C$, we want to find the position and orientation of the end-effector associated with that configuration. This is known as the forward kinematics problem. Conversely, the inverse kinematics problem consists of finding the configurations associated with a predefined position and orientation of the end-effector. In other words, the forward kinematics problem consists on finding the continuous function f that assigns the position and orientation $\boldsymbol{x}$ of the end-effector to each configuration $\boldsymbol{q}$:

$$\begin{aligned} f : C &\to \mathcal{X} \\ \boldsymbol{q} &\mapsto \boldsymbol{x} \end{aligned} \tag{1}$$

Since f is well-defined, the forward kinematics is said to have *analytical solution*. However, f has not, in general, a global inverse [2, 3] and, as a consequence, the inverse kinematics problem of an arbitrary serial industrial robot has no analytical solution. Hence, geometric and numerical methods need to be developed to solve it.

2.1 Forward Kinematics

In practice, we can also attach a frame to each joint of the robot, that is, for every $1 \leq i \leq n$, we have a frame $\{\boldsymbol{o}_i, \boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i\}$ attached to joint i. They are known as the *joint frames* of the robot.

Each one of these joint frames depends on the position and orientation of the previous joints, i.e., it depends on the previous joint frames. In particular, for $2 \leq$

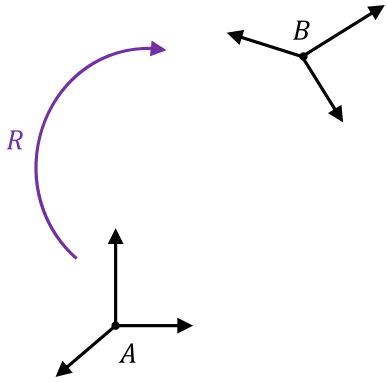


Fig. 2 Given two different frames, there is always a rotor relating one to the other. In this case, such a rotor R describes a screw or helical motion (translation followed by a rotation around the same axis) between the two frames

$i \leq n$, the i-th joint frame is related to the $(i-1)$-th joint frame, that works as a reference frame. The case $i = 1$ is a special case since there is no previous joint. Hence, the first joint frame has, as a reference frame, the frame placed at the base of the robot, the so-called *base frame*. It is fixed, i.e., it does not depend on the configuration of the robot. Due to this, the base frame is usually considered the global reference frame of the robot.

Each joint frame is determined, not only by the joint variable q_i, but also by the following set of rules [16]:

- The z_i-axis is aligned with the rotational/translational joint axis.
- The x_i-axis is aligned with the common perpendicular to z_i and z_{i-1}, where the latter is the z-axis of the $(i-1)$-th joint frame.
- The origin o_i is set at the intersection of z_i with the common perpendicular to z_i and z_{i-1}.

It is well-known that, given any two linear frames $\{x, y, z\}$ and $\{x', y', z'\}$, there always exists a rotor $R \in \mathcal{G}_3^+$, uniquely determined up to sign, such that:

$$\left\{ \begin{array}{l} x' = Rx\widetilde{R} \\ y' = Ry\widetilde{R} \\ z' = Rz\widetilde{R} \end{array} \right\} \tag{2}$$

In conformal geometric algebra, since translations are also encoded by rotors (see section 2.4 of chapter 2 in [13]), we can extend the result to include also their respective origins (see Fig. 2).

Theorem 1. *Given two arbitrary frames* $\{o, x, y, z\}$ *and* $\{o', x', y', z'\}$*, there always exists a rotor* $R \in \mathcal{G}_{4,1}$*, uniquely determined up to sign, satisfying that:*

$$\left\{\begin{array}{l} \boldsymbol{o}' = R\boldsymbol{o}\widetilde{R} \\ \boldsymbol{x}' = R\boldsymbol{x}\widetilde{R} \\ \boldsymbol{y}' = R\boldsymbol{y}\widetilde{R} \\ \boldsymbol{z}' = R\boldsymbol{z}\widetilde{R} \end{array}\right\} \tag{3}$$

In addition, $R = R_1 R_2$*, where:*

$$R_1 = 1 - \frac{\boldsymbol{v}e_\infty}{2}, \tag{4}$$

with $\boldsymbol{v} = \boldsymbol{o}' - \boldsymbol{o}$ *and*

$$R_2 \in \mathcal{G}_3^+, \tag{5}$$

i.e., R_1 *is the rotor encoding the translation that maps* $\boldsymbol{o}$ *to* $\boldsymbol{o}'$ *and* R_2 *is the rotor encoding the rotation that transforms* $\{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$ *to* $\{\boldsymbol{x}', \boldsymbol{y}', \boldsymbol{z}'\}$*.*

As a corollary of Theorem 1, we have that we can recover the rotor that transforms the $(i-1)$-th joint frame into the i-th joint frame for every $1 \leq i \leq n$. These rotors allow to relate the base frame of the robot with the end-effector, thus allowing to compute its position and orientation with respect to the global reference frame—i.e., the base frame. Notice that, since the joint frames are determined by the joint variables, so are the rotors recovered from them.

Now, given two consecutive joint frames, how can we construct the rotor that transforms one into the other? There are different ways. For instance, in [13, chapter 4], we followed the standard convention in robot kinematics (known as the Denavit-Hartenberg convention) to construct each intermediate joint frame by means of a set of four parameters (the Denavit-Hartenberg parameters) and, with those parameters, recover the rotor relating each frame to the following one. Here, we are going to follow a different approach based on the use of the reciprocal frame.

Definition 4. Given a linear frame $\{e_1, e_2, e_3\}$, its *reciprocal frame* $\{e^1, e^2, e^3\}$ is the linear frame satisfying that $e_i \cdot e^j = \delta_{ij}$, where $\delta_{(\cdot)}$ denotes the Kronecker delta.

In [7], the reader can find some useful properties of reciprocal frames as well as an explicit expression for calculating them. In addition, there is an expression for the rotor relating two different linear frames $\{e_1, e_2, e_3\}$ and $\{f_1, f_2, f_3\}$:

$$R = \frac{1 + f_1 e^1 + f_2 e^2 + f_3 e^3}{|1 + f_1 e^1 + f_2 e^2 + f_3 e^3|}. \tag{6}$$

If we apply Eq. (6) to the joint frames, we have that:

$$R_{i-1}^i = \frac{1 + \boldsymbol{x}_i \boldsymbol{x}^{i-1} + \boldsymbol{y}_i \boldsymbol{y}^{i-1} + \boldsymbol{z}_i \boldsymbol{z}^{i-1}}{|1 + \boldsymbol{x}_i \boldsymbol{x}^{i-1} + \boldsymbol{y}_i \boldsymbol{y}^{i-1} + \boldsymbol{z}_i \boldsymbol{z}^{i-1}|} \tag{7}$$

is the rotor that transform the $(i-1)$-th linear joint frame into the i-th linear joint frame. Now, taking the rotor defined in Eq. (4) and applying it to the origins of these joint frames, we have that:

$$T_{i-1}^{i} = 1 + \frac{e_\infty \boldsymbol{v}}{2}, \tag{8}$$

where $\boldsymbol{v} = \boldsymbol{o}_i - \boldsymbol{o}_{i-1}$.

In summary, the rotor that transforms the $(i-1)$-th joint frame into the i-th joint frame is:

$$M_{i-1}^{i} = T_{i-1}^{i} R_{i-1}^{i}. \tag{9}$$

Hence, the rotor that relates the base frame with the end-effector for a specific configuration $\boldsymbol{q} \in C$ is:

$$M = M_0^1 M_1^2 \cdots M_{n-1}^n. \tag{10}$$

Therefore, if we take the base frame, i.e., the global reference frame $\{\boldsymbol{o}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$, and we applied M to each one of its elements (with the sandwiching product $M(\cdot)\widetilde{M}$), we get the frame attached to the end-effector, i.e., its position and orientation with respect to that global reference frame. Again, all we have done so far is configuration-dependent, which means that M is the rotor that solves the forward kinematics problem for a specific $\boldsymbol{q} \in C$. If we change the configuration, we will need to construct the new joint frames and recover the associated rotors to compute its product. In general, we have the following result.

Theorem 2. *For every configuration $\boldsymbol{q} = (q_1, \ldots, q_n) \in C$, the position and orientation of the end-effector associated with $\boldsymbol{q}$ is:*

$$P' = M(\boldsymbol{q}) P \widetilde{M}(\boldsymbol{q}) = M_0^1(q_1) \cdots M_{n-1}^n(q_n) P \widetilde{M}_{n-1}^n(q_n) \cdots \widetilde{M}_0^1(q_1), \tag{11}$$

where P is the position (orientation) of the base frame and P' denotes the position (orientation) of the end-effector.

The advantages of the approach presented here include a compact formulation of both the forward kinematics problem and its solution. In addition, since rotors are elements of the algebra, the manipulation of complex geometric structures (like serial chains) becomes easier.

2.2 Inverse Kinematics

A serial industrial robot is said to have a *spherical wrist* if their last three joint axes either intersect at a single point or are parallel. It is well-known that serial industrial robots with a spherical wrist have always analytical or closed-form solutions (by Pieper's theorem [15]). In addition, the proof of Pieper's theorem is constructive in the sense that closed-form solutions are explicitly derived for any type of robot

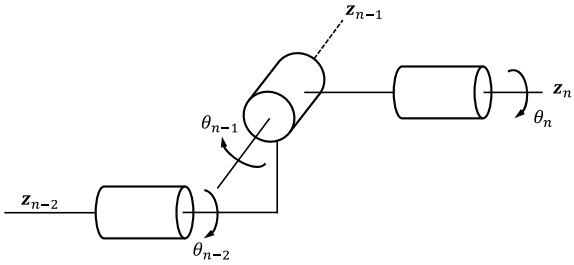


Fig. 3 Non-spherical wrist. Notice the offset between the $(n-2)$-th and the $(n-1)$-th joints

with a spherical wrist. However, if there is an offset between any of the last three joint axes (as shown in Fig. 3), then the robot has no longer a spherical wrist and, hence, Pieper's theorem cannot be applied. To solve the inverse kinematics problem for those robots, Paul [14] developed a method based on the homogeneous matrices T_{i-1}^{i} used to describe the kinematics of serial industrial robots [16]. Indeed, given the kinematic identity:

$$T_0^1 \cdot T_1^2 \cdots T_{n-1}^n = T_0^n, \tag{12}$$

where we recognize in T_0^n the homogeneous matrix describing the position and orientation of the end-effector with respect to the base frame and where T_{i-1}^{i} only depends on the joint variable q_i, Paul's method consists of analyzing each one of the following matrix equations:

$$T_{i-1}^{i} \cdots T_{n-1}^{n} = \left(T_{i-2}^{i-1}\right)^{-1} \cdots \left(T_0^1\right)^{-1} \cdot T_0^n \quad \text{for } i = 2, \ldots, n \tag{13}$$

to isolate known trigonometric equations that can be solved analytically for one or more joint variables. However, the large number of different combinations together with the intricacies for solving analytically arbitrary trigonometric equations makes this method not suitable for kinematic chains of complex geometry. Most of the contributions found in the literature [5, 8, 10] focus either on numerical methods or on particular geometric methods. Although the latter can only be applied to the specific robots they have been designed for, they give the complete set of solutions, contrary to what happens with the former, where only one solution is obtained. Nevertheless, geometric methods are difficult to design, especially for robots without a spherical wrist. This is one of the reasons why conformal geometric algebra turns out to be useful to deal with this problem. For instance, the works [4, 6, 9, 18, 19] solve the inverse kinematics of different type of robots by means of conformal geometric algebra. The idea exploited in all of them is to define different geometric entities whose intersections coincide with the origins of the frames attached to the joints. Those points allow us to recover the joint variables and, hence, the configuration or configurations associated with a predefined position and orientation of the end-effector. Finally, in [1] a numerical method also based on conformal geometric algebra is developed to solve the inverse position problem for arbitrary long serial robots.

In our previous work [13, chapter 4], we used conformal geometric algebra to develop a geometric method to cope with the inverse kinematics of serial industrial robots with a spherical wrist based on the same idea of the works cited above. Here, we will go one step further by considering robots without a spherical wrist. As stated before, the most typical case is when there is an offset between any of the last three joint axes. In that case, we cannot split the problem in the two classical subproblems, namely the inverse position problem and the inverse orientation problem (as we did in [13, chapter 4]). This is the first time, to the author's knowledge, that the inverse kinematics of this particular type of robots is addressed by means of conformal geometric algebra.

Let us suppose that the predefined position is denoted by $\boldsymbol{p} \in \mathbb{R}^3$, while the predefined orientation is denoted by $\{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$. First, we compute the null vector $p \in \mathcal{G}_{4,1}$ associated with $\boldsymbol{p}$:

$$p = H(\boldsymbol{p}) = \boldsymbol{p} + e_0 + \frac{1}{2}\boldsymbol{p}^2 e_\infty, \tag{14}$$

where $H(\cdot)$ denotes the Hestenes' embedding (as defined in section 2.1 of chapter 2 in [13]). With the three vectors $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ we define three lines, whose inner representation are:

$$\begin{aligned} \ell_{\boldsymbol{x}} &= \boldsymbol{x} I - (\boldsymbol{x} \wedge \boldsymbol{p}) I e_\infty \\ \ell_{\boldsymbol{y}} &= \boldsymbol{y} I - (\boldsymbol{y} \wedge \boldsymbol{p}) I e_\infty \\ \ell_{\boldsymbol{z}} &= \boldsymbol{z} I - (\boldsymbol{z} \wedge \boldsymbol{p}) I e_\infty, \end{aligned} \tag{15}$$

where $I = e_1 \wedge e_2 \wedge e_3$ is the pseudoscalar of $\mathcal{G}_3$. Clearly, the orientation of the end-effector is uniquely determined by any two of those lines.

Now, let us consider the serial industrial robot depicted in Fig. 4, i.e., the typical serial industrial robot but, instead of having a spherical wrist, it has an offset between the fourth and the fifth joints. In this case, we have the points p_0—the null vector representation of the origin of the base frame—and p—the null vector representation of the target position $\boldsymbol{p}$. Hence, we need to find the points p_1, p_2 and p_3.

Point p_1 is the translation of the point p_0 through the z-axis of the first joint frame, z_1, a displacement by the length of the first link, a_1. Therefore:

$$p_1 = T_{z_1} p_0 \widetilde{T}_{z_1}, \tag{16}$$

where:

$$T_{z_1} = 1 + \frac{a_1 e_\infty z_1}{2}. \tag{17}$$

Notice that, since z_1 is aligned with the first joint axis, it does not change under the action of the joint variable q_1 (in fact, it does not change under the action of any joint variable) and, thus, the base frame and the first joint frame have the same z-axis.

Analogously, point p_3 is the translation of the point p along the z-axis of the last joint, which coincides with the z-axis of the target orientation, a displacement by the length of the last link, a_6. Hence:

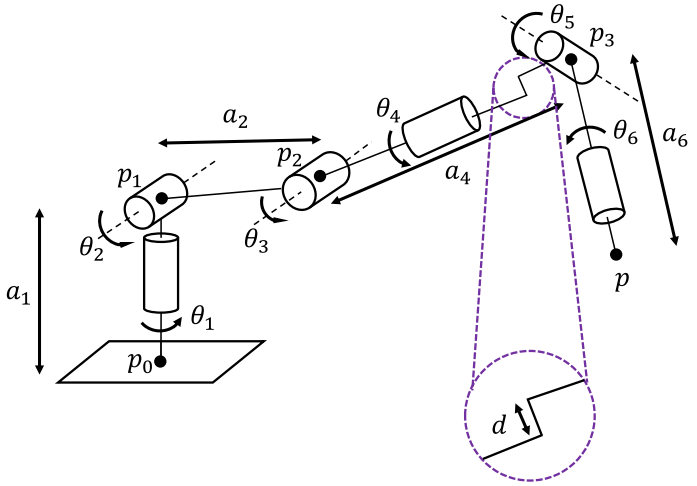


Fig. 4 Schematic representation of a serial industrial robot: the length of the i-th link is denoted by a_i, while the joint variable of the i-th joint is denoted by the angle θ_i. The value d is the length of the offset placed between the fourth and fifth joints (so this robot has not a spherical wrist). To find out the value of the unknown joint variables θ_i, the points p_i, attached to the joints, need to be computed

$$p_3 = T_z p \widetilde{T}_z, \tag{18}$$

where:

$$T_z = 1 + \frac{a_6 e_\infty z}{2}, \tag{19}$$

Finally, to compute the point p_2, we proceed as follows. We need two spheres and one plane, whose inner representations are:

$$\begin{aligned} \pi &= p_0 \wedge p_1 \wedge p_3 \wedge e_\infty \\ s_1 &= p_1 - \frac{1}{2} a_2^2 e_\infty \\ s_2 &= p_3 - \frac{1}{2}(a_4 + d)^2 e_\infty \end{aligned} \tag{20}$$

Clearly, p_2 belongs to the intersection between these three geometric objects.

To compute such a intersection we use the following definition:

Definition 5. Let O_1 and O_2 be two different geometric objects with outer representations K_1^* and K_2^*. The *meet* or *intersection* between O_1 and O_2, denoted by $K_1^* \vee K_2^*$, is defined as the multivector $K_1^* \vee K_2^* = (K_1 \wedge K_2)^* = K_1 \cdot K_2^*$.

Extended to three geometric objects with outer representations K_1^*, K_2^* and K_3^*, we have that:

$$K_1^* \vee K_2^* \vee K_3^* = (K_1 \wedge K_2 \wedge K_3)^*. \tag{21}$$

Now, since the inner representation of any plane and sphere is a grade-1 element of $\mathcal{G}_{4,1}$, $s_1 \wedge s_2 \wedge \pi$ is a grade-3 element and, as a consequence, its dual is a grade-2 element of $\mathcal{G}_{4,1}$, i.e., a bivector. Therefore:

$$B = s_1^* \vee s_2^* \vee \pi^* = (s_1 \wedge s_2 \wedge \pi)^* \tag{22}$$

is a bivector. This bivector represents a pair of points in conformal geometric algebra, so $B = b_1 \wedge b_2$ for some null vectors b_1 and b_2. It is clear that p_2 is one of these two null vectors. To extract them from B, the following equations are used [11]:

$$\begin{aligned} b_1 &= -\widetilde{P}\left((b_1 \wedge b_2) \cdot e_\infty\right) P, \\ b_2 &= P\left((b_1 \wedge b_2) \cdot e_\infty\right) \widetilde{P}, \end{aligned} \tag{23}$$

where P denotes the projector operator defined as:

$$P = \frac{1}{2}\left(1 + \frac{b_1 \wedge b_2}{|b_1 \wedge b_2|}\right). \tag{24}$$

It only remains to find the value of the joint variables. Since all joints are revolute, their joint variables are angles. First, we need to construct four auxiliary lines and two planes with the already obtained points:

$$\ell_1^* = p_0 \wedge p_1 \wedge e_\infty, \tag{25}$$
$$\ell_2^* = p_1 \wedge p_2 \wedge e_\infty, \tag{26}$$
$$\ell_3^* = p_2 \wedge p_3 \wedge e_\infty, \tag{27}$$
$$\ell_4^* = p_3 \wedge p \wedge e_\infty, \tag{28}$$
$$\pi_1 = \boldsymbol{x}_1, \tag{29}$$
$$\pi_2^* = p_2 \wedge p_2 \wedge p_3 \wedge e_\infty. \tag{30}$$

Finally, using the geometric entities defined in Eqs. (15), (20) and (25–30), the joint variables are obtained:

$$\theta_1 = \angle(\pi_1^*, \pi^*), \tag{31}$$
$$\theta_2 = \angle(\ell_1^*, \ell_2^*), \tag{32}$$
$$\theta_3 = \angle(\ell_2^*, \ell_3^*), \tag{33}$$
$$\theta_4 = \angle(\ell_z^*, \pi_2^*), \tag{34}$$
$$\theta_5 = \angle(\ell_z^*, \ell_3^*), \tag{35}$$
$$\theta_6 = \angle(\ell_{\boldsymbol{x}}^*, \ell_4^*), \tag{36}$$

where $\angle(\cdot, \cdot)$ denotes the main angle defined by two geometric entities. More precisely, for two geometric objects with outer representation K_1^* and K_2^*, it is defined by the following formula:

$$\angle(K_1^*, K_2^*) = \cos^{-1}\left(\frac{K_1^* \cdot K_2^*}{|K_1^*||K_2^*|}\right). \tag{37}$$

3 Motion Planning

We start this section by giving some basic definitions that will allow us to formulate the motion planning problem for serial industrial robots [12, 16]. Here, a serial industrial robot of n DoF is denoted by $\mathcal{R}_n$.

Definition 6. The space of all positions and orientations that the end-effector of a serial industrial robot can reach is known as the *workspace* of the robot and is denoted by $\mathcal{W}$. Clearly, $\mathcal{W} \subset \mathcal{X}$.

Definition 7. An *obstacle* O is a rigid-body object in $\mathcal{W}$.

Since obstacles are closed and bounded sets in $\mathbb{R}^3$, they are compact sets with respect to the standard topology of $\mathbb{R}^3$. Hence, every obstacle O can be covered by a finite collection of open sets $\{S_j\}_{j=1}^m$, i.e., $O \subset \bigcup_{j=1}^m S_j$. Obviously, since we are in $\mathbb{R}^3$, these open sets may be assumed to be open balls. But then, we have that:

$$O \subset \bigcup_{j=1}^{m} S_j \subset \bigcup_{j=1}^{m} \overline{S}_j, \tag{38}$$

so every obstacle O can be covered by a finite set of closed balls.

As elements of the workspace of the robot, every obstacle can be represented in the configuration space C.

Definition 8. Let $\mathcal{R}_n$ be a serial robot and $O \subset \mathcal{W}$, an obstacle. Then, as stated before, there exists a finite set of closed balls $\{\overline{S}_j\}_{j=1}^m$ such that $O \subset \bigcup_{j=1}^m \overline{S}_j$. The representation of O in the configuration space C is computed as:

$$C(O) = \{\boldsymbol{q} \in C \ : \ \mathcal{R}_n(\boldsymbol{q}) \cap \overline{S}_j \neq \emptyset \text{ for some } j = 1, \ldots, m\}, \tag{39}$$

where $\mathcal{R}_n(\boldsymbol{q})$ is the subset of $\mathcal{W}$ occupied by $\mathcal{R}_n$ in that specific configuration $\boldsymbol{q}$. Such representation is known as the *C-obstacle* or *C-obsctacle* of O.

Definition 9. Let $\mathcal{R}_n$ be a serial robot and $O_1, \ldots, O_r \subset \mathcal{W}$, r different obstacles. The *free-of-obstacles configuration space* C_{free} is:

$$C_{\text{free}} = C \setminus \bigcup_{k=1}^{r} C(O_k). \tag{40}$$

Proposition 1. *The free-of-obstacles configuration space* C_{free} *is an open set.*

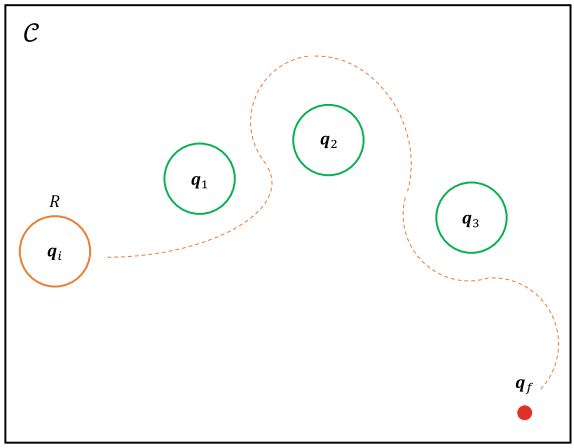


Fig. 5 Graphical example of a planar motion planning problem: the mobile platform R should move from an initial configuration $\boldsymbol{q}_i$ to a goal configuration $\boldsymbol{q}_f$ avoiding the special configurations $\boldsymbol{q}_1$ to $\boldsymbol{q}_3$ in its way. A sphere is defined around each special configuration and, then, the solution trajectory is computed so its distance to all the spheres is always higher than a given value

Proof. We have seen that, for every $k = 1, \ldots, r$, O_k is a closed set. In addition, since every serial industrial robot is a sequence of rigid-bodies, $\mathcal{R}_n(\boldsymbol{q})$ is also a closed set for every $\boldsymbol{q} \in C$ and, thus, $O_k \cap \mathcal{R}_n(\boldsymbol{q})$ is a closed set and so $C_{\text{obs}} = \bigcup_{k=1}^{r} C(O_k)$. Therefore, C_{free} is an open set.

Definition 10. The *motion planning problem* consists of finding a mapping, i.e., a path, $c : [0, 1] \to C$ such that no configuration along the path intersects an obstacle. If the map c has codomain C_{free}, the path is said to be a *free-path*.

Now, we can explore how conformal geometric algebra can be applied to this problem. In particular, it can be applied in two different ways: (1) by allowing an efficient computation of the free-of-obstacles configuration space C_{free} and (2) by improving the computation of the solution trajectories for any motion planning problem.

For the first application, the number and position of the closed balls covering each obstacle are not relevant. The only key point is the fact that they are covered by a finite set of closed balls (which is guaranteed by the compactness of the obstacles). Analogously, $\mathcal{R}_n(\boldsymbol{q})$ is also a compact set and, hence, can be covered by a finite set of closed balls. Therefore, we can rewrite Eq. (39) as:

$$C(O) = \left\{ \boldsymbol{q} \in C \ : \ \overline{C}_\ell \cap \overline{S}_j \neq \emptyset \ \text{ for } \begin{array}{l} \text{some } j = 1, \ldots, m \\ \text{some } \ell = 1, \ldots, p \end{array} \right\}, \tag{41}$$

where $\mathcal{R}_n(\boldsymbol{q}) \subset \bigcup_{\ell=1}^{p} \overline{C}_\ell$.

Now, the expression $\overline{C}_\ell \cap \overline{S}_j$ is equivalent to $(c_\ell \wedge s_j)^*$, where c_ℓ and s_j are the inner representation of the spheres defined by the boundary of $\overline{C}_\ell$ and $\overline{S}_j$ respectively. Such intersection is empty if, and only if, $(c_\ell \wedge s_j)^2 > 0$. We add the extra condition $c_\ell \cdot s_j < 0$ since we want to avoid to have one sphere contained in the other (if $c_\ell \cdot s_j > 0$, the intersection between both spheres is still empty but one is contained

in the other and, hence, it is clear that such configuration does not belong to the free-of-obstacles configuration space). Therefore, Eq. (39) can be rewritten as:

$$C(O) = \left\{ \boldsymbol{q} \in C \ : \ \begin{array}{r} (c_\ell \wedge s_j)^2 > 0 \\ c_\ell \cdot s_j < 0 \end{array} \text{ for } \begin{array}{l} \text{some } j = 1, \ldots, m \\ \text{some } \ell = 1, \ldots, p \end{array} \right\}. \tag{42}$$

This is a more efficient way of computing $C(O)$ for every possible obstacle and, as a consequence, of computing C_{free}.

With respect to the second application, we should restrict the robots we use to serial industrial robots of two or three degrees of freedom so its configuration space has dimension two or three. This is done for practical reasons as we shall see later. This may seem too restrictive, but there is a large number of serial industrial robots with few degrees of freedom. A good example are mobile manipulators, where a platform with two translational degrees of freedom and one rotational degree of freedom moves freely in a planar environment (see Fig. 5). The main methods used for computing the solution trajectories in motion planning problems can be grouped into three categories:

- Potential field methods, where a differentiable real-valued function $U : C \to \mathbb{R}$, called the potential function, is defined. Such function has an attractive component that pulls the trajectory towards the goal configuration and a repulsive component that pushes the trajectory away from the start configuration and from the obstacles.
- Sampling-based multi-query methods, where a roadmap is constructed over C_{free}. The nodes represent free-of-obstacles configurations, while the edges represent feasible local paths between those configurations. Once the roadmap is constructed, a search algorithm finds out the best solution trajectory by selecting and joining the local paths through an optimization process.
- Sampling-based single-query methods, where a tree-structure data is constructed by searching new configurations (nodes) in C_{free} and connecting them through local paths (edges). Its main difference with respect to the multi-query methods is that, while the multi-query methods work in two times (construction of the roadmap and searching of a solution trajectory), in the single-query methods both steps are taken together. Each new configuration added to the set of nodes is connected by a local path and evaluated in order to check its feasibility.

Now, we will introduce a very useful concept. Given a sphere (or a circle) S centered at $\boldsymbol{c}$ with radius $r > 0$ and a point $\boldsymbol{p}$ exterior to S, we can always construct a right triangle with the segment defined by $\boldsymbol{p}$ and $\boldsymbol{c}$ as its hypotenuse and the radius as one of its legs. Then, by the Pythagorean theorem, the other leg has a length equal to $d^2(\boldsymbol{c}, \boldsymbol{p}) - r^2$. This is known as the *tangential distance of* S *from* $\boldsymbol{p}$ and is denoted by $d_T(\boldsymbol{p}, S)$.

If s denotes the inner representation of S and p and c are the null vector representations of $\boldsymbol{p}$ and $\boldsymbol{c}$, then:

$$\begin{aligned} s \cdot p &= (c - \frac{1}{2} r^2 e_\infty) \cdot p \\ &= c \cdot p - \frac{1}{2} r^2 e_\infty \cdot p \\ &= -\frac{1}{2} d^2(\boldsymbol{c}, \boldsymbol{p}) + \frac{1}{2} r^2 \\ &= -\frac{1}{2} \left(d^2(\boldsymbol{c}, \boldsymbol{p}) - r^2 \right) \\ &= -\frac{1}{2} d_T^2(\boldsymbol{p}, S), \end{aligned} \tag{43}$$

and, thus, $d_T(\boldsymbol{p}, S) = \sqrt{-2s \cdot p}$.

For any of the three methods listed above, we can use the tangential distance d_T defined in Eq. (43) to simplify the computations and improve the overall performance. Indeed, let us first consider special configurations like, for instance, singularities—configurations where the serial industrial robot loses the ability to move in at least one direction of the operational space $\mathcal{X}$—and colliding configurations—configurations lying in the boundary between C_{free} and C_{obs}. These configurations should be taken into account when computing the solution trajectory of any motion planning problem. Then, for every special configuration $\boldsymbol{q}_0$, we define a sphere S centered at $\boldsymbol{q}_0$ and with a small random radius $r > 0$. Now, we proceed as follows:

- For a potential field method, we define the potential function so it has a repulsive component that pushes the trajectory away from these special configurations. To do so, the most efficient way is to define, for each configuration $\boldsymbol{q}_0$ of this type, a quadratic repulsive component as follows:

$$U_{\boldsymbol{q}_0}(\boldsymbol{q}) = \begin{cases} \frac{\kappa}{2} \left(\frac{1}{d_T(\boldsymbol{q}, S)} - \frac{1}{d_0} \right)^2 & \text{if } d_T(\boldsymbol{q}, S) \leq d_0 \\ 0 & \text{if } d_T(\boldsymbol{q}, S) > d_0 \end{cases} \tag{44}$$

  where s is the inner representation of S, d_0 is set as a threshold for the distance d_T and $\kappa \in \mathbb{R}$. Notice that $d_T^2(\boldsymbol{q}, S) = -2q \cdot s$, where q is the null vector representation of $\boldsymbol{q}$.
- For a sampling-based method with multiple queries, it is sufficient with removing from the roadmap those nodes associated with special configurations. During the construction of the roadmap, each configuration $\boldsymbol{q} \in C$ is evaluated to determine whether $\boldsymbol{q}$ is free-of-obstacles or not. Similarly, the idea is to evaluate each $\boldsymbol{q} \in C$ in order to determine whether $\boldsymbol{q}$ is close to a special configuration or not. To speed up the process, both evaluations can be carried out together:

  1) Select a value $d_0 > 0$ that will work as a threshold.
  2) Given a discretization of the configuration space C, each $\boldsymbol{q}$ of such discretization is evaluated to check whether:

- Is free-of-obstacles.
- Is far from any special configuration $\boldsymbol{q}_0$. This can be done simply by evaluating whether $d_T(\boldsymbol{q}, S) > d_0$ or $d_T(\boldsymbol{q}, S) \leq d_0$, where, again, s is the inner representation of the sphere S and $d_T^2(\boldsymbol{q}, S) = -2q \cdot s$.

3) If $\boldsymbol{q}$ is free-of-obstacles and far from any special configuration, then it can be added to the set of nodes of the roadmap.

- For a sampling-based method with a single query, the approach is completely analogous to the one used for methods with multiple-queries due to the similarities between both categories.

Remark 1. As stated before, we are restricting the robots we use to robots with two or three degrees of freedom. The reason stems from the fact that every method explained here works in the configuration space C of the robot. Hence, for an arbitrary serial industrial robot of n degrees of freedom, C has dimension n and any special configuration will be an n-dimensional vector $\boldsymbol{q} \in C$. Since the proposed solution is based on the computation of the tangential distance between the current robot configuration and several spheres (each one centered at one of those special configurations), we will need to compute the inner product between two grade-1 elements of $\mathcal{G}_{n+1,1}$. Theoretically speaking, there is no problem in that but, in practice, most of the current libraries, toolboxes and computer-based tools work with the conformal geometric algebra of vector spaces of small dimension, so the methods introduced here would not be able to be implemented.

4 Conclusions

In this chapter, we have dealt with three problems of serial industrial robots, namely the forward kinematics, the inverse kinematics and the motion planning problem. We have used the key elements of conformal geometric algebra to formally introduce them, as well as to develop specific tools to solve them. In particular, we have shown that given an arbitrary configuration $\boldsymbol{q} \in C$ of the robot, there exists a rotor $M \in \mathcal{G}_{4,1}$ such that $P' = M P \widetilde{M}$ is either the position or orientation of the end-effector of the robot and where P is the position/orientation of the base frame. This means that rotor M relates the base frame with the end-effector's position and orientation for that specific configuration $\boldsymbol{q}$.

In addition, we have developed a geometric method based on the definition and manipulation of several geometric entities to solve the inverse kinematics of serial industrial robots without a spherical wrist and that, together with the method introduced in [13] provides a general method to deal with the inverse kinematics of arbitrary serial robots. Notice that the two methods have some similarities, so the question on whether a more general geometric method to solve the inverse kinematics of these robots exists or not arises naturally. Or whether is there a numerical method based on conformal geometric algebra that solves this problem? We have seen that,

in [1], a numerical method is developed using conformal geometric algebra. However, since its target is to solve just the inverse position problem of arbitrary long serial robots with only revolute joints, the algorithm can only be applied to these type of robots.

Finally, we have formulated the motion planning problem as well as shown how we can compute the free-of-obstacles configuration space C_{free} by means of evaluating the intersection between spheres when represented in conformal geometric algebra. In addition, we have also seen that we can compute a solution trajectory between two configurations $\boldsymbol{q}_i, \boldsymbol{q}_f \in C_{\text{free}}$ by evaluating, in each step of the process, the distance between the sphere or spheres representing the serial robot and the spheres covering each obstacle, singularity and/or forbidden configuration. There is, however, much to do in this field. For instance, two open questions are:

- We have seen several methods to compute solution trajectories. Can any of them be improved or its performance enhanced if entirely formulated using conformal geometric algebra?
- Could a new method be defined using this mathematical framework?

References

1. Aristidou, A., Lasenby, J.: Inverse kinematics solutions using conformal geometric algebra. In: Dorst, L., Lasenby, J. (eds.) Guide to Geometric Algebra in Practice, pp. 47–62. Springer, London (2011)
2. Baker, D.R., Wampler, C.W.: Some facts concerning the inverse kinematics of redundant manipulators. In: IEEE International Conference on Robotics and Automation (ICRA), Raleigh, NC, USA, pp. 604–609, March 1987
3. Baker, D.R., Wampler, C.W.: On the inverse kinematics of redundant manipulators. Int. J. Robot. Res. **7**(2), 3–21 (1988)
4. Bayro-Corrochano, E., Zamora-Esquivel, J.: Differential and inverse kinematics of robot devices using conformal geometric algebra. Robotica **25**(1), 43–61 (2007)
5. Buss, S.R.: Introduction to inverse kinematics with Jacobian transpose, pseudoinverse and Damped Least Squares methods. Technical report, University of California, San Diego (2009)
6. Campos-Macías, L., Carbajal-Espinosa, O., Loukianov, A., Bayro-Corrochano, E.: Inverse kinematics for a 6-DOF walking humanoid robot leg. Adv. Appl. Clifford Algebras **27**(1), 581–597 (2017)
7. Doran, C., Lasenby, A.: Geometric Algebra for Physicists. Cambridge University Press, Cambridge (2003)
8. Drexler, D.A.: Solution of the closed-loop inverse kinematics algorithm using the Crank-Nicolson method. In: IEEE 14th International Symposium on Applied Machine Intelligence and Informatics (SAMI), Herlany, Slovakia, 21–23 January 2016, pp. 351–356 (2016)
9. Hildenbrand, D., Zamora, J., Bayro-Corrochano, E.: Inverse kinematics computation in computer graphics and robotics using conformal geometric algebra. Adv. Appl. Clifford Algebras **18**(3), 699–713 (2008)
10. Huang, L.S., Jiang, R.K.: A new method of inverse kinematics solution for industrial 7DoF robot. In: 32nd Chinese Control Conference (CCC), Xi'an, China, 26–28 July 2013, pp. 6063–6065 (2013)
11. Lasenby, A., Lasenby, J., Wareham, R.: A convariant approach to geometry using geometric algebra. Technical report, Department of Engineering, University of Cambridge (2004)

12. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge (2006)
13. Lavor, C., Xambó-Descamps, S., Zaplana, I.: A Geometric Algebra Invitation to Space-Time Physics, Robotics and Molecular Geometry. SBMAC/SpringerBriefs. Springer, Cham (2018)
14. Paul, R.P.: Robot Manipulators: Mathematics, Programming and Control. The MIT Press, Cambridge (1981)
15. Pieper, D.L.: The kinematics of manipulation under computer control. Ph.D. thesis, Stanford Artificial Intelligence Laboratory - Stanford University (1968)
16. Siciliano, B., Sciavicco, L., Villani, L., Oriolo, G.: Robotics: Modelling, Planning and Control. Springer, London (2008)
17. Spong, M.W., Hutchinson, S., Vidyasagar, M.: Robot Modeling and Control. Wiley, Hoboken (2006)
18. Tørdal, S.S., Hovland, G., Tyapin, I.: Efficient implementation of inverse kinematics on a 6-DOF industrial robot using conformal geometric algebra. Adv. Appl. Clifford Algebras **27**(3), 2067–2082 (2017)
19. Zamora, J., Bayro-Corrochano, E.: Inverse kinematics, fixation and grasping using conformal geometric algebra. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Sendai, Japan, 28 September–2 October 2004, vol. 4, pp. 3841–3846 (2004)

Recent Advances on Oriented Conformal Geometric Algebra Applied to Molecular Distance Geometry

Carlile Lavor and Rafael Alves

Abstract Oriented Conformal Geometric Algebra was recently applied to Molecular Distance Geometry, where we want to determine 3D protein structures using distance information provided by Nuclear Magnetic Resonance experiments. We present new results that simplify the associated calculations.

1 Introduction

Distance Geometry (DG) deals with calculation of points using distances between some of them [30, 31]. One of the most important applications of DG is related to the determination of 3D protein structures using distance data from Nuclear Magnetic Resonance (NMR) experiments [11, 27, 34, 40]. For other applications and more information about DG, see [4, 5, 32, 33, 36].

Given a graph $G = (V, E, d)$, where V represents the set of atoms of a molecule and E is the set of atom pairs for which a distance is available (defined by $d : E \rightarrow (0, \infty)$), the *Molecular Distance Geometry Problem* (MDGP) is to find a function $x : V \rightarrow \mathbb{R}^3$ that associates each element of V with a point in $\mathbb{R}^3$ in such a way that the Euclidean distances between the points correspond to the values given by d [30].

Information from protein geometry and NMR data allow us to solve the MDGP using a combinatorial method, called *Branch-and-Prune* (BP) [7, 29]. The discrete MDGP is called the *Discretizable MDGP* (DMDGP), which is based on a vertex order $v_1, ..., v_n \in V$, also given as input of the problem [8, 15, 25, 26, 35]. Formally, the DMDGP is defined as follows [19, 20] (we denote x_i instead of $x(v_i)$ and $d_{i,j}$ instead of $d(v_i, v_j)$):

Definition 1 Given a simple undirected graph $G = (V, E, d)$, whose edges are weighted by $d : E \rightarrow (0, \infty)$, and a vertex order $v_1, ..., v_n \in V$, such that

C. Lavor (✉)
University of Campinas, 13081 -970 Campinas, SP, Brazil
e-mail: clavor@unicamp.br

R. Alves
Federal University of ABC, 09606-070 Sao Bernardo do Campo, SP, Brazil
e-mail: alves.rafael@ufabc.edu.br

S. Xambó-Descamps (ed.), *Systems, Patterns and Data Engineering with Geometric Calculi*, SEMA SIMAI Springer Series 13, https://doi.org/10.1007/978-3-030-74486-1_2

1. For $v_1, v_2, v_3 \in V$, there exist $x_1, x_2, x_3 \in \mathbb{R}^3$ satisfying the given distances;
2. For $i > 3$,

$$\{\{v_{i-3}, v_i\}, \{v_{i-2}, v_i\}, \{v_{i-1}, v_i\}\} \subset E \tag{1}$$

and

$$d_{i-3,i-2} + d_{i-2,i-1} > d_{i-3,i-1}, \tag{2}$$

find a function $x : V \to \mathbb{R}^3$ satisfying

$$\forall \{v_i, v_j\} \in E, \ ||x_i - x_j|| = d_{i,j}. \tag{3}$$

Property 1 avoids solutions modulo rotations and translations and Property 2 allows us to calculate the two possible positions for v_4. For each position for v_4, we have other two for v_5, and so on, implying that the DMDGP search space is finite [20].

For $i > 3$, we may also have $\{v_j, v_i\} \in E$, $j < i - 3$, adding another equation ($||x_i - x_j|| = d_{j,i}$) to the system related to v_i:

$$\begin{aligned} ||x_i - x_{i-1}|| &= d_{i-1,i}, \\ ||x_i - x_{i-2}|| &= d_{i-2,i}, \\ ||x_i - x_{i-3}|| &= d_{i-3,i}. \end{aligned} \tag{4}$$

We obtain a unique solution x_i^* for v_i, supposing $||x_i^* - x_j|| = d_{j,i}$ and that the points $x_{i-1}, x_{i-2}, x_{i-3}, x_j \in \mathbb{R}^3$ are not in the same plane. If both possible positions for v_i are unfeasible with respect to additional distances $d_{j,i}$, $j < i - 3$, it is necessary to consider the other possible position for v_{i-1} and repeat the procedure. Essentially, this is what the BP algorithm does [20].

Since distances $d_{i-1,i}$ and $d_{i-2,i}$ are related to bond lengths and bond angles of a protein, they can be considered precise values. This may not happen to distances $d_{j,i}$, $j \leq i - 3$, since they may be provided by NMR experiments [28, 40]. In [21], distances $d_{j,i}$ are represented as *interval distances* $[\underline{d}_{j,i}, \overline{d}_{j,i}]$, $\underline{d}_{j,i} \leq d_{j,i} \leq \overline{d}_{j,i}$, and an extension of the BP algorithm, called *i*BP, is proposed. The idea is to sample values from intervals $[\underline{d}_{i-3,i}, \overline{d}_{i-3,i}]$ in order to solve the associated system to calculate positions to v_i, which implies that, choosing many values, the search space increases exponentially, and for small samples, a solution may not be found [1, 9, 10, 16, 37, 39].

Geometrically, even considering interval distances, solving systems like (4) is to intersect spheres and spherical shells centered at the positions for vertices $v_{i-1}, v_{i-2}, v_{i-3}, v_j$ ($j < i - 3$) with radius $d_{i-1,i}, d_{i-2,i}, [\underline{d}_{i-3,i}, \overline{d}_{i-3,i}], [\underline{d}_{j,i}, \overline{d}_{j,i}]$, respectively.

In [2, 3], Conformal Geometric Algebra (CGA) was used to model uncertainties in the DMDGP, for the *branching phase* of *i*BP (intersection of two spheres and a spherical shell), and, in [24], the *Oriented* CGA (OCGA) [6] was applied for the *pruning phase* of *i*BP (when additional spherical shells must be considered in the intersection). Another CGA approach is discussed in [13].

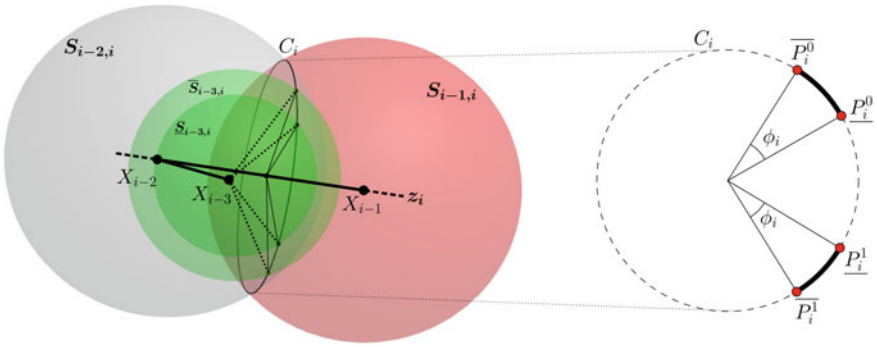


Fig. 1 Intersection of two spheres and a spherical shell resulting in two arcs

We present new results that simplify the calculations in the pruning phase of *i*BP algorithm.

Next section explains how CGA and OCGA replace the classical approach for solving the DMDGP and Sect. 3 provides our contribution.

2 Conformal Geometric Algebra (CGA) and Oriented CGA (OCGA) for the *i*BP Algorithm

2.1 CGA for Branching

Replacing $d_{i-3,i} \in \mathbb{R}$ by interval distance $[\underline{d}_{i-3,i}, \overline{d}_{i-3,i}]$ in (4), we have to intersect two spheres with one spherical shell resulting in two arcs, instead of two points in $\mathbb{R}^3$ (see Fig. 1).

The points $\underline{P_i^0}$, $\underline{P_i^1}$ and $\overline{P_i^0}$, $\overline{P_i^1}$, from the intersection of spheres centered at the positions for v_{i-1}, v_{i-2}, v_{i-3} with radius $d_{i-1,i}$, $d_{i-2,i}$, $\underline{d}_{i-3,i}$ and $d_{i-1,i}$, $d_{i-2,i}$, $\overline{d}_{i-3,i}$, respectively (see Fig. 1), can be obtained from the *point pairs* generated by

$$S_{i-1,i} \wedge S_{i-2,i} \wedge \underline{S}_{i-3,i} \text{ and } S_{i-1,i} \wedge S_{i-2,i} \wedge \overline{S}_{i-3,i},$$

where underline and overline indicate the use of $\underline{d}_{i-3,i}$ and $\overline{d}_{i-3,i}$, respectively [2] (in the conformal model, $S_{i,j}$ is the sphere centered at the position for vertex v_i with radius $d_{i,j}$).

With the starting and the ending point of an arc, we can define a *rotor*, for $i \geq 4$, given by

$$R_i = \cos\left(\frac{\lambda_i}{2}\right) - \sin\left(\frac{\lambda_i}{2}\right) z_i^*, \tag{5}$$

where $\lambda_i \in [0,\ \phi_i]$ is the rotation angle corresponding to the arcs $\underline{P_i^0}\overline{P_i^0}$ and $\underline{P_i^1}\overline{P_i^1}$ (see Fig. 1), z_i^* is the *dual* of z_i, and

$$z_i = X_{i-2} \wedge X_{i-1} \wedge e_\infty,$$

since the rotation axis of R_i is defined by X_{i-2} and X_{i-1} (the positions for v_{i-2} and v_{i-1} in the conformal model). Note that ϕ_i, $i \geq 4$, can be computed *apriori* based on the given intervals $[\underline{d}_{i-3,i}, \overline{d}_{i-3,i}]$ and the DMDGP definition [2].

In order to consider the effect of changing points in the arcs (to avoid the sampling process), the rotation axes of the rotors must be replaced by (see [3])

$$z_i = (R_i \cdots R_4)\left(\underline{P_{i-2}^b} \wedge \underline{P_{i-1}^b} \wedge e_\infty\right)\left(R_4^{-1} \cdots R_i^{-1}\right),$$

implying that

$$X_i^b(\lambda_4, \ldots, \lambda_i) = (R_i \cdots R_4)\,\underline{P_i^b}\left(R_4^{-1} \cdots R_i^{-1}\right),$$

for $i \geq 4$. The values $b \in \{0, 1\}$ are defined when *i*BP chooses one of the branches in the search tree [3].

Note that, when all distances $d_{i-3,i}$ are precise values, *i.e.* $\phi_i = 0$ for $i \geq 4$,

$$X_i^b(0, \ldots, 0) = \underline{P_i^b} = \overline{P_i^b}.$$

2.2 OCGA for Pruning

During the pruning phase of *i*BP, when additional spherical shells must be considered, the arc orientation is even more important. In [24], this was done using the Oriented Conformal Geometric Algebra (OCGA) [6], which is an extension of the Oriented Projective Geometry [38].

First, we define an orientation for the circle obtained from the intersection $S_{i-1,i} \wedge S_{i-2,i}$ (Fig. 2), given by

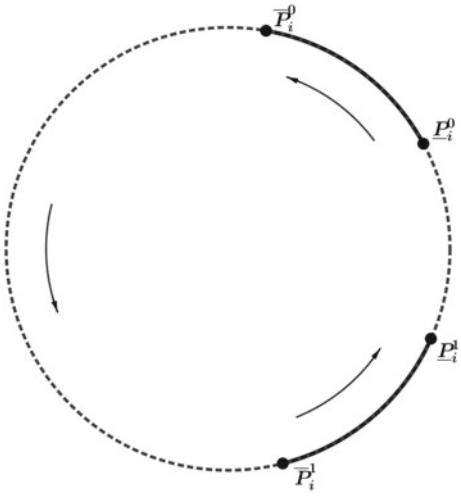
$$C_i = \underline{P_i^0} \wedge \overline{P_i^0} \wedge \overline{P_i^1}.$$

Since C_i is a trivector in the conformal space, its dual C_i^* is a bivector orthogonal to the plane that contains the circle C_i, which implies that the line

$$C_i^* \wedge e_\infty$$

is oriented according to C_i.

Fig. 2 Oriented circle C_i



Using the normalized bivector dual to the rotation axis $C_i^* \wedge e_\infty$, we can define z_i^* in a different way, that carries the orientation of C_i. The associated rotor R_i is now defined by:

$$R_i = \cos\left(\frac{\lambda_i}{2}\right) - \sin\left(\frac{\lambda_i}{2}\right) z_i^*, \ 0 \leq \lambda_i \leq \phi_i,$$

where

$$z_i = \frac{C_i^* \wedge e_\infty}{||C_i^* \wedge e_\infty||}.$$

Supposing that for v_i, $i > 4$, there is a pruning edge $\{v_j, v_i\} \in E$, $j < i - 3$, with interval distance $[\underline{d}_{j,i}, \overline{d}_{j,i}]$, and denoting by $\underline{P_j^0 P_j^0}$ and $\underline{P_j^1 P_j^1}$ the arcs obtained from the intersections $S_{i-1,i} \wedge S_{i-2,i} \wedge \underline{S}_{j,i}$ and $S_{i-1,i} \wedge S_{i-2,i} \wedge \overline{S}_{j,i}$, we compute the new starting and ending points of the associated rotors doing the following calculations [24] (all the possible cases are illustrated in Fig. 3):

$$t_1 = (\underline{P_j^0} \wedge \overline{P_i^0} \wedge \overline{P_i^1})C_i, \, t_2 = (\underline{P_i^0} \wedge \underline{P_j^0} \wedge \overline{P_i^1})C_i$$

and

$$t_1 = (\overline{P_j^0} \wedge \overline{P_i^0} \wedge \overline{P_i^1})C_i, \, t_2 = (\underline{P_i^0} \wedge \overline{P_j^0} \wedge \overline{P_i^1})C_i,$$

where

$$C_i = \underline{P_i^0} \wedge \overline{P_i^0} \wedge \overline{P_i^1}.$$

The same procedure must be done for $\underline{P_j^1}$ and $\overline{P_j^1}$.

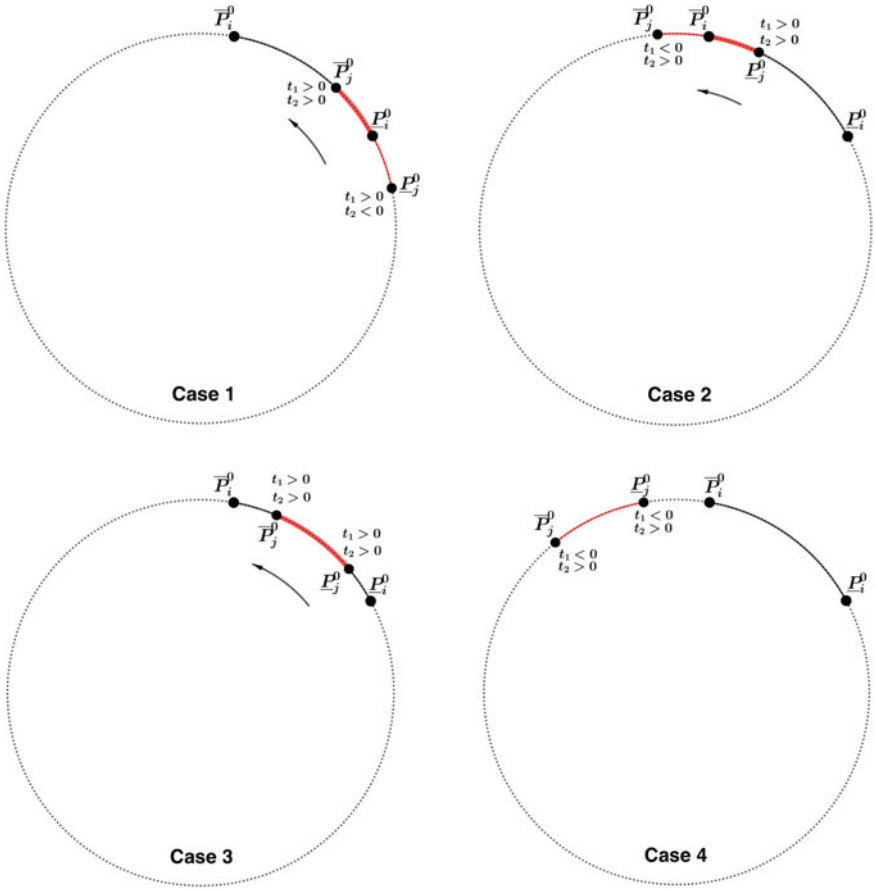


Fig. 3 Possible cases for intersecting arcs

3 New Approach for *i*BP Prunning Phase

Considering a circle C, obtained from the intersection of spheres S_1 and S_2, we obtain

$$C = S_1 \wedge S_2 \Rightarrow \left(\frac{C}{||C||}\right)^* = \frac{S_1^* \wedge S_2^*}{||S_1^* \wedge S_2^*||}.$$

Since C can also be defined by three points $P_1,\ P_2,\ P_3 \in C$, let us suppose that

$$\left(\frac{P_1 \wedge P_2 \wedge P_3}{||P_1 \wedge P_2 \wedge P_3||}\right)^* = \frac{S_1^* \wedge S_2^*}{||S_1^* \wedge S_2^*||}.$$

For other three points $Q_1,\ Q_2,\ Q_3 \in C$, but with opposite orientation, we have

$$\left(\frac{Q_1 \wedge Q_2 \wedge Q_3}{||Q_1 \wedge Q_2 \wedge Q_3||}\right)^* = -\left(\frac{P_1 \wedge P_2 \wedge P_3}{||P_1 \wedge P_2 \wedge P_3||}\right)^*.$$

Thus, for any three distinct points in C, given by $S_1 \wedge S_2$, the expression $\left(\frac{C}{||C||}\right)^*$ is constant (up to a sign $\pm$). This implies that to distinguish the orientation of two trivectors that define C, it is enough to check the signs of some fixed coordinate ($\neq 0$) of the trivectors.

For example, for points $P_1,\ P_2,\ P_3 \in C$, with

$$\begin{aligned}
P_1 &= e_0 + x_1e_1 + x_2e_2 + x_3e_3 + \frac{1}{2}||(x_1, x_2, x_3)||^2e_\infty,\\
P_2 &= e_0 + y_1e_1 + y_2e_2 + y_3e_3 + \frac{1}{2}||(y_1, y_2, y_3)||^2e_\infty,\\
P_3 &= e_0 + z_1e_1 + z_2e_2 + z_3e_3 + \frac{1}{2}||(z_1, z_2, z_3)||^2e_\infty,
\end{aligned}$$

and choosing the coordinate of

$$e_1 \wedge e_2 \wedge e_0$$

of the trivector $P_1 \wedge P_2 \wedge P_3$, we have to calcultate

$$x_1y_2 - x_2y_1 - (x_1 - y_1)z_2 + (x_2 - y_2)z_1. \tag{6}$$

From Sect. 2, in order to know the position of $\underline{P_j^0}$ (obtained from the intersection $S_{i-1,i} \wedge S_{i-2,i} \wedge \underline{S}_{j,i}$), in terms of arc $\underline{P_i^0}\overline{P_i^0}$, we have to calculate

$$t_1 = (\underline{P_j^0} \wedge \overline{P_i^0} \wedge \overline{P_i^1})C_i \text{ and } t_2 = (\underline{P_i^0} \wedge \underline{P_j^0} \wedge \overline{P_i^1})C_i.$$

Using the new idea, in addition to avoid two trivector products, we just calculate (and compare the signs) expressions like (6): one for $\underline{P_j^0} \wedge \overline{P_i^0} \wedge \overline{P_i^1}$ and another for $\underline{P_i^0} \wedge \underline{P_j^0} \wedge \overline{P_i^1}$.

Without loss of generality, let us suppose that the sign of expression (6), associated to the points $\underline{P_j^0}, \overline{P_i^0}, \overline{P_i^1}$, is positive, denoted by

$$\left[\underline{P_j^0}, \overline{P_i^0}, \overline{P_i^1}\right] > 0.$$

So, all the cases illustrated in Fig. 3 are given, respectively, by

- Case 1:

$$\left[\underline{P_j^0}, \overline{P_i^0}, \overline{P_i^1}\right] > 0$$
$$\left[\underline{P_i^0}, \underline{P_j^0}, \overline{P_i^1}\right] < 0$$
$$\left[\overline{P_j^0}, \overline{P_i^0}, \overline{P_i^1}\right] > 0$$
$$\left[\underline{P_i^0}, \overline{P_j^0}, \overline{P_i^1}\right] > 0$$

- Case 2:

$$\left[\underline{P_j^0}, \overline{P_i^0}, \overline{P_i^1}\right] > 0$$
$$\left[\underline{P_i^0}, \underline{P_j^0}, \overline{P_i^1}\right] > 0$$
$$\left[\overline{P_j^0}, \overline{P_i^0}, \overline{P_i^1}\right] < 0$$
$$\left[\underline{P_i^0}, \overline{P_j^0}, \overline{P_i^1}\right] > 0$$

- Case 3:

$$\left[\underline{P_j^0}, \overline{P_i^0}, \overline{P_i^1}\right] > 0$$
$$\left[\underline{P_i^0}, \underline{P_j^0}, \overline{P_i^1}\right] > 0$$
$$\left[\overline{P_j^0}, \overline{P_i^0}, \overline{P_i^1}\right] > 0$$
$$\left[\underline{P_i^0}, \overline{P_j^0}, \overline{P_i^1}\right] > 0$$

- Case 4:

$$\left[\underline{P_j^0}, \overline{P_i^0}, \overline{P_i^1}\right] < 0$$
$$\left[\underline{P_i^0}, \underline{P_j^0}, \overline{P_i^1}\right] > 0$$
$$\left[\overline{P_j^0}, \overline{P_i^0}, \overline{P_i^1}\right] < 0$$
$$\left[\underline{P_i^0}, \overline{P_j^0}, \overline{P_i^1}\right] > 0$$

3.1 Example

Let us consider the same example given in [24], that is, a DMDGP instance with the following data (all the calculations were done using GAALOP [18]):

$$\begin{aligned} d_{i-1,i} &= 1, i = 2, ..., 6, \\ d_{i-2,i} &= \sqrt{3}, i = 3, ..., 6, \\ d_{1,4} &= 2.15, d_{2,5} \in [2.20, 2.60], d_{3,6} \in [2.40, 2.60], \\ d_{1,5} &\in [2.45, 2.55]. \end{aligned}$$

Since $d_{1,4}$ is also a precise value, we can fix the first four points, given by

$$x_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, x_2 = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}, x_3 = \begin{bmatrix} -1.5 \\ \frac{\sqrt{3}}{2} \\ 0 \end{bmatrix}, x_4 = \begin{bmatrix} -1.311 \\ 1.552 \\ 0.702 \end{bmatrix}.$$

From the intersections of spheres $\underline{S}_{2,5} \wedge S_{3,5} \wedge S_{4,5}$ and $\overline{S}_{2,5} \wedge S_{3,5} \wedge S_{4,5}$, we obtain the arcs $\underline{P_5^0}\overline{P_5^0}$ and $\underline{P_5^1}\overline{P_5^1}$, defined by the points

$$\begin{aligned} \underline{P_5^0} &= e_0 - 0.409e_1 + 1.981e_2 + 0.753e_3 + 2.329e_\infty, \\ \underline{P_5^1} &= e_0 - 1.502e_1 + 1.350e_2 + 1.663e_3 + 3.422e_\infty, \\ \overline{P_5^0} &= e_0 - 1.386e_1 + 2.525e_2 + 0.484e_3 + 4.266e_\infty, \\ \overline{P_5^1} &= e_0 - 2.046e_1 + 2.144e_2 + 1.033e_3 + 4.966e_\infty. \end{aligned}$$

Using the interval distance $d_{1,5}$, we calculate $\underline{S}_{1,5} \wedge S_{3,5} \wedge S_{4,5}$ and $\overline{S}_{1,5} \wedge S_{3,5} \wedge S_{4,5}$, giving the points

$$\begin{aligned} \underline{A_5^0} &= e_0 - 0.674e_1 + 2.299e_2 + 0.513e_3 + 3.001e_\infty, \\ \underline{A_5^1} &= e_0 - 1.260e_1 + 1.283e_2 + 1.664e_3 + 3.001e_\infty, \\ \overline{A_5^0} &= e_0 - 0.795e_1 + 2.377e_2 + 0.470e_3 + 3.251e_\infty, \\ \overline{A_5^1} &= e_0 - 1.407e_1 + 1.317e_2 + 1.670e_3 + 3.251e_\infty. \end{aligned}$$

With the orientation of $C_5 = S_{3,5} \wedge S_{4,5}$ defined by

$$C_5 = \underline{P_5^0} \wedge \overline{P_5^0} \wedge \overline{P_5^1},$$

the calculations necessary to test if $\underline{A_5^0} \in \underline{P_5^0}\overline{P_5^0}$, using the original strategy, are the following ($e_{ij} = e_i \wedge e_j$ and $E = e_\infty \wedge \overline{e_0}$):

$$
\begin{aligned}
C_5 &= +0.732e_{12} \wedge e_0 - 0.715e_{13} \wedge e_0 + 0.197e_{23} \wedge e_0 - 1.634e_{13} \wedge e_\infty \\
&\quad -2.830e_{23} \wedge e_\infty + 0.634e_1 \wedge E + 1.098e_2 \wedge E - 1.243e_3 \wedge E + 1.887e_{123}, \\
\underline{A_5^0} \wedge \overline{P_5^0} \wedge \overline{P_5^1} &= +0.421e_{12} \wedge e_0 - 0.411e_{13} \wedge e_0 + 0.113e_{23} \wedge e_0 - 0.940e_{13} \wedge e_\infty \\
&\quad -1.628e_{23} \wedge e_\infty + 0.365e_1 \wedge E + 0.631e_2 \wedge E - 0.715e_3 \wedge E + 1.085e_{123}, \\
\underline{P_5^0} \wedge \underline{A_5^0} \wedge \overline{P_5^1} &= +0.478e_{12} \wedge e_0 - 0.467e_{13} \wedge e_0 + 0.128e_{23} \wedge e_0 - 1.067e_{13} \wedge e_\infty \\
&\quad -1.848e_{23} \wedge e_\infty + 0.414e_1 \wedge E + 0.717e_2 \wedge E - 0.811e_3 \wedge E + 1.232e_{123}
\end{aligned}
$$

and

$$
\begin{aligned}
\left(\underline{A_5^0} \wedge \overline{P_5^0} \wedge \overline{P_5^1}\right) C_5 &= 0.468, \\
\left(\underline{P_5^0} \wedge \underline{A_5^0} \wedge \overline{P_5^1}\right) C_5 &= 0.531.
\end{aligned}
$$

However, using the proposed approach, we just calculate the coordinates of $e_{12} \wedge e_0$, using (6):

$$
\begin{aligned}
\left[\underline{A_5^0}, \overline{P_5^0}, \overline{P_5^1}\right] &= (-0.674)(2.525) - (2.299)(-1.386) \\
&\quad -(-0.674 + 1.386)(2.144) + (2.299 - 2.525)(-2.046) \\
&= 0.421,
\end{aligned}
$$

$$
\begin{aligned}
\left[\underline{P_5^0}, \underline{A_5^0}, \overline{P_5^1}\right] &= (-0.409)(2.299) - (1.981)(-0.674) \\
&\quad -(-0.409 + 0.674)(2.144) + (1.981 - 2.299)(-2.046) \\
&= 0.478.
\end{aligned}
$$

Since $\left[\underline{A_5^0}, \overline{P_5^0}, \overline{P_5^1}\right] > 0$ and $\left[\underline{P_5^0}, \underline{A_5^0}, \overline{P_5^1}\right] > 0$, we conclude that

$$
\underline{A_5^0} \in \underline{P_5^0}\overline{P_5^0}.
$$

4 Conclusion and Acknowledgements

The first mathematical relationship between Distance Geometry and Geometric Algebra was established by Dress and Havel, in 1993 [14]. Since 2015 [22], the connection between the DMDGP and Geometric Algebra has been studied, becoming part of the relevant and challenging applications of Conformal Geometric Algebra [12, 17, 23].

We would like to thank Sebastià Xambó for the organization of the minisymposium "Systems, Patterns and Data Engineering with Geometric Calculi", at the

ICIAM 2019, and to encourage us to write this work. We are also grateful to the Brazilian research agencies CNPq and FAPESP for the support and to the reviewer's comments.

References

1. Agra, A., Figueiredo, R., Lavor, C., Maculan, N., Pereira, A., Requejo, C.: Feasibility check for the distance geometry problem: an application to molecular conformations. Int. Trans. Oper. Res. **24**, 1023–1040 (2017)
2. Alves, R., Lavor, C.: Geometric algebra to model uncertainties in the discretizable molecular distance geometry problem. Adv. Appl. Clifford Algebra **27**, 439–452 (2017)
3. Alves, R., Lavor, C., Souza, C., Souza, M.: Clifford algebra and discretizable distance geometry. Math. Methods Appl. Sci. **41**, 3999–4346 (2018)
4. Billinge, S.J., Duxbury, P.M., Gon calves, D.S., Lavor, C., Mucherino, A.: Assigned and unassigned distance geometry: applications to biological molecules and nanostructures. *4OR* **14**, 337–376 (2016)
5. Billinge, S., Duxbury, P., Gonçalves, D., Lavor, C., Mucherino, A.: Recent results on assigned and unassigned distance geometry with applications to protein molecules and nanostructures. Ann. Oper. Res. **271**, 161–203 (2018)
6. Cameron, J., Lasenby, J.: Oriented conformal geometric algebra. Adv. Appl. Clifford Algebra **18**, 523–538 (2008)
7. Carvalho, R., Lavor, C., Protti, F.: Extending the geometric build-up algorithm for the molecular distance geometry problem. Inf. Process. Lett. **108**, 234–237 (2008)
8. Cassioli, A., Gunluk, O., Lavor, C., Liberti, L.: Discretization vertex orders in distance geometry. Disc. Appl. Math. **197**, 27–41 (2015)
9. Cassioli, A., Bardiaux, B., Bouvier, G., Mucherino, A., Alves, R., Liberti, L., Nilges, M., Lavor, C., Malliavin, T.E: An algorithm to enumerate all possible protein conformations verifying a set of distance constraints. BMC Bioinformatics **16**, 16–23 (2015)
10. Costa, T., Bouwmeester, H., Lodwick, W., Lavor, C.: Calculating the possible conformations arising from uncertainty in the molecular distance geometry problem using constraint interval analysis. Inf. Sci. **415–416**, 41–52 (2017)
11. Crippen, G., Havel, T.: Distance Geometry and Molecular Conformation. Wiley (1988)
12. Dorst, L., Fontijne, D., Mann, S.: Geometric Algebra for Computer Science: An Object-Oriented Approach to Geometry. Morgan Kaufman (2007)
13. Dorst, L.: Boolean combination of circular arcs using orthogonal spheres. Adv. Appl. Clifford Algebra **29**, 1–21 (2019)
14. Dress, A., Havel, T.: Distance geometry and geometric algebra. Found. Phys. **23**, 1357–1374 (1993)
15. Gonçalves, D., Mucherino, A.: Discretization orders and efficient computation of Cartesian coordinates for distance geometry. Optim. Lett. **8**, 2111–2125 (2014)
16. Gonçalves, D., Mucherino, A., Lavor, C., Liberti, L.: Recent advances on the interval distance geometry problem. J. Global Optim. **69**, 525–545 (2017)
17. Hestenes, D.: Old wine in new bottles: a new algebraic framework for computational geometry. In: Advances in Geometric Algebra with Applications in Science and Engineering, Corrochano, E., Sobczyk, G. (eds.), Birkhäuser, pp. 1–14 (2001)
18. Hildenbrand, D.: Foundations of Geometric Algebra Computing. Springer (2012)
19. Lavor, C., Liberti, L., Maculan, N., Mucherino, A.: Recent advances on the discretizable molecular distance geometry problem. Eur. J. Oper. Res. **219**, 698–706 (2012)
20. Lavor, C., Liberti, L., Maculan, N., Mucherino, A.: The discretizable molecular distance geometry problem. Comput. Optim. Appl. **52**, 115–146 (2012)

21. Lavor, C., Liberti, L., Mucherino, A.: The interval BP algorithm for the discretizable molecular distance geometry problem with interval data. J. Global Optim. **56**, 855–871 (2013)
22. Lavor, C., Alves, R., Figueiredo, W., Petraglia, A., Maculan, N.: Clifford algebra and the discretizable molecular distance geometry problem. Adv. Appl. Clifford Algebra **25**, 925–942 (2015)
23. Lavor, C., Xambó-Descamps, S., Zaplana, I.: A Geometric Algebra Invitation to Space-Time Physics Robotics and Molecular Geometry. SpringerBriefs in Mathematics, Springer (2018)
24. Lavor, C., Alves, R.: Oriented conformal geometric algebra and the molecular distance geometry problem. Adv. Appl. Clifford Algebra **29**, 1–19 (2019)
25. Lavor, C., Liberti, L., Donald, B., Worley, B., Bardiaux, B., Malliavin, T., Nilges, M.: Minimal NMR distance information for rigidity of protein graphs. Discrete Appl. Math. **256**, 91–104 (2019)
26. Lavor, C., Souza, M., Mariano, L., Liberti, L.: On the polinomiality of finding ^{K}DMDGP re-orders. Discrete Appl. Math. **267**, 190–194 (2019)
27. Lavor, C., Alves, R., Souza, M., Aragon, J.: NMR protein structure calculation and sphere intersections. Comput. Math. Biophys. **8**, 89–101 (2020)
28. Lavor, C., Souza, M., Carvalho, L., Gonçalves, D., Mucherino, A.: Improving the sampling process in the interval branch-and-prune algorithm for the discretizable molecular distance geometry problem. Appl. Math. Comput. **389**, 125586–125597 (2021)
29. Liberti, L., Lavor, C., Maculan, N.: A branch-and-prune algorithm for the molecular distance geometry problem. Int. Trans. Oper. Res. **15**, 1–17 (2008)
30. Liberti, L., Lavor, C., Maculan, N., Mucherino, A.: Euclidean distance geometry and applications. SIAM Rev. **56**, 3–69 (2014)
31. Liberti, L., Lavor, C.: Six mathematical gems from the history of distance geometry. Int. Trans. Oper. Res. **23**, 897–920 (2016)
32. Liberti, L., Lavor, C.: Euclidean Distance Geometry: An Introduction. Springer (2017)
33. Liberti, L., Lavor, C.: Open research areas in distance geometry. In: Open Problems in Optimization and Data Analysis, Migalas, A., Pardalos, P. (eds.), Springer, pp. 183–223 (2018)
34. Malliavin, T., Mucherino, A., Lavor, C., Liberti, L.: Systematic exploration of protein conformational space using a distance geometry approach. J. Chem. Inf. Model. **59**, 4486–4503 (2019)
35. Mucherino, A., Lavor, C., Liberti, L.: The discretizable distance geometry problem. Optimization Lett. **6**, 1671–1686 (2012)
36. Mucherino, A., Lavor, C., Liberti, L., Maculan, N.: Distance Geometry: Theory, Methods, and Applications. Springer (2013)
37. Souza, M., Lavor, C., Muritiba, A., Maculan, N.: Solving the molecular distance geometry problem with inaccurate distance data. BMC Bioinformatics **14**, S71–S76 (2013)
38. Stolfi, J.: Oriented Projective Geometry - A Framework for Geometric Computations. Academic Press (1991)
39. Worley, B., Delhommel, F., Cordier, F., Malliavin, T., Bardiaux, B., Wolff, N., Nilges, M., Lavor, C., Liberti, L.: Tuning interval branch-and-prune for protein structure determination. J. Global Optim. **72**, 109–127 (2018)
40. Wütrich, K.: Protein structure determination in solution by nuclear magnetic resonance spectroscopy. Science **243**, 45–50 (1989)

Geometric Calculus Applications to Medical Imaging: Status and Perspectives

Silvia Franchini and Salvatore Vitabile

Abstract Medical imaging data coming from different acquisition modalities requires automatic tools to extract useful information and support clinicians in the formulation of accurate diagnoses. Geometric Calculus (GC) offers a powerful mathematical and computational model for the development of effective medical imaging algorithms. The practical use of GC-based methods in medical imaging requires fast and efficient implementations to meet real-time processing constraints as well as accuracy and robustness requirements. The purpose of this article is to present the state of the art of the GC-based techniques for medical image analysis and processing. The use of GC-based paradigms in Radiomics and Deep Learning, i.e. a comprehensive quantification of tumor phenotypes by applying a large number of quantitative image features and its classification, is also outlined.

1 Introduction

In the last few decades, Geometric Calculus (GC) has attracted growing attention in many research fields as a universal and comprehensive mathematical language for expressing and elaborating geometric concepts. GC, often referred to as Geometric Algebra (GA) or Clifford Algebra (CA), integrates and reformulates in a single unified framework several mathematical models and theories including complex numbers, quaternions, matrix algebra, vector algebra, spinor and tensor calculus, and differential forms [1]. GC offers powerful mathematical and computational models that allow for the formulation of effective algorithms in a large spectrum of application domains, such as computer graphics, computer vision, robotics, and image analysis.

Several applications of GC in the field of medical image analysis and processing have been recently proposed. The great amount of available medical imaging data coming from different modalities, such as Computer Tomography (CT), Magnetic

S. Franchini (✉) · S. Vitabile
Department of Biomedicine, Neuroscience and Advanced Diagnostics, University of Palermo, Via del Vespro, 129, 90127 Palermo, Italy
e-mail: silvia.franchini@unipa.it

S. Xambó-Descamps (ed.), *Systems, Patterns and Data Engineering with Geometric Calculi*, SEMA SIMAI Springer Series 13, https://doi.org/10.1007/978-3-030-74486-1_3

Resonance (MR), and Positron Emission Tomography (PET), requires effective automatic techniques and tools to extract useful information in order to assist clinicians in diagnosing accuracy. Medical data needs to be validated and certified by medical experts before being processed by the automated computing tools. Target applications are automatic detection of relevant regions (tissues, tumors, lesions, anatomical areas of interest), segmentation, shape extraction, classification, 3D modeling and reconstruction, volume registration. Image segmentation is often the first step in image processing and analysis since it allows for partitioning the image into multiple regions or areas based on similar pixel characteristics, such as color, shape, or texture. In medical imaging, automatic segmentation is useful to detect and label pixels in an image or voxels in a 3D volume that represent a tumor within a patient's organ. Another key task in computer-aided diagnosis is medical image classification that consists in extracting a set of features from the image and using a trainable classifier to assign one or more labels to the image. The availability of 3D models of the anatomical areas of interest (organs, bones, tumors, lesions, etc.) is important since they allow for a better visualization and understanding of the situation. The reconstruction of a 3D shape starting from a sequence of 2D scans and their boundary points consists in extracting and display a 3D surface composed of geometric primitives that approximate the object of interest. Medical image registration (or alignment) consists in finding the proper geometric transformation to align two misaligned 2D images or 3D surfaces captured in different moments or by different acquisition modalities. The registration process is needed to compare or integrate medical data derived from different measurements. In clinical practice, medical image registration has many important applications including: 1) aligning temporal sequences of images to compensate for motion of the patient between scans, 2) combining images of the same subject from different imaging modalities (the multi-modality fusion facilitates diagnosis by integrating information acquired by diverse imaging devices, as CT and MR), 3) image guidance during interventions, 4) aligning images from the same subject or from multiple patients in longitudinal studies, which investigate temporal changes in anatomical structures (regions of interest, organs, cancers, etc.). The powerful representation and computation models provided by the GC framework have been used to develop effective medical imaging methods for pathology diagnosis and prognostication.

This paper presents an overview of the state of the art of GC-based medical imaging techniques and outlines future directions on the application of GC in this relevant application domain.

The use of Conformal Geometric Algebra (CGA), namely, five-dimensional (5D) Clifford algebra, has been proposed as an efficient tool to develop several medical imaging techniques including segmentation, 3D surface reconstruction, and registration of medical images [2–6]. These algorithms massively use geometric operations, such as reflections, rotations, translations, and dilations, so that they can exploit the efficient formulation of these operations provided by the CGA framework.

Furthermore, GA entities and operators allow for a more effective handling of multi-dimensional images, such as multi-channel or multi-spectral images, which are frequently used in computer-aided medical diagnosis for automatic tumor detection,

segmentation, and classification. The use of GA multivectors allows images to be represented with the different channels defined as a single entity, rather than separate quantities so as to preserve the correlations among channels and avoid information loss. Several GA-based methods to process multi-spectral and multi-channel medical images have been recently reported in the literature, including Clifford convolution and Clifford Fourier transform for multi-spectral medical image segmentation [7–12], quaternionic Gabor filters for medical image texture segmentation [13], Clifford neural networks [14–17] and Clifford Support Vector Machines [20, 21] for medical image classification, and GA-based algorithms, such as GA-SIFT [29], GA-SURF [30], and GA-ORB [31], for feature extraction in multi-spectral medical images.

The practical use of GA-based methods in medical imaging requires fast and efficient implementations to meet specific constraints such as the real-time processing of large amounts of medical data or the required accuracy and precision. To this aim, different specialized hardware architectures able to directly support GA elements and operators have been designed to accelerate GA-based medical imaging algorithms [5, 12, 32].

Future directions in the application of GC for medical imaging can be envisaged in the integration and reformulation of two advanced medical imaging techniques, such as Radiomics [51] and Deep Learning [57], exploiting the new powerful GC-based computation paradigms for the quantification and classification of tumor phenotypes based on the efficient extraction and processing of a large number of quantitative image features.

The rest of the paper is organized as follows: Sect. 2 presents the state of the art of medical imaging methods based on GC including CGA-based algorithms for segmentation, 3D modeling and registration of medical data as well as GA-based techniques for feature extraction, segmentation, and classification of multi-dimensional medical images. Section 3 summarizes the hardware implementations of GC-based medical imaging algorithms reported in literature. Section 4 outlines future perspectives on the application of GC in the medical imaging field focusing on the use of GC-based paradigms in Radiomics and Deep Learning. Finally, conclusions are contained in Sect. 5.

2 Applications of Geometric Calculus to Medical Image Processing: State of the Art

In the last few years, several research works have proposed the use of Geometric Calculus (GC) for medical image analysis and processing. The following sections present an overview covering various GC-based medical imaging methods.

2.1 Conformal Geometric Algebra for Medical Image Segmentation, 3D Modeling and Registration

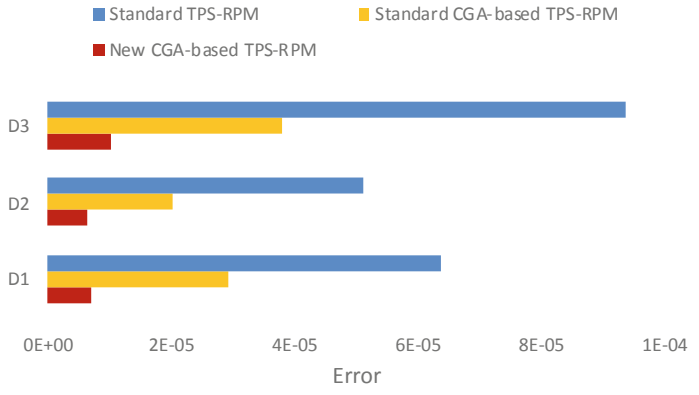
Several medical imaging tasks, such as segmentation, shape extraction, 3D surface modeling, and registration (alignment) of medical images, involve complex geometric operations, which are usually solved by using classical geometry based on standard vector algebra and matrix calculations.

In the last few years, however, different works have proposed the use of Conformal Geometric Algebra (CGA) to reformulate, in a more effective way, several medical imaging algorithms. CGA, a.k.a. five-dimensional (5D) Geometric or Clifford algebra, is a 5D representation of three-dimensional (3D) geometry that allows for a simple and efficient modeling of 3D space objects and transformations. In CGA, 5D algebraic elements can be directly used to represent 3D geometric entities, while conformal (angle-preserving) geometric transformations (reflections, rotations, translations, dilations) are obtained by universal operators, called versors, which are directly applied to the geometric objects to be transformed.

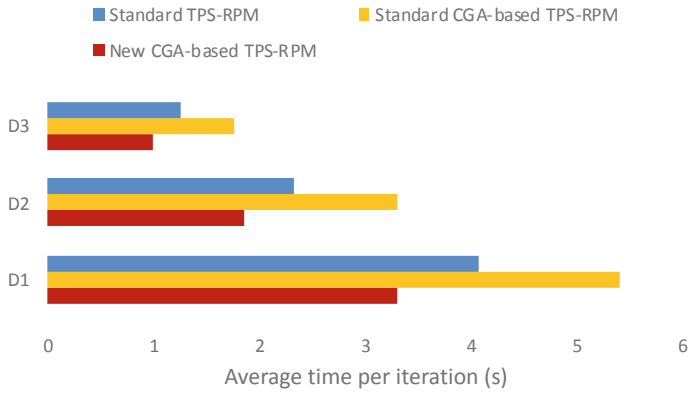
Recent works have proposed new CGA-based approaches for the segmentation, 3D modeling and registration of medical data. A medical image segmentation method based on gradient information and CGA-based self-organizing neural networks has been proposed by J. Rivera-Rovelo and E. Bayro-Corrochano [2]. The algorithm uses the Generalized Gradient Vector Flow (GGVF) to extract the contour points of the object of interest, which are then used as the training set of a Growing Neural Gas (GNG) network having CGA translators as weights. After the training stage, the GNG neurons will contain the proper CGA translators, which will be applied to the centroid of the contour points to translate it to the object contour and will define the shape of the object of interest. This method has been also extended for the approximation of 3D surfaces [3] and tested with several medical datasets, including CT and MR images. Another work has proposed the use of CGA for the 3D rendering and registration of CT and MR sequences [4]. The authors have reformulated the standard Marching Cubes rendering algorithm to reconstruct a 3D model based on spheres defined in the CGA framework. Furthermore, the Thin-Plate Spline Robust Point Matching (TPS-RPM) algorithm has been adapted for the registration of two misaligned 3D models based on CGA spheres derived from two misaligned sequences of CT or MR medical images.

However, compute-intensive medical imaging tasks require the analysis and processing of massive volumes of medical data. Fast and efficient computing techniques are needed to meet real-time processing constraints as well as result precision and accuracy requirements. To this aim, a novel simplified formulation of CGA operations has been introduced in [5] to reduce computation times and accelerate the CGA-based algorithms so as to make the CGA-based methods practically usable for clinical practice. The reformulated CGA operators have been used to re-design a suite of CGA-based medical imaging automatic techniques [6]. The proposed medical image processing chain includes three stages: segmentation of the 2D image sequence (CT or MR), 3D surface reconstruction, and registration of two misaligned 3D models.

Experimental results show that reformulated CGA-based medical imaging methods lead to shorter computation times as well as to higher precision results with respect to both standard CGA algorithms and traditional medical imaging methods. Figure 1 shows the results obtained when the registration method TPS-RPM, reformulated according the new formulation of CGA, is compared, in terms of both error and execution times, with both the traditional TPS-RPM method and the TPS-RPM method based on the standard formulation of CGA. As it can be observed, the new CGA-based TPS-RPM algorithm shows better results than the traditional TPS-RPM algorithm in terms of both precision and computation speed.



(a)



(b)

Fig. 1 Comparison between three different registration methods: standard TPS-RPM, standard CGA-based TPS-RPM and new CGA-based TPS-RPM in terms of (a) error and (b) execution times for three different medical datasets (D1: MR brain, D2: CT brain tumor, D3: CT liver lesion)

2.2 Geometric Algebra for Multi-dimensional Medical Image Processing

Multi-dimensional medical images (multi-channel images, multi-spectral images) are effectively used for several tasks, such as automatic tumor detection, segmentation and classification. Medical images acquired by different imaging modalities are often combined since the multi-modality fusion facilitates the diagnosing accuracy by integrating information derived from different imaging devices (CT, MR, PET). Classical medical imaging methods treat multi-dimensional data as multiple independent data (for multi-channel images, each channel is processed separately), which can lead to information loss since the correlations among multiple dimensions are ignored. Conversely, GA defines multivectors that can be used to represent multi-dimensional data so preserving correlations among different dimensions and avoiding information loss. The following subsections summarize a suite of GA-based techniques for feature extraction, segmentation, and classification of multi-dimensional medical images.

2.2.1 Clifford Fourier Transform for GA-based Multi-dimensional Image Analysis

The classical Fourier transform is applied to scalar fields. Clifford Fourier transform represents an extension of standard Fourier transform to vector fields [7–9]. The geometric product or Clifford product defined in the Clifford algebra framework is used to generalize the concepts of convolution and Fourier transform to vector fields.

These concepts have been used to formulate new edge detection methods for color images [10, 11]. Usually, traditional edge detection methods, used for gray-scale images, are extended to color images and applied to the three color channels separately. This approach however leads to increased computational load and long execution times. In the GA-based methods, conversely, color value triples are treated as multivectors and Clifford convolution and Clifford Fourier transform are used for edge detection. Color value triples are converted in luminance and chrominance components. A standard gray-scale edge detection method is applied to the luminance component, while Clifford convolution with proper vector-value filter masks is used to detect edges in the chrominance part.

The edge detection algorithm proposed in [10] uses the Sobel technique to extract edges in the luminance component, while a single fixed value is used as the thresholding value.

A novel, enhanced version of this algorithm is proposed in [11], where the Canny algorithm is applied to the gray-scale component of the image, while a thresholding with hysteresis is applied to both the gray-scale and color components of the image. As reported in [11], the GA-based algorithm achieves a detection performance

comparable to classical edge detection methods allowing at the same time for a significant reduction of computational times in comparison with classical component-wise Canny color edge detection method.

The GA-based method has been also successfully applied for feature extraction of multi-spectral MR images [12]. In particular, as reported in Fig. 2, experimental tests have been performed on brain MR images in three different bands (PD-weighted, T1-weighted, and T2-weighted, respectively). The Clifford edge detector has been

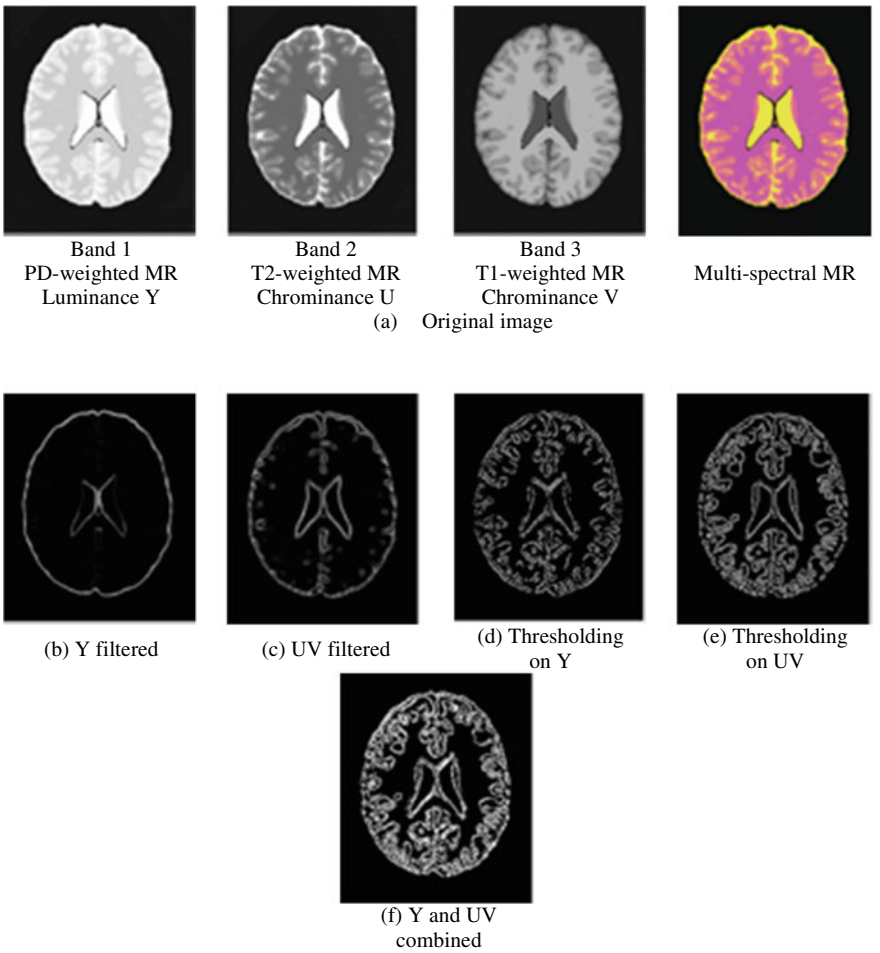


Fig. 2 The figure [12] shows three MR brain images in three different bands. Bands 1, 2 and 3 are PD-weighted, T2-weighted and T1-weighted, respectively. Clifford edge detector has been applied to the multi-spectral MR image using band 1 as luminance component Y and bands 2 and 3 as chrominance components U and V, respectively. It can be observed that processing the multi-spectral image (Fig. 2(f)) adds further details to the simple edge detection performed on the single-spectrum image (Fig. 2(d))

applied to multi-spectral MR images using band 1 (PD-weighted) as luminance component (Y) and bands 2 and 3 (T2-weighted and T1-weighted) as chrominance components (UV), respectively. Experimental results show that the GA-based method applied to multi-spectral MR images achieves a better performance in terms of precision with respect to the classical edge detection algorithm applied to the single-spectrum images.

2.2.2 Quaternionic Gabor Filters for Medical Image Texture Segmentation

The use of quaternionic Gabor filters for the classification of local image structure was proposed by T. Bulow and G. Sommer in [13]. Quaternionic Gabor filters are defined as Gaussian windowed kernels of the quaternionic Fourier transform. Based on these filters, the above-mentioned work introduces a novel quaternionic phase concept for 2D signals. The local quaternionic phase is computed from the response of the quaternionic Gabor filter. The authors show that it exists a specific relation between the local quaternionic phase and the 2D local structure. Different values of the local phase correspond to different local structures (such as lines or step edges) that can be therefore discriminated from their local phase.

The paper shows that the additional phase value resulting from the quaternionic filtering provides new information that is not provided by the phase when estimated with two complex Gabor filters. The local image structure can be therefore classified by the value of the local quaternionic phase. Quaternionic Gabor filters can be applied for medical image texture segmentation and image matching.

2.2.3 Clifford Neural Networks and Clifford Support Vector Machines for Medical Image Classification

Several applications of Clifford algebra in the field of neural computing and machine learning, including Clifford neural networks and Clifford Support Vector Machines, have been proposed. Various studies have been performed and different approaches have been presented to extend classical real-valued neural networks to higher-dimension numbers, such as complex numbers [14].

Clifford algebras provide a higher-dimension generalization of the complex numbers. Clifford neural networks extend classical neural networks to Clifford numbers or multivectors, including not only complex numbers, but also quaternions, and, in general, hypercomplex numbers [15–17]. In these works, new back error propagation algorithms are introduced for multi-layered neural networks with Clifford valued weight and activation values. The use of Clifford neurons and Clifford multi-layer perceptrons has been demonstrated to give a better performance when compared with standard real-valued neural networks. Clifford neural networks have applications in pattern recognition and classification.

Interesting approaches have been also proposed to extend Support Vector Machines (SVM) in the GA context. SVM have been originally designed for binary classification. Different solutions have been proposed to extend this method to multi-class classification. Most of these solutions combine multiple binary SVM classifiers to obtain a multiclass SVM classifier. Such an approach is used in both one-vs-all and one-vs-one algorithms [18]. Conversely, other solutions consider all the classes at once in a single large optimization problem [19]. These one-step methods, however, present higher computational complexity. A more performant solution consists in extending classical SVM methods to the GA space. In [20, 21], the authors introduce Clifford Support Vector Machines (CSVM) by redesigning classical real-valued SVM kernels in the Clifford algebra framework. In Clifford SVM, optimization variables are redefined as multivectors, multivector values are used as inputs and outputs so that CSVM can be applied to solve multiclass classification tasks. According to the Clifford algebra dimension, multiple classes can be represented. Geometric product or Clifford product is used to design CSVM kernels for non-linear classification. In multiclass classification tasks, using a single CSVM kernel instead of multiple real-valued SVM kernels can significantly reduce the computational load. Clifford SVM algorithms include complex, quaternion, and hypercomplex SVM algorithms. Clifford SVM methods have a large spectrum of applications including pattern recognition, signal and image processing, robotics, and computer vision. In the medical imaging field, SVM classifiers are used to classify medical images and detect the presence/absence of a specific pathology [22, 23], while multiclass SVM classifiers are needed when we have not only to detect the presence of a specific disease, but also to estimate its activity according to multiple classes or levels [24]. Clifford SVM can be an effective solution for the multi-level classification and grading of different pathologies since they can guarantee higher result precision as well as reduced computational complexity with respect to the standard multiclass classification methods.

2.2.4 GA-Based Feature Extraction Methods

Image feature extraction algorithms are often applied as an important basis for different medical image processing tasks, such as image segmentation, registration, classification, and matching. In particular, image interest point detectors, such as SIFT (Scale Invariant Feature Transform) [25], SURF (Speeded Up Robust Features) [26, 27], and ORB (Oriented Fast and Rotated Brief) [28], are used to detect and describe image local features that are invariant to image scaling, orientation, translation, rotation, and illumination changes and robust to local geometric distortion. However, most of these methods are designed to handle gray-scale images and cannot be applied directly to extract features from multi-dimensional images. As an example, multi-channel images are usually converted into gray-scale images before being processed by these algorithms. This conversion leads to information loss since the correlations among channels are not considered. To simultaneously capture relations among multiple channels and maintain the performance of these methods also for multi-spectral and multi-channel images, different approaches have been proposed

that reformulate image feature extraction algorithms to operate in the GA framework and treat directly multi-dimensional images. The GA-SIFT method is the reformulation of the standard SIFT algorithm for multi-spectral images [29]. Exploiting the GA theory, the authors propose a new representation of multi-spectral images including spatial and spectral information. The feature points are detected and described in the GA space. Comparison results reported in the paper show that the GA-SIFT method outperforms some previously reported SIFT algorithms in the feature extraction from multi-spectral images. Another algorithm that uses the GA theory to extract features from multi-spectral images is the GA-SURF [30]. A Hessian matrix based on GA is calculated for locating interest points in both spatial and spectral spaces. The calculation of the Hessian matrix is simplified by using box filters based on GA, while interest points are extracted according to the procedures of SURF and described in the GA mathematical context. The paper demonstrates that the GA-SURF method is faster and more robust than other existing feature extraction algorithms for multi-spectral images. In [31], the authors propose a new version of the ORB algorithm based on the GA theory for multi-spectral images. Multi-spectral images are represented as GA multivectors and each spectral channel of multispectral images is mapped to each blade of GA. For each 2D coordinate of the multispectral image, the image data of the i-th band is mapped to the i-th blade of the associated multivector. The scale information for multi-spectral images in both spectral and spatial domains is derived in the GA space, where the inherent spectral structures of the images can be successfully retained, while the image interest points are extracted and described in the GA framework. Experimental results show that the GA-ORB algorithm is faster when compared with the standard ORB method as well as with the GA-SIFT and GA-SURF algorithms.

3 Hardware Implementations of GA-Based Image Processing Applications

Geometric Algebra offers a powerful computing tool that allows for a simple and elegant representation of complex mathematical operations in high-dimensional spaces. The high symbolic representation power of GA allows for high-dimensional objects and their complex transformations to be modeled in a direct, intuitive, and universal way. However, the low symbolic complexity of GA is accompanied by its high numeric complexity, which requires to handle calculations on high-dimensional elements involving a great number of numerical coefficients. This computational complexity is especially significant in compute-intensive applications, such as image processing and medical imaging algorithms, which demand the real-time processing of massive volumes of image data. For these reasons, GA-based algorithms need to be accelerated by using specialized hardware architectures able to directly support GA elements and operators. Different hardware coprocessors have been reported to accelerate GA-based image processing applications.

The coprocessor presented in [5] is conceived to support a suite of CGA-based medical imaging algorithms, including segmentation, 3D modeling and registration of medical data, which require the execution of a large amount of rigid body motion operations, namely, rotations, translations, and uniform scaling. The paper first introduces a novel, simplified, and hardware-oriented formulation of these operations and then presents a specialized coprocessor properly designed to accelerate CGA-based geometric operations by using advanced parallelism techniques, such as pipelining. A Field Programmable Gate Array (FPGA) prototyping board has been used for the implementation of the proposed coprocessor in the form of a System on Programmable Chip (SoPC), while experimental tests have been executed on several CT and MR medical datasets. Experimental results show average speedups of about one order of magnitude with respect to the execution on a conventional general-purpose processor.

In [32], the authors present an application-specific hardware architecture. The proposed coprocessor is specifically designed to accelerate GA-based color edge detection methods. The specialized architecture has been implemented on a custom Application Specific Integrated Circuit (ASIC). The coprocessor supports the 3D GA vector rotation operations required in the color image edge detection algorithm. Experimental results show significant speedups of the proposed GA coprocessor with respect to existing software implementations of GA.

Another hardware architecture has been proposed in [12] to accelerate a GA-based edge detection algorithm that can be used to process multi-channel images, including not only color images, but also multi-spectral medical images. The proposed edge detection method exploits the generalized convolution operator and the generalized Fourier transform for vector fields, which are based on the geometric product of vectors given in the Clifford algebra framework. Using this GA-based approach, the three channels of color images or multispectral medical images are treated as a single entity rather than as separated entities as in the traditional image processing algorithms. The proposed hardware architecture has been prototyped on an FPGA device. Experimental tests executed on the FPGA prototype show that the coprocessor allows for an average speedup ranging between $6\times$ and $18\times$ for different image sizes against the execution on a standard general-purpose CPU.

4 Future Directions

4.1 Multi-channel Medical Imaging

Several works show that multi-channel imaging is very effective in prostate cancer detection [33–35], in breast cancer detection [36], in brain cancer detection [37].

The great amount of available medical imaging data coming from different modalities can be combined to improve the accuracy in automatic cancer detection, segmentation, and classification in order to assist doctors in diagnosing accuracy.

In the above scenario, a principal role is played by image data representation dealing with multi-channel medical imaging. As shown before [11, 12], GC can be used to represent a three-channel imaging data allowing for algorithms dealing with the whole image set rather than with each single image. As result, a very effective segmentation algorithm is obtained. In image processing, quaternions have been used to represent color images [38, 39], color sensitive filtering [40], edge detection in color images [41, 42], and cross correlation of color images [43]. Quaternions representation can be adapted to three-channel medical imaging creating a framework to develop powerful cancer segmentation and detection systems. In [44] a new model for image processing tasks, such as image reconstruction and image denoising, of multi-channel images is proposed. The model uses the octonion algebra for the representation of multi-spectral images with up to 7 color channels. The same schema can be used for representing seven-channel medical images dealing with the previously described medical scenarios.

4.2 Radiomics

Radiomics is defined as the high-throughput extraction and analysis of large amounts of quantitative features from medical imaging data to characterize tumor phenotypes [51]. Nowadays, there are two main applications of radiomics: the classification of lesions/nodules (diagnostic) or prognostication of established cancer (theragnostic). Radiomics features are divided in two wide classes: "semantic" and "agnostic" features [45]. The "semantic" features are used to describe qualitative morphological aspect of lesions such as size, shape, location, vascularity, speculation, necrosis, and attachments or lepidics [46]. The "agnostic" features refer to invisibly quantitative description of lesions heterogeneity such as textures, histogram, wavelets, Laplacian transforms, Minkowski functionals, and fractal dimensions. Textures can be also obtained through first, second, and high-order statistical methods [47–50].

GC can improve radiomics features' number and quality introducing new operators for feature extraction in medical image. The use of hypercomplex Fourier transforms [52–54], quaternionic Gabor filters [55], and Clifford wavelet filters [56] can be explored to select powerful features for cancer diagnostic and theragnostic.

In addition, the high number and complexity of the "radiomics space", suggest innovative approaches for features representation, separation, and classification. Geometrical transformations, defined in the context of GC, as well as dimensionality reduction techniques [23] can help features processing efficiency and results by providing efficient algebraic methods for manipulating geometrical data as well as flexible nonlinear functions operating in high dimensional spaces.

4.3 Deep Learning

Deep Learning (DL) models are composed of many layers transforming input image data in classification results. With more details, DL models are composed of one or more convolutional layers for image feature detection and extraction and one or more fully connected layers for data classification. Generally, a convolutional layer consists of neurons connected to the input images or to the previous layer outputs extracting image features while scanning through an image [57].

Following our analysis in Radiomics, several new image filters, i.e. feature detectors, can be developed for qualitative and quantitative feature extraction. At each layer, the convolution output (feature map learned from that layer) becomes the input for the next convolutional layer, and so on. GC can be also used to replace standard scalar methods, such as convolution, Fourier transform, wavelet transform with more powerful GA-based methods, such as Clifford convolution, Clifford Fourier transform, Geometric Algebra wavelet in each layer of the Convolutional Neural Network. Its use allows for extending deep learning methods from scalar data to vector data involving multi-channel and multi-spectral images.

As pointed out before, the output feature maps of the final convolution layer are connected to one or more fully connected layers through learnable weights. Usually, the final fully connected layer has the same number of output nodes and classes [57]. GC has been used to generate new neural network models and new Support Vector Machine models [15, 20, 21, 58]. As an extension, GC can be used to generate new fully connected layers dealing with vector data for new and more powerful Deep Learning models.

5 Conclusions

Medical images contain powerful and unexplored hidden information. Radiomics represents a new discipline for dealing with that information. On the other hand, Deep Learning models are attracting more and more researchers for their classification and, in some cases, forecasting results. They are complementary disciplines and can be easily integrated in very powerful models in pathology diagnosis and prognostication. Geometric Calculus is able to support each single layer development aspect as well as model integrations and developments by providing the necessary support towards the personalized medicine.

References

1. Hestenes, D., Sobczyk, G.: Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics. Kluwer, Dordrecht, The Netherlands (1987)
2. Rivera-Rovelo, J., Bayro-Corrochano, E.: Medical image segmentation using a self-organizing neural network and Clifford geometric algebra. Int. Joint Conf. Neural Netw. IJCNN **2006**, 3538–3545 (2006)
3. Rivera-Rovelo, J., Bayro-Corrochano, E.: Surface approximation using growing self-organizing nets and gradient information. Appl. Bionics Biomech. **4**(3), 125–136 (2007)
4. Bayro-Corrochano, E., Rivera-Rovelo, J.: The use of geometric algebra for 3D modeling and registration of medical data. J. Math. Imaging Vis. **34**(1), 48–60 (2009)
5. Franchini, S., Gentile, A., Sorbello, F., Vassallo, G., Vitabile, S.: ConformalALU: a conformal geometric algebra coprocessor for medical image processing. IEEE Trans. Comput. **64**(4), 955–970 (2015)
6. Franchini, S., Gentile, A., Vassallo, G., Vitabile, S.: Implementation and evaluation of medical imaging techniques based on conformal geometric algebra. Int. J. Appl. Math. Comput. Sci. (IJAMCS) **30**(3), 415–433 (2020)
7. Hitzer, E.: The Clifford Fourier transform in real Clifford algebras. J. Fourier Anal. Appl. **2**(3), 669–681 (2013)
8. Ebling, J., Scheuermann, G.: Clifford convolution and pattern matching on vector fields. Proc. IEEE Visual. **2003**, 193–200 (2003)
9. Ebling, J., Scheuermann, G.: Clifford fourier transform on vector fields. IEEE Trans. Vis. Comput. Graph. **11**(4), 469–479 (2005)
10. Schlemmer, M., Hagen, H., Holtz, I., Hamann, B.: Clifford pattern matching for color image edge detection, in visualization of large and unstructured data sets. GI Edition Lecture Notes in Informatics (LNI), vol. S-4, pp. 47–58 (2006)
11. Franchini, S., Gentile, A., Vassallo, G., Vitabile, S., Sorbello, F.: Clifford algebra based edge detector for color images. In: Proceedings 6th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS-2012), Palermo, Italia, July 2012, pp. 84–91 (2012). https://doi.org/10.1109/CISIS.2012.128
12. Franchini, S., Gentile, A., Vassallo, G., Vitabile, S., Sorbello, F.: A specialized architecture for color image edge detection based on clifford algebra. In: Proceedings 7th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS-2013), Taichung, Taiwan, July 2013, pp. 128–135 (2013). https://doi.org/10.1109/CISIS.2013.29.
13. Bulow, T., Sommer, G.: Quaternionic Gabor filters for local structure classification. In: Proceedings Fourteenth International Conference on Pattern Recognition (Cat. No. 98EX170), Brisbane, Queensland, Australia, 1998, vol.1, pp. 808–810 (1998). https://doi.org/10.1109/ICPR.1998.711271.
14. Hirose, A.: Complex-Valued Neural Networks. Studies in Computational Intelligence, Springer (2012)
15. Buchholz, S., Sommer, G.: Clifford algebra multilayer perceptrons. In: Sommer, G. (eds.) Geometric Computing with Clifford Algebras. Springer, Heidelberg (2001)
16. Buchholz, S., Sommer, G.: On Clifford neurons and Clifford multi-layer perceptrons. Neural Netw. **21**(7), 925–935 (2008)
17. Pearson, J.K., Bisset, D.L.: Neural networks in the Clifford domain, Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN 1994), Orlando, FL, USA, 1994, vol.3, pp. 1465–1469 (1994). https://doi.org/10.1109/ICNN.1994.374502.
18. Deng, N., et al.: Support Vector Machines: Optimization Based Theory, Algorithms and Extensions. Chapman & Hall /CRC. Data mining and Knowledge Discovery Series (2012)
19. Weston, J., Watkins, C.: Multi-class support vector machines. In: Verleysen, M. (eds.) Proceedings of ESANN99, pp. 219–224, Brussels. D. Facto Press (1999)
20. Bayro-Corrochano, E.J., Arana, N., Vallejo, R.: Design of kernels for support multivector machines involving the Clifford geometric product and the conformal geometric neuron. In: International joint conference on neural networks, pp. 2893–2898. IEEE (2003)

21. Bayro-Corrochano, E.J., Arana-Daniel, N.: Clifford support vector machines for classification, regression, and recurrence. IEEE Trans. Neural Net. **21**(11), 1731–1746 (2010)
22. Zhang, Y., Wang, S., Ji, G., Dong, Z.: An MR brain images classifier system via particle swarm optimization and Kernel support vector machine. Sci. World J. **2013**, 9 (2013)
23. Franchini, S., Terranova, M.C., Lo Re, G., Salerno, S., Midiri, M., Vitabile, S.: Evaluation of a support vector machine based method for Crohn's disease classification. In: Esposito, A., Faundez-Zanuy, M., Morabito, F.C., Pasero, E. (eds.) Neural Approaches to Dynamics of Signal Exchanges, pp. 313–327. Springer, Singapore (2020). https://doi.org/10.1007/978-981-13-8950-4_29
24. Franchini, S., Terranova, M.C., Lo Re, G., Galia, M., Salerno, S., Midiri, M., Vitabile, S.: A novel system for multi-level Crohn's disease classification and grading based on a multiclass support vector machine. In: Esposito, A., Faundez-Zanuy, M., Morabito, F.C., Pasero, E. (eds.) Progresses in Artificial Intelligence and Neural Systems, pp. 185–197. Springer, Singapore (2021). https://doi.org/10.1007/978-981-15-5093-5_18
25. Lowe, D.G.: Object recognition from local scale-invariant features. Int. Conf. Comput. Vis. **2**(1999), 1150–1157 (1999)
26. Bay, H., Tuytelaars, T., Van Gool, L.: SURF: speeded up robust features. In: European conference on computer vision, pp. 404–417 (2006)
27. Insalaco M., Bruno A., Farruggia A., Vitabile S., Ardizzone E.: An unsupervised method for suspicious regions detection in mammogram images. In: Proceedings of the International Conference on Pattern Recognition Applications and Methods (ICPRAM-2015), SCITEPRESS Lda, pp. 302–308 (2015). https://doi.org/10.5220/0005277103020308 ISBN: 978–989–758–077–2
28. Rublee, E., Rabaud, V., Konolige, K., Bradski, G.: ORB: an efficient alternative to SIFT or SURF. In: IEEE International Conference on Computer Vision (ICCV) (2011)
29. Li, Y., Liu, W., Li, X., Huang, Q., Li, X.: GA-SIFT: a new scale invariant feature transform for multispectral image using geometric algebra. Inf. Sci. **281**, 559–572 (2014)
30. Wang, R., Shi, Y., Cao, W.: GA-SURF: a new speeded-up robust feature extraction algorithm for multispectral images based on geometric algebra. Pattern Recogn. Lett. **127**, 11–17 (2019)
31. Wang, R., Zhang, W., Shi, Y., Wang, X., Cao, W.: GA-ORB: a new efficient feature extraction algorithm for multispectral images based on geometric algebra. IEEE Access **7**, 71235–71244 (2019). https://doi.org/10.1109/ACCESS.2019.2918813
32. Mishra, B., Wilson, P., Wilcock, R.: A geometric algebra coprocessor for color edge detection. Electronics **4**(1), 94–117 (2015)
33. Chan, I., Wells 3rd., W., Mulkern, R.V., Haker, S., Zhang, J., Zou, K.H., Maier, S.E., Tempany, C.M.: Detection of prostate cancer by integration of line-scan diffusion, T2-mapping and T2-weighted magnetic resonance imaging; a multichannel statistical classifier. Med. Phys. **30**(9), 2390–2398 (2003). https://doi.org/10.1118/1.1593633
34. Bae, H., Kim, S.-S., Lee, S., Song, H., Lee, S., Koh, D., Kim, J.G., Jung, D.C.: Development of a multi-channel NIRS-USG hybrid imaging system for detecting prostate cancer and improving the accuracy of imaging-based diagnosis: a phantom study. Ultrasonography **38**(2), 143–148 (2019). https://doi.org/10.14366/usg.18030
35. Rundo, L., Han, C., Nagano, Y., Zhang, J., Hataya, R., Militello, C., Tangherloni, A., Nobile, M.S., Ferretti, C., Besozzi, D., Gilardi, M.C., Vitabile, S., Mauri, G., Nakayama, H., Cazzaniga, P.: USE-Net: incorporating squeeze-and-excitation blocks into U-net for prostate zonal segmentation of multi-institutional MRI datasets. Neurocomputing **365**, 31–43 (2019). https://doi.org/10.1016/j.neucom.2019.07.006
36. Yin, X.X., Hadjiloucas, S., Chen, J.H., Zhang, Y., Wu, J.L., Su, M.Y.. Tensor based multichannel reconstruction for breast tumours identification from DCE-MRIs. PLoS One **12**(3), e0172111 (2017). https://doi.org/10.1371/journal.pone.0172111.
37. Banerjee, S., Sushmita Mitra, B., Shankar, U., Hayashi, Y.: A novel GBM saliency detection model using multi-channel MRI. PLoS ONE **11**(1), e0146388 (2016). https://doi.org/10.1371/journal.pone.0146388

38. Pei, S.C., Cheng, C.M.: A novel block truncation coding of color images by using quaternion-moment-preserving principle. Connecting the World. In: IEEE International Symposium on Circuits and Systems, ISCAS 1996, pp. 684–687 (1996)
39. Sangwine, S.: Fourier transforms of colour images using quaternion or hypercomplex numbers. Electron. Lett. **32**(21), 1979–1980 (1996)
40. Sangwine, S., Ell, T.: Colour image filters based on hypercomplex convolution. IEE Proc. Vis. Image Signal Process. **147**(2), 89–93 (2000)
41. Ell, T., Sangwine, S.: Hypercomplex fourier transforms of color images. IEEE Trans. Image Process. **16**(1), 22–35 (2007). (PubMed: 17283762)
42. Sangwine, S.: Colour image edge detector based on quaternion convolution. Electron. Lett. **34**(10), 969–971 (1998)
43. Moxey, C., Sangwine, S., Ell, T.: Hypercomplex correlation techniques for vector images. IEEE Trans. Signal Process. **51**(7), 1941 (1953)
44. Lazendić, S., De Bie, H., Pižurica, A.: Octonion sparse representation for color and multispectral image processing. In: 2018 26th European Signal Processing Conference (EUSIPCO), pp. 608–612 (2018)
45. Gillies, R.J., Kinahan, P.E., Hricak, H.: Radiomics: images are more than pictures, they are data. Radiology **278**(2), 563–577 (2016). https://doi.org/10.1148/radiol.2015151169
46. Cameron, A., Khalvati, F., Haider, M.A., Wong, A.: MAPS: a quantitative radiomics approach for prostate cancer detection. IEEE Trans. Biomed. Eng. **63**(6), 1145–1156 (2016)
47. Haralick, R.M., Shanmugam, K., Dinstein, I.H.: Textural features for image classification. IEEE Trans. Syst. Man Cybern. **3**(6), 610–621 (1973)
48. Galloway, M.M.: Texture analysis using gray level run lengths. Comput. Graph. Image Process. **4**(2), 172–179 (1975)
49. Amadasun, M., King, R.: Textural features corresponding to textural properties. IEEE Trans. Syst. Man Cybern. **19**(5), 1264–1274 (1989)
50. Thibault, G., Fertil, B., Navarro, C., Pereira, S., Cau, P., Levy, N., et al.: Shape and texture indexes application to cell nuclei classification. Int. J. Pattern Recogn. Artif. Intell. **27**(01), 1545 (2013)
51. Lambin, P., Rios-Velazquez, E., Leijenaar, R., Carvalho, S., van Stiphout, R.G., Granton, P., Zegers, C.M., Gillies, R., Boellard, R., Dekker, A., et al.: Radiomics: extracting more information from medical images using advanced feature analysis. Eur. J. Cancer **48**(4), 441–446 (2012)
52. Delsuc, M.A.: Spectral representation of 2D NMR spectra by hypercomplex numbers. J. Magn. Reson. **77**(1), 119–124 (1988)
53. Ell, T.: Quaternion-fourier transforms for analysis of two-dimensional linear time-invariant partial differential systems. In: Proceedings of the 32nd IEEE Conference on Decision and Control, December 1993, pp. 1830–1841 (1993)
54. Bülow, T., Sommer, G.: Das Konzept einer zweidimensionalen Phase unter Verwendung einer algebraisch erweiterten Signalrepräsentation. In: Paulus, E., Wahl, F.M. (eds.) Mustererkennung 1997, pp. 351–358. Springer, Heidelberg (1997). https://doi.org/10.1007/978-3-642-60893-3_37
55. Subakan, O.N., Vemuri, B.C.: Color image segmentation in a quaternion framework. Energy Minimization Methods Comput. Vis. Pattern Recogn. **1**(5681), 401–414 (2009). https://doi.org/10.1007/978-3-642-03641-5_30
56. Fletcher, P.: Discrete wavelets with quaternion and clifford coefficients. Adv. Appl. Clifford Algebras **28**, 59 (2018). https://doi.org/10.1007/s00006-018-0876-5
57. Yamashita, R., Nishio, M., Do, R.K.G., et al.: Convolutional neural networks: an overview and application in radiology. Insights Imaging **9**, 611–629 (2018). https://doi.org/10.1007/s13244-018-0639-9
58. Bayro-Corrochano, E., Vallejo, R., Arana-Daniel, N.: Geometric preprocessing, geometric feedforward neural networks and Clifford support vector machines for visual learning. Neurocomputing **67**, 54–105 (2005). https://doi.org/10.1016/j.neucom.2004.11.041

Optimal Combination of Orientation Measurements Under Angle, Axis and Chord Metrics

Leo Dorst

Abstract Orientation measurements of attitudes estimate relative rotations of objects. The non-commutative algebra of rotations makes transference of techniques inspired by the usual vector-based approaches for translations non-trivial.

We treat three different metrics that may be used to compare orientations, compute the corresponding optimal averages, and relate them in a unified framework. The metrics are based on measuring differences in angular arc, axis tilt (or bivector) and rotational chord. We also compute the optimal combination of orientation estimates according to their local variances, as may be employed in a Kalman filter update step.

Our use of the geometric algebra characterization of rotations (through the bivector angle of rotors) allows us to perform computations and comparisons in a coordinate-free manner, and thus to compare and evaluate the alternative parametrizations. We briefly discuss how this subsumes and extends the traditional quaternion representation of rotations.

1 Processing Orientations and Rotations

In many applications of computer vision to 3D scenes, we encounter the problem of pose estimation. We may need to know where an object is in space relative to the camera, either to measure that object or, as in robotics, to calibrate our own stance in space. Typically, the scene or camera is moving, and we obtain a sequence of orientation measurements (or 'attitude measurements' for applications in space). We would like to combine the various measurements in a consistent way to obtain the best current pose estimate. For the translational (i.e. positional) part of the stance, this is not too hard; but orientations are more involved. Typical problems that need to be addressed are:

L. Dorst (✉)
Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands
e-mail: l.dorst@uva.nl

S. Xambó-Descamps (ed.), *Systems, Patterns and Data Engineering with Geometric Calculi*, SEMA SIMAI Springer Series 13, https://doi.org/10.1007/978-3-030-74486-1_4

- Interpolation or extrapolation of orientation sequences
- Filtering of orientation sequences
- Minimum variance estimation of orientations of the same pose.

It is common to describe *orientations* (or, equivalently, *attitudes*) using *rotation operations*, since the orientation as the state of an object is naturally characterized by referring to a standard object and a rotation operation: namely the (relative) rotation one has to give the reference copy of the object to reach the actual orientation. Though we will tend to use the three terms orientation, attitude and rotation interchangeably, 'rotations' will be the central term in our treatment.

1.1 Manifold Issues

We cannot immediately use familiar Euclidean techniques to solve these orientation problems, since the parameter space of rotations is not just a vector space to which linear techniques apply. Rotations in more than 2 dimensions form a (Lie-)group with a non-commutative multiplication (the composition of rotations), and 'live' on a somewhat unusual manifold. Yet one can still use the intuition of the processing of translations to develop very similar techniques for orientations, but now carefully taking into account the proper algebra. This is done by a combination of the technique of local linearization through employing the tangent space of the manifold, combined with a choice of algebraic representation for the non-commutative nature of the rotation operation.

There are many papers processing rotations or orientations; reference [1] contains a very extensive list, focussed on Kalman filtering in particular. Of course all workers attempting to provide processing of filtering of rotations recognize that accurate treatment requires taking the manifold structure into account. Yet even if one performs a linearized analysis in the local tangent space (and many do), this still requires the choice of a way to represent both the local perturbation and the rotational elements that are to be perturbed, plus the assignment of a metric, to measure the relative significance of differences.

1.2 Representation Matters

In older papers, such representational choices are not always made explicitly; one tends to use a standard representation. This may inadvertently affect the flexibility or applicability of the results. The natural noise model for one's mathematical representation may actually not be reasonable in practice. Lately, some structural frameworks for Lie-groups (including rotations) have been formulated [1, 7], which allow one to postpone the representational choice. Among the representations one encounters:

- *Rotation matrices* are notoriously awkward: all parametrizations by Euler angles contain discontinuities; they reside on a non-flat submanifold (orthogonal matrices) of the more tractable manifold of all matrices, and their derivatives as skew-symmetric matrices are sparse in the parametrization of change.
- *Unit quaternions* are in common use, in the usual representation of '4D complex numbers'. They are better behaved in terms of singularities, but there is still a mismatch between the number of dimensions of the 4D embedding space, and the 3D tangent space. Also, quaternion perturbation is non-linear, one of the reasons Kraft [11] gives for his need to resort to an *Extended* Kalman filter to process them. We will view this need as caused by the representation of orientations, rather than by their essence. Even then, there are many options for metrics [8].
- *Rotors or spinors* from geometric algebra are slowly gaining acceptance, especially in higher dimensions. They permit easy integration with the objects that are rotated, through the geometric sandwiching product. As we will show, even in 3D there is an advantage to rotors over quaternions, especially when they are represented as exponentials of bivectors. These rotors and their bivectors will be our preferred description.

Hertzberg et al. [7] give a general 'encapsulation' framework for Kalman filters on Lie groups, and work out the specific cases of directions and orientations in detail, demonstrating how their framework uncovers the common structure in various representations. But even they use the classical vector structure of the tangent space as their fundamental tool for modeling. Kanatani [9], too, uses the local Lie algebra structure, now in the form of sparse skew-symmetric matrices, and their vector characterization by means of cross products.

1.3 The Geometric Algebra Perspective

If one admits parametrization of the operations not by mere vectors from (metric) vector space, but *by the geometric algebra of that vector space*, one's tools for operational parametrization extend considerably. In that geometric algebra context, rotations are orthogonal transformations represented efficiently by *rotors* (a form of spinor). This works in n-dimensional space. In 3D, the rotors strongly resemble unit quaternions, but now embedded in the real algebra of elements of the 3D space itself. And in the full geometric algebra, those rotors can be parametrized by the linear space of bivectors (namely, as their exponentials). This gives their tangent space the structure of a Lie group, simply by applying the geometric product to the bivector characterization. The embedding permits the exponential map to operate, by the same geometric product, on the manifold itself, and also to make the rotational elements act (in a spinorial fashion) on all the linear subspaces of the vector space they act on (what roboticists would call the 'task space'.)

When applied to 3D rotations, the geometric algebra modelling clears up the mysterious nature of quaternions (they are geometric ratios of real vectors, a construction that easily extends to n dimensions), and permits a minimal parametrization by the linear space of bivectors—the available exponential map takes care of the 'unit quaternion' nature of the desired rotation representation. These instruments greatly clarify the computational relationships of the tangent space parametrization of perturbations and the rotational elements they affect. And an augmented set of tools, including geometric calculus, is then available to solve problems analytically.

The geometric algebra modeling does not resolve all problems in rotation processing. As for unit quaternions, the rotor representation appears to provide a 'double cover', so one should take care that all elements are part of the same sheet (though for composite objects the double cover does actually have a physically meaningful interpretation). And the fact that things can be compactly and consistently represented does not imply that problems also have a closed form solution: attempting to find an average rotation that minimizes the sum of relative bivectors is a counterexample treated in this paper. The non-commutative algebra of the tangent space, which is fundamental to Lie groups, does not disappear in geometric algebra (and it should not!); though we hope it may become more tractable when additional techniques are developed for the framework.

In this paper we present how the geometric algebra modelling of the tangent spaces of the particular Lie group of 3D rotations extends our analytical tools. The linear bivector nature of the tangent space allows combinations based on the locally flat nature of the tangent space to be expressed compactly—as we will see in the update step for a Kalman filter for rotations. Constructing actual operators from the perturbations is more clearly a separate step than in the matrix or quaternion frameworks, and this helps. Even within the geometric algebra framework, there are different ways of characterizing the noisy perturbations, and these can be made to correspond to actual measurement situations, rather than being chosen for convenience in one's representation of rotations. We treat an *arc metric*, a *bivector metric*, and a *chord metric* on the perturbations. Their unified consideration within 3D geometric algebra allows us to give explicit algebraic relationships between these characterizations, exact or to first or second order, so that the rotation filters and their results can be explicitly related. We show how the chord metric on rotors turns out to be very tractable, and to be virtually equivalent in its results to more classical metrics.

1.4 Paper Overview

We build up our understanding of this geometric algebra representation in a briefly tutorial manner, first introducing geometric algebra and its rotors in Sect. 2. We investigate, in Sect. 3, *three reasonable metrics* that one might impose on rotations, based on net differences in arc, bivector and (rotor) chord. We then show how to *average* estimates of poses for each of those metrics in Sect. 4. This leads naturally to partly known results on *interpolation* in Sect. 5. In Sect. 6 we present new results,

when we investigate how to characterize noise in orientation, and how to combine various estimates in an optimal manner by *minimizing the total error covariance*, as required in the Kalman filtering update step. We provide details on the conversion between the various parametrizations of orientations, to first order, including their covariance properties. In the Conclusions of Sect. 7 we discuss how the rotor chord representation effectively combines accuracy with tractability.

2 Rotation Representation by Rotors

2.1 Geometric Algebra

We give a brief summary of geometric algebra, focusing on the concepts and operations we need. More complete tutorial introductions may be found elsewhere, such as in [3].

Any vector space with a dot (or inner) product based on a symmetric bilinear form has a geometric algebra, in particular the n-dimensional Euclidean vector spaces. The basis for the algebra is the *geometric product*, which is bilinear, associative, commutes with scalars, and for a vector $\mathbf{x}$ with itself equals a scalar equal to the dot product $\mathbf{x} \cdot \mathbf{x}$. Because of its fundamental nature, we will denote the geometric product of two quantities $\mathbf{A}$ and $\mathbf{B}$ simply as $\mathbf{A}\,\mathbf{B}$. (This gives no confusion with the scalar product in the vector space, since that is just a special case of the geometric product, as is the product of two scalars.)

The geometric product is not commutative, and that fact is related to the properties of parallelness and perpendicularity. Since those properties are important, we define two convenient derived products, the *inner and outer product*. For vectors $\mathbf{a}$ and $\mathbf{b}$, these are defined as

$$\begin{aligned} \text{inner product}: \quad & \mathbf{a} \cdot \mathbf{b} \equiv \tfrac{1}{2}(\mathbf{a}\,\mathbf{b} + \mathbf{b}\,\mathbf{a}) \\ \text{outer product}: \quad & \mathbf{a} \wedge \mathbf{b} \equiv \tfrac{1}{2}(\mathbf{a}\,\mathbf{b} - \mathbf{b}\,\mathbf{a}). \end{aligned}$$

The inner product is symmetric and scalar-valued. For vectors it coincides with the familiar dot product, but it is naturally extended to the multivectors that constitute the geometric algebra. It is associated with the geometrical intuition of orthogonal projection and perpendicularity. The outer product is anti-symmetric and for vectors is 'bivector-valued'. Bivectors are interpretable as directed area elements (similar to the way that vectors are directed line elements), at least when they are factorizable as an outer product of vectors (and in 3D, all of them are). The outer product thus makes the geometrical concept of a 'span' of vectors a computable element of the algebra, which is a handy 'upgrade' of linear algebra. It relates to parallelness of vectors and of the subspaces it spans. In 3-dimensional Euclidean space, the outer product is related to the cross product from classical vector algebra, as we show below.

The outer product can be extended by associativity; and generates elements representing subspaces of various dimensionalities. These subspaces are called *blades* and their dimensions *grades*. Due to the anti-symmetry, the k-vectors form an $\binom{n}{k}$-dimensional linear space (in which the k-blades form a manifold). The highest non-zero blade in an n-dimensional space is an n-blade; it can be interpreted as an n-dimensional oriented hypervolume.

Geometric algebra focuses on blades and their products. Semantically, it is therefore the quantitative algebra of the subspaces of a metric vector space. The geometric product of an element grade k and an element of grade ℓ in general contains elements of grades $k+\ell, k+\ell-2, \cdots, |k-\ell|$. The part of grade i of an element A is selected by the *grade operator* denoted $\langle A \rangle_i$. So for example for two vectors $\mathbf{a}$ and $\mathbf{b}$ we have $\mathbf{a} \cdot \mathbf{b} = \langle \mathbf{ab} \rangle_0$ and $\mathbf{a} \wedge \mathbf{b} = \langle \mathbf{ab} \rangle_2$.

The *norm squared* $\|A\|^2$ of a quantity A is defined as

$$\|A\|^2 = \langle \widetilde{A} A \rangle_0, \tag{1}$$

where $\widetilde{A}$ (alternatively denoted $A^\sim$) is the *reverse*, obtained by writing all multiplicative factors in A in reverse order. For instance for vectors $\mathbf{a}$ and $\mathbf{b}$ we have $\widetilde{\mathbf{a}} = \mathbf{a}$ and $(\mathbf{a} \wedge \mathbf{b})^\sim = \mathbf{b} \wedge \mathbf{a} = -\mathbf{a} \wedge \mathbf{b}$. For blades in a Euclidean vector space, $\|A\|^2$ is a positive scalar quantity, so its square root is well-defined.

A very useful property of the geometric product is its *invertibility*: we can divide by vectors, bivectors (or rather, 2-blades) etcetera. Dividing is multiplication by the inverse of an element (if it exists); and for blades this gives:

$$A^{-1} \equiv \frac{\widetilde{A}}{\widetilde{A} A}. \tag{2}$$

The inverse of a vector is therefore $\mathbf{a}^{-1} = \frac{\mathbf{a}}{\mathbf{a}\,\mathbf{a}} = \frac{\mathbf{a}}{\mathbf{a} \cdot \mathbf{a}} = \frac{\mathbf{a}}{\|\mathbf{a}\|^2}$, and indeed it is easily verified that $\mathbf{a}^{-1}\mathbf{a} = 1$. A unit vector is its own inverse, a unit bivector $\mathbf{B}$ has its reverse $\widetilde{\mathbf{B}} = -\mathbf{B}$ as inverse.

The volume element of the n-D space I_n is an n-blade called its *pseudoscalar*; it has a sign reflecting the chirality. In $\mathbb{R}^n$ one commonly normalizes it to unity. Then division by the unit pseudoscalar I_n converts an element of grade k into its *dual* of grade $n-k$, through division by the pseudoscalar:

$$\text{duality operation:} \quad A^* = A\, I_n^{-1}. \tag{3}$$

In 3D, this duality can be used to convert a 2-blade (representing a rotation plane) to an axial vector (representing a rotation axis). This conversion is important in the context of this paper since the classical description tends to use the latter, while the geometric algebra approach uses the former—among other reasons, because that also works in n dimensions, not just 3. Retrieving A from A^* involves 'undualization': $A = (A^*)^{-*}$, which in Euclidean $\mathbb{R}^3$ differs from dualization by a minus sign.

Note that the norm of the 2-blade and its dual vector are the same, since $\widetilde{I_3} = I_3^{-1}$:

$$\|A\|^2 = \langle A\widetilde{A}\rangle_0 = \langle AI_3^{-1}I_3\widetilde{A}\rangle_0 = \langle AI_3^{-1}(AI_3^{-1})^{\sim}\rangle_0 = \|A^*\|^2. \tag{4}$$

A specific example of duality in 3D is the relationship between the vector cross product (which yields a vector) and the outer product (which yields a bivector):

$$\mathbf{a} \times \mathbf{b} = (\mathbf{a} \wedge \mathbf{b})^*. \tag{5}$$

The inverse involved in the duality Eq. (3) ensures that $\mathbf{e}_1 \wedge \mathbf{e}_2$ corresponds with the vector $\mathbf{e}_3 = \mathbf{e}_1 \times \mathbf{e}_3$, for a right-handed oriented $I_3 = \mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \mathbf{e}_3$, so a right-hand rule applies.

2.2 The Geometric Algebra of 3-Dimensional Euclidean Space

Let us list the elements of the geometric algebra of 3-dimensional Euclidean space, our only concern in this paper. Linearity of the construction means that we are free to choose our basis. For convenience, we use an orthonormal basis $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ for the vectors. Coefficients of vectors or scalars on this basis are all real. The 'basis' for the scalars is thus $\{1\}$. For an orthonormal basis, $\mathbf{e}_i\,\mathbf{e}_i = 1$, and for $i \neq j$ we have $\mathbf{e}_i\,\mathbf{e}_j = -\mathbf{e}_j\,\mathbf{e}_i (= \mathbf{e}_i \wedge \mathbf{e}_j)$.

With this we can develop the geometric product of two general vectors $\mathbf{a}$ and $\mathbf{b}$. Choose a suitable basis to have $\mathbf{a} = \|\mathbf{a}\|\,\mathbf{e}_1$, and $\mathbf{b} = \|\mathbf{a}\|\,(\cos\theta\,\mathbf{e}_1 + \sin\theta\,\mathbf{e}_2)$. This implies that $\mathbf{I} \equiv \mathbf{e}_1 \wedge \mathbf{e}_2$ can be interpreted as the unit area element of the $(\mathbf{a}, \mathbf{b})$-plane, and θ is the angle *from* $\mathbf{a}$ *to* $\mathbf{b}$ in that plane $\mathbf{I}$ (where the 'positive direction of angles' is denoted by the sign of $\mathbf{I}$). That gives:

$$\mathbf{a}\,\mathbf{b} = \|\mathbf{a}\|\,\|\mathbf{b}\|\,(\cos\theta + \mathbf{I}\sin\theta). \tag{6}$$

Since this result is independent of the coordinate system (because the definition of the geometric product is), the geometric product always contains this complete geometric information about the relationship of the vectors: relative angle θ and common plane $\mathbf{I}$, and relative size (the latter is slightly better conveyed by the geometric division $\mathbf{a}/\mathbf{b} = \mathbf{a}\,\mathbf{b}^{-1} = (\|\mathbf{a}\|/\|\mathbf{b}\|)(\cos\theta + \mathbf{I}\sin\theta)$ which gives $\|\mathbf{a}\|/\|\mathbf{b}\|$ as the magnitude). The fact that $\mathbf{a}$ can be retrieved from knowing $\mathbf{b}$ and the quantity $(\mathbf{a}/\mathbf{b})$ (namely as $\mathbf{a} = (\mathbf{a}/\mathbf{b})\,\mathbf{b}$) implies that *the geometric ratio contains the full information on the relative geometry of* $\mathbf{a}$ *and* $\mathbf{b}$.

Actually computing the geometric product for two general vectors on an arbitrary orthonormal basis reveals how familiar the coefficients are:

$$
\begin{aligned}
&(a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + a_3\mathbf{e}_3)\,(b_1\mathbf{e}_1 + b_2\mathbf{e}_2 + b_3\mathbf{e}_3) \\
&= (a_1b_1 + a_2b_2 + a_3b_3) \\
&\quad + (a_2b_3 - a_3b_2)\mathbf{e}_2\mathbf{e}_3 + (a_3b_1 - a_1b_3)\mathbf{e}_3\mathbf{e}_1 + (a_1b_2 - a_2b_1)\mathbf{e}_1\mathbf{e}_2
\end{aligned} \tag{7}
$$

This confirms that the scalar part is equal to the dot product, and shows how the part of grade 2 has coefficients which are similar to that of a cross product but now on a *bivector basis* $\{\mathbf{e}_2 \wedge \mathbf{e}_3, \mathbf{e}_3 \wedge \mathbf{e}_1, \mathbf{e}_1 \wedge \mathbf{e}_2\} = \{\mathbf{e}_2\mathbf{e}_3, \mathbf{e}_3\mathbf{e}_1, \mathbf{e}_1\mathbf{e}_2\}$ rather than the vector basis $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$. This is the coordinate form of the duality correspondence Eq. (5).

Taking the geometric product of three vectors $\mathbf{a}\,\mathbf{b}\,\mathbf{c}$, one finds that the product of 3 vectors contains a 1-vector part and a 3-vector part. That 3-vector part is a multiple of a *basis trivector* $\mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \mathbf{e}_3$, by the determinant of the matrix $(\mathbf{a}\,\mathbf{b}\,\mathbf{c})$. We will often denote this trivector element by I_3, since it is independent of the actual orthonormal basis used: it is just the oriented unit volume element in 3D.

As an 8-dimensional basis to denote elements of the geometric algebra of 3-dimensional real Euclidean space, we thus obtain:

$$
\left\{ \underbrace{1}_{\text{scalars}},\ \underbrace{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3}_{\text{vector space}},\ \underbrace{\mathbf{e}_2 \wedge \mathbf{e}_3,\ \mathbf{e}_3 \wedge \mathbf{e}_1,\ \mathbf{e}_1 \wedge \mathbf{e}_2}_{\text{bivector space}},\ \underbrace{\mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \mathbf{e}_3}_{\text{trivector space}} \right\}. \tag{8}
$$

The geometric product multiplication table of these basis elements is as follows. (We indicated the blades with geometric products to save space; for our orthonormal basis this is allowed since $\mathbf{e}_i \wedge \mathbf{e}_j = \mathbf{e}_i\,\mathbf{e}_j$ if $i \neq j$.)

1	$\mathbf{e}_1$	$\mathbf{e}_2$	$\mathbf{e}_3$	$\mathbf{e}_2\mathbf{e}_3$	$\mathbf{e}_3\mathbf{e}_1$	$\mathbf{e}_1\mathbf{e}_2$	$\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$
$\mathbf{e}_1$	1	$\mathbf{e}_1\mathbf{e}_2$	$-\mathbf{e}_3\mathbf{e}_1$	$\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$	$-\mathbf{e}_3$	$\mathbf{e}_2$	$\mathbf{e}_2\mathbf{e}_3$
$\mathbf{e}_2$	$-\mathbf{e}_1\mathbf{e}_2$	1	$\mathbf{e}_2\mathbf{e}_3$	$\mathbf{e}_3$	$\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$	$-\mathbf{e}_1$	$\mathbf{e}_3\mathbf{e}_1$
$\mathbf{e}_3$	$\mathbf{e}_3\mathbf{e}_1$	$-\mathbf{e}_2\mathbf{e}_3$	1	$-\mathbf{e}_2$	$\mathbf{e}_1$	$\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$	$\mathbf{e}_1\mathbf{e}_2$
$\mathbf{e}_2\mathbf{e}_3$	$\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$	$-\mathbf{e}_3$	$\mathbf{e}_2$	-1	$-\mathbf{e}_1\mathbf{e}_2$	$\mathbf{e}_3\mathbf{e}_1$	$-\mathbf{e}_1$
$\mathbf{e}_3\mathbf{e}_1$	$\mathbf{e}_3$	$\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$	$-\mathbf{e}_1$	$\mathbf{e}_1\mathbf{e}_2$	-1	$-\mathbf{e}_2\mathbf{e}_3$	$-\mathbf{e}_2$
$\mathbf{e}_1\mathbf{e}_2$	$-\mathbf{e}_2$	$\mathbf{e}_1$	$\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$	$-\mathbf{e}_3\mathbf{e}_1$	$\mathbf{e}_2\mathbf{e}_3$	-1	$-\mathbf{e}_3$
$\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$	$\mathbf{e}_2\mathbf{e}_3$	$\mathbf{e}_3\mathbf{e}_1$	$\mathbf{e}_1\mathbf{e}_2$	$-\mathbf{e}_1$	$-\mathbf{e}_2$	$-\mathbf{e}_3$	-1

Note that the basis 2-blades each square to -1. Yet these are not complex numbers: for instance, $\mathbf{e}_1 \wedge \mathbf{e}_2$ does not commute with $\mathbf{e}_1$, whereas a complex scalar would have done so. In 3D, the pseudoscalar $I_3 = \mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$ does commute with all elements, and it does square to -1. Still, in $\mathbb{R}^n$ the pseudoscalar only commutes for odd n, and only squares to -1 when $n = 2 \mod 4$ or $n = 3 \mod 4$), so that the pseudoscalar should not be considered to be the complex unit either. Actually, the point is moot: with the availability of truly geometrical plane elements that square to -1, we will not need to employ complex numbers to perform rotations.

The bilinearity and associativity of the geometric product imply that it can simply be implemented on the basis of Eq. (8) as a $2^3 \times 2^3$ matrix multiplication in a standard linear algebra package, and you can gain some experience with it in that way. But once you switch over to geometric algebra, there are more efficient direct implementations,

especially for the geometrically meaningful operations which tend to be rather sparse in that coarse matrix viewpoint.

2.3 Reflections and Rotations

We can think of the blades of the geometric algebra of $\mathbb{R}^3$ as real representations of spanned subspaces of dimension 0, 1, 2 and 3 within a 3-dimensional Euclidean space. These indicate weighted, oriented directional elements of various dimensions in $\mathbb{R}^3$. We can change such directional elements in two closely related ways: by reflection and by rotation.

The reflection of a vector $\mathbf{x}$ into a plane through the origin characterized by the *normal* vector $\mathbf{a}$ is given by the elementary classical formula:

$$\mathbf{x} \;\mapsto\; \mathbf{x} - 2\,(\mathbf{a}\cdot\mathbf{x})\,\mathbf{a}/\|\mathbf{a}\|^2. \tag{9}$$

Using the geometric product to express the dot product as $\mathbf{a}\cdot\mathbf{x} \equiv \frac{1}{2}(\mathbf{a}\,\mathbf{x} + \mathbf{x}\,\mathbf{a})$, and our capability of dividing by vectors, we can rewrite this to the form

$$\mathbf{x} \;\mapsto\; -\,\mathbf{a}\,\mathbf{x}\,\mathbf{a}^{-1}. \tag{10}$$

Two reflections make a rotation: first reflecting in $\mathbf{a}$, then in $\mathbf{b}$, gives a rotation around an axis perpendicular to the $\mathbf{a}\wedge\mathbf{b}$-plane, by an angle that is *twice* the angle from $\mathbf{a}$ to $\mathbf{b}$ (and this order also gives the sense of positive rotation). Therefore, when we have two unit vectors $\mathbf{a}$ and $\mathbf{b}$, the operation

$$\mathbf{x} \;\mapsto\; \mathbf{b}(\mathbf{a}\,\mathbf{x}\,\mathbf{a}^{-1})\mathbf{b}^{-1} = (\mathbf{b}\,\mathbf{a})\,\mathbf{x}\,(\mathbf{b}\,\mathbf{a})^{-1}. \tag{11}$$

is that rotation in the $\mathbf{a}\wedge\mathbf{b}$ plane. We observe that this rotation is generated by an element $R = \mathbf{b}\,\mathbf{a}$, as applied to a vector by the *sandwiching product*:

$$\mathbf{x} \;\mapsto\; R\,\mathbf{x}\,R^{-1}. \tag{12}$$

It is common to avoid the inversion by computing the element R as the product of two *unit vectors*. This normalization allows one to replace the inverse by a reverse, and thus rewrite the sandwiching product to

$$\mathbf{x} \;\mapsto\; R\,\mathbf{x}\,\widetilde{R}. \tag{13}$$

This normalized element R is called a *rotor*, and it satisfies

$$\|R\|^2 = R\widetilde{R} = 1, \tag{14}$$

as is easily verified: $R\widetilde{R} = \mathbf{b}\,\mathbf{a}\,\mathbf{a}\,\mathbf{b} = 1$. We will use these rotors as the representation of rotations. The fact that elements of geometric algebra can be (recursively) expressed as weighted sums of geometric products of vectors implies that the application of rotors extends beyond vectors: *any* element of the geometric algebra rotates as $X \mapsto R\,X\,\widetilde{R}$, i.e., by sandwiching.

Writing out the geometric product, we find for the unit vectors $\mathbf{a}$ and $\mathbf{b}$:

$$\mathbf{b}\,\mathbf{a} = \mathbf{b}\cdot\mathbf{a} + \mathbf{b}\wedge\mathbf{a} = \cos(\theta/2) - \mathbf{I}\sin(\theta/2) = \mathrm{e}^{-\mathbf{I}\theta/2}, \tag{15}$$

where $\theta/2$ is the angle from $\mathbf{a}$ to $\mathbf{b}$, and $\mathbf{I}$ is the unit 2-blade for the $(\mathbf{a}\wedge\mathbf{b})$-plane. The exponential notation is based on employing the geometric product in the usual power series definition:

$$\mathrm{e}^{X} = 1 + X + \tfrac{1}{2}X^2 + \tfrac{1}{3!}X^3 + \cdots, \tag{16}$$

using the algebraic property that $\mathbf{I}^2 = -1$ to produce the trigonometric functions. This operator $(\mathbf{b}\,\mathbf{a})$ rotates over an angle θ in the $\mathbf{I}$-plane (or, if you prefer, around the $\mathbf{I}^*$-axis). The fact that the rotor can be written only in terms of $\mathbf{I}\theta$ shows the convenience of the parametrization in terms of this *bivector angle* $\mathbf{I}\theta$. That weighted, oriented 2-blade contains all information about the rotation: both the magnitude of the angle and the oriented plane in which it should be measured. Keeping these in one quantity $\mathbf{I}\theta$ makes the characterization geometrical, independent of the convention of clockwise or anti-clockwise assignation. From this bivector angle, one can immediately construct the rotor performing the corresponding rotation through exponentiation (well, after multiplying by $-\frac{1}{2}$ to conform to conventions of angular sign and magnitude as Eq. (15) shows).

Geometric algebra contains a logarithm to retrieve the bivector angle from a rotor. In 3D, that is a straightforward inversion of the definition of the exponential:

$$\log(R) = \frac{\langle R\rangle_2}{\|\langle R\rangle_2\|}\,\mathrm{atan}\Big(\frac{\|\langle R\rangle_2\|}{\langle R\rangle_0}\Big), \tag{17}$$

where $\langle R\rangle_k$ denotes the grade-k-part of R. We will take the principal value of this logarithm, with a magnitude less than $\pi/2$, so that the rotation angle is in the range $(-\pi, \pi)$, corresponding to rotors for which $\langle R\rangle_0 \geq 0$. This formula for the logarithm of a rotor is ill-defined when it has no bivector component. When this occurs at $R = 1$, we define the logarithm to be 1; when this occurs at $R = -1$, we can return an arbitrary bivector $\mathbf{I}\pi$. But in what follows, we will avoid this ambiguous situation by ensuring that $\langle R\rangle_0$ is non-negative. This is no loss of applicability to orientation measurement, since using the transformation formula $\mathbf{x} \mapsto R\,\mathbf{x}\,\widetilde{R}$, we see that a rotor R and 'minus that rotor' $-R$ give the same resulting rotation to a vector $\mathbf{x}$.

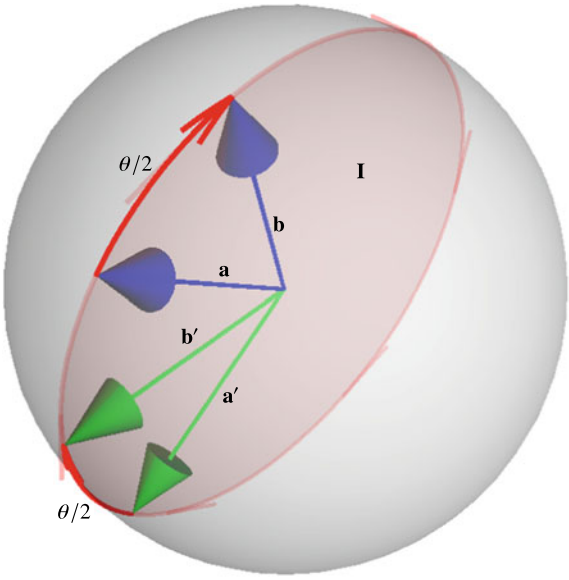


Fig. 1 A rotor $R_{\mathbf{I}\theta}$ can be visualized as an oriented arc on the intersection of the unit sphere with the **I**-plane, of arc length $\theta/2$. The arc should be allowed to slide freely along the circle, since both $\mathbf{b}/\mathbf{a}$ and its rotated version $\mathbf{b}'/\mathbf{a}'$ must represent the same rotor

2.4 The Geometry of Rotors: Adding Spherical Arcs

It is sometimes useful to have a visualization of the rotors, and their combination. We will provide this for 3D rotations. This visualization juggles the two alternative but equivalent representations: as a geometric product of unit vectors **a** and **b**, and as the exponential of a 2-blade $\mathbf{I}\theta$:

$$R_{\mathbf{I}\theta} = \mathbf{b}\,\mathbf{a} = \mathrm{e}^{-\mathbf{I}\theta/2}, \tag{18}$$

which are related by $\mathbf{I} = (\mathbf{a} \wedge \mathbf{b})/\|\mathbf{a} \wedge \mathbf{b}\|$ being the oriented plane of rotation, and θ the effective rotation angle, twice the angle between the reflection vectors. Note that this bivector angle $\mathbf{I}\theta$, and with it $R_{\mathbf{I}\theta}$, is invariant under a common rotation of **a** and **b** in their common plane. We can visualize $R_{\mathbf{I}\theta}$ in Fig. 1 as an arc on the unit sphere, of arc length $\theta/2$, in the **I**-plane (we will take $+\theta$ since that sign corresponds with the actual rotation performed by the rotor). But this arc should be allowed to be anywhere on the great circle in which the **I**-plane cuts the unit sphere, since the rotor is invariant under rotations in the **I**-plane.

These 'slidable arcs' help visualize the composition of two rotations R_1 then R_2 in 3D: represent each by an arc of half their angle on the great circle cut into the unit sphere by their rotation plane. Now rotate the arc of the second rotation R_2 in its plane $\mathbf{I}_2$, so that its start connects to the end of the rotor R_1 in its plane $\mathbf{I}_1$ (which may need to be rotated to an intersection point to make this possible). Then a great circle arc that connects the tail of R_1 to the head of R_2 is the arc of the resulting rotation $R_2 R_1$: its attitude gives the resulting rotation plane, and the total resulting angle is twice its arc length, see Fig. 2.

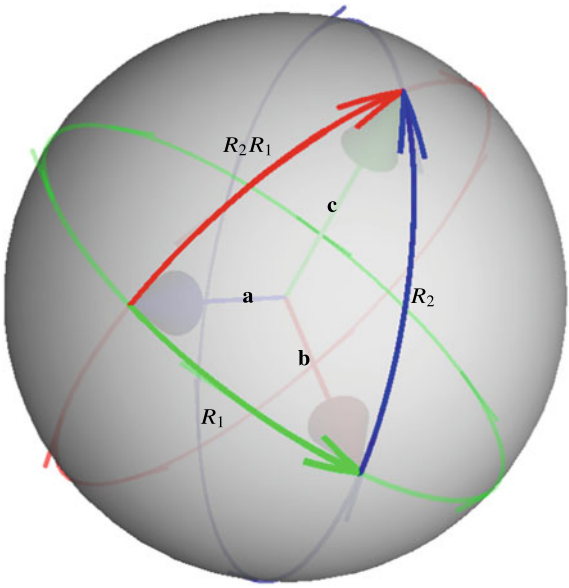


Fig. 2 Composition of rotations through concatenation of their rotor arcs: R_2R_1 is the composite rotor of doing first R_1, then R_2, and is the arc oriented completing the spherical triangle formed by the arcs of R_1 and R_2

Why does this work? Let us switch to viewing the rotors again as a product of the form $\mathbf{b}\,\mathbf{a}$, the two unit vectors $\mathbf{a}$ and $\mathbf{b}$ making a 'vee'. We can rotate this vee freely in its plane, and still have the same rotor. Now, composing two rotors (in possibly different planes) is identical to composing two double reflections, in two planes with these vectors as their unit normals. It is then natural to rotate the two vees of normal vectors so that the last vector of the first (rightmost) rotor and the first vector of the last (leftmost) rotor coincide (since we are free to rotate them). In 3D, this can always be done: it amounts to moving them both to the common line of intersection of the two rotor planes. Let us call this common unit vector direction $\mathbf{b}$; then the first rotor can be written as $\mathbf{ba}$, and the second as $\mathbf{cb}$. Combining the two in the correct order gives $(\mathbf{c}\,\mathbf{b})\,(\mathbf{b}\,\mathbf{a}) = \mathbf{c}\,(\mathbf{b}\,\mathbf{b})\,\mathbf{a} = \mathbf{ca}$ (since $\mathbf{b}$ is a unit normal vector). So the common direction of the aligned vees cancels, and what is left is the vee $\mathbf{ca}$, defining the arc of the net rotation using the first vector of the first realigned rotor, followed on the left by the second vector of the second realigned rotor. The arc defines the resulting rotation plane and angle.

Thus *rotor multiplication* implements this kind of *arc addition* on the unit sphere. This makes it easy to predict the consequence of performing two rotations in 3D—as easy as combining translations, really.

2.5 Rotors and Quaternions

Rotors are closely related to quaternions: quaternions are simply rotors separated from their natural context in geometric algebra (and because of that, to many

people unfortunately more mysterious than they need to be). Rotors are real operators in a real Euclidean vector space, applying to any element in its geometric algebra (scalars, vectors, bivectors, trivectors, other rotors, etc., which represent real geometric primitives such as points, lines, planes, volumes and rotation operators). When split into its grades, a rotor has two interpretable components: a scalar part (related to the cosine of the angle) and a bivector part (containing sine and rotation plane). In quaternion literature, the non-scalar part of a quaternion is often seen as a vector that denotes the rotation axis, but expressed on a strange basis of complex vector quantities i, j, k that square to -1 and do not commute. For us, these basis elements are not vectors but bivectors, representing the coordinate planes:

$$i = \mathbf{e}_1 I_3 = \mathbf{e}_2\mathbf{e}_3, \quad j = \mathbf{e}_2 I_3 = \mathbf{e}_3\mathbf{e}_1, \quad k = \mathbf{e}_3 I_3 = \mathbf{e}_1\mathbf{e}_2, \tag{19}$$

with I_3 the 3D pseudoscalar; so we prefer to characterize a rotation directly by its plane rather than its complexified axis. Note that our basic geometric product then automatically gives the quaternion relations $j\,i = k$ and cyclic, and $i\,j\,k = 1$; i.e., the usual quaternion product is a special case of the geometric product. Quaternions are considered non-intuitive, due to their confusing correspondence to complex quantities, and due to the artificial way of mapping vectors to be 'pure' quaternions in order to allow them to be rotated. We view quaternions as a historic, even antiquated, encoding of rotors out of their natural context [3].

As an aside that may convince the skeptics, this geometric view of quaternions is actually productive: it has led to a new but simple and fast formula to compute the quaternion for two vector correspondences [6].

> Given two pairs of corresponding vector $(\mathbf{x}, \mathbf{x}')$ and $(\mathbf{y}, \mathbf{y}')$, the (unnormalized) rotor V that produces the correspondences $\mathbf{x}' = V\,\mathbf{x}\,V^{-1}$ and $\mathbf{y}' = V\,\mathbf{y}\,V^{-1}$ is:
> $V = (\mathbf{y}' + \mathbf{y}) \cdot (\mathbf{x}' - \mathbf{x}) + (\mathbf{y}' - \mathbf{y}) \wedge (\mathbf{x}' - \mathbf{x})$.

Converting this to the vector notation of quaternions merely involves replacing the outer product by a cross product. It is geometric algebra's credit that this basic formula for generating a quaternion based on data had not been found in the quaternion literature. Viewing rotors (and hence quaternions) as multiple planar reflections is crucial to its simple proof in [6].

3 Three Metrics for Orientations and Rotations

If one had to design a metric for two *orientations* (or *attitudes*) of an object, to give a measure for their difference, some function of the angle between them would seem appropriate. When considering *rotations*, which differ in both angle and in their rotation planes, it seems less obvious how to mix those into a sensible measure. Since we treat rotations and orientations as interchangeable, one could argue that the spatial issue arises for orientations as well.

In this paper, we study three different methods of measuring orientation differences, and the consequences they have for how different estimates are combined. To compare them, we need a common setting for the curved group manifold of rotations and its flat (Euclidean) tangent spaces. The metric one employs in those tangent spaces is to be expressed in one's chosen parametrization of rotations.

Recently, two practical frameworks for processing data in this algebra of 3D rotations and other Lie algebras have been formulated, in Bourmaud et al. [1] and Hertzberg et al. [7]. Both are usable, but the *encapsulation framework* of [7] has a simpler notation, and is more explicit on the metric consequences of different parametrizations; that is why we adapt it in this paper.

The encapsulation framework elegantly encodes the mapping between a manifold $\mathcal{S}$ (in our case, $SO(3)$) and the Euclidean tangent space $\mathbb{R}^n$. We quote directly from [7]:

> We propose to implement the mapping by means of two encapsulation operators $\boxplus$ ("boxplus") and $\boxminus$ ("boxminus") where:
>
> $$\boxplus : \mathcal{S} \times \mathbb{R}^n \to \mathcal{S} \tag{20}$$
>
> $$\boxminus : \mathcal{S} \times \mathcal{S} \to \mathbb{R}^n \tag{21}$$
>
> Here, $\boxplus$ takes a manifold state and a small change expressed in the mapped local neighborhood in $\mathbb{R}^n$ and applies this change to the state to yield a new, modified state. Conversely, $\boxminus$ determines the mapped difference between two states.
>
> The encapsulation operators capture an important duality: The generic sensor fusion algorithm uses $\boxminus$ and $\boxplus$ in place of the corresponding vector operations $-$ and $+$ to compare and to modify states, respectively, based on flattened perturbation vectors, and otherwise treats the state space as a black box. Problem-specific code such as measurement models, on the other hand, can work inside the black box, and use the most natural representation for the state space at hand.

The choice of parametrization one employs for the rotations may be a compromise between mathematical tractability and actual noise characteristics of one's measurement setup. Depending on how we consider the noise in rotations, a metric may suggest itself. The encapsulation framework *connects that metric uniquely to the parametrization*: the Euclidean norm in the parametrized tangent space will be the metric for comparison of rotations (Lemma 2 in [7]). This metric is thus induced by the chosen $\boxminus$ operator, as

$$d : \mathcal{S} \times \mathcal{S} \to \mathbb{R} : d(R_1, R_2) = \|R_2 \boxminus R_1\|. \tag{22}$$

Tying the metric this directly to the parametrization connects the two choices in a formal way which permits us to compare and relate the various options. We appreciate the increased clarity which such a strict use of the encapsulation framework enforces explicitly.

So our focus will be on the tangent space parametrization. Our reference [7] uses classical linear algebra, but we will find that the same principles can be given more analytical techniques by employing the geometric algebra of the tangent spaces. To enable this, we characterize rotations (and hence relative orientations) by rotors

R, which may also be characterized by their bivector $\mathbf{B} = -\mathbf{I}\theta/2$.[1] This in itself is simply the language in which we describe the rotations, and does not yet involve a parametrization which uniquely fixes (by means of the encapsulation framework) the metric of the tangent space. We will initially show the vector form of each of our parametrizations, to correspond to the techniques in [7]. But we quickly switch over to geometric algebra bivectors (their duals), since those give us more tools for manipulating the induced metric structure of the tangent space—and suggest more clearly how one would extend these techniques to arbitrary dimensions. The connections of geometric algebra rotors to traditional quaternions will allow us to identify some of the metrics with alternatives discussed in [8] for quaternions.

3.1 Arc Metric

A noisy measurement of a rotor can be considered as a perturbation afterward by an extra rotor, typically close to the identity 1:

$$\text{multiplicative rotation noise: } \mathrm{e}^{\mathbf{a}}\,\mathrm{R} = \mathrm{e}^{\mathbf{a}}\,\mathrm{e}^{\mathbf{B}}, \tag{23}$$

where $\mathbf{a}$ is a small bivector (the small font reminds us). Due to the non-commutation of $\mathbf{a}$ and $\mathbf{B}$, we cannot write this simply as a single exponential. In terms of [7], we have effectively defined an operation $\boxplus : \mathcal{S} \times \mathbb{R}^3 \to \mathcal{S}$ combining a perturbation vector in the Euclidean space $\mathbb{R}^3$ to characterize a transformation to a new rotor in the rotor manifold $\mathcal{S}$. As we said, our reference [7] prefers characterization by a vector, so to conform we simply take one whose dual is the bivector $\mathbf{a}$, i.e., $\overrightarrow{a}^{\,*} = \mathbf{a}$. Then we can cast our perturbation as:

$$R \boxplus_{\mathrm{a}} \overrightarrow{a} \equiv \mathrm{e}^{\overrightarrow{a}^{\,*}}\,\mathrm{R}. \tag{24}$$

(The subscript 'a' in $\boxplus_{\mathrm{a}}$ index refers to 'arc' or 'arc angle' of Fig. 1, we will define other $\boxplus$ operators later.)

Accompanying this is a difference operation $\boxminus_{\mathrm{a}} : \mathcal{S} \times \mathcal{S} \to \mathbb{R}^3$ which compares two rotors:

$$R_2 \boxminus_{\mathrm{a}} R_1 \equiv \Big(\log(R_2\,\widetilde{R}_1)\Big)^{-*} = \Big(\log(R_2\,\widetilde{R}_1)\Big)\,I_3, \tag{25}$$

undualizing to retrieve the vector $\overrightarrow{a}$ characterizing the perturbation (though in the remainder we will be content with retrieving the bivector $\mathbf{a}$). One easily verifies that $(R \boxplus_{\mathrm{a}} \mathbf{a}) \boxminus_{\mathrm{a}} R = \mathbf{a}$, as required. In the encapsulation framework, a natural metric is associated with this difference operation, according to Eq. (22):

[1] We assume, without loss of generality, that those bivectors are taken within the main range of half-angle values, so $\mathbf{B}$ is a unit bivector multiplied by a half-angle in the range $(-\pi, \pi)$ (avoiding the ambiguous value $\pm\pi$ altogether).

$$d_{\mathrm{a}}(R_2, R_1) = \| R_2 \boxminus_{\mathrm{a}} R_1 \| = \| \log(R_2 \widetilde{R}_1) \|, \tag{26}$$

which is the absolute magnitude of the difference in angle (modulo 2π). This Euclidean norm has the same value whether defined on the vector $\overrightarrow{a}$ or its dual bivector $\mathbf{a}$, by Eq. (4), so we dropped the I_3 from Eq. (26). Either way, d_{a} evaluates to $\|\mathbf{a}\| = |\theta/2|$, half the absolute angle of the perturbation (since rotors employ half angles).

This is effectively the metric $\Phi_3 = 2\Phi_6$ of [8], even though the formula given there is rather different.

3.2 *Bivector Metric (or Axis Metric)*

Alternatively, if one has characterized the rotation by an axial vector $\overrightarrow{b}$, it is naturally tempting to characterize the noise by a small additive vector perturbing this axis, and this is very commonly done (some examples are [5, 9, 14]). In terms of geometric algebra, this additive axis perturbation is equivalent to changing the bivector by a small additive bivector, so that the rotor $\mathrm{e}^{\mathbf{B}}$ becomes the noisy rotor

$$\text{additive bivector noise: } \mathrm{e}^{\mathbf{B}+\mathbf{b}}. \tag{27}$$

Again, we could use this bivector $\mathbf{b}$ directly, but to show the correspondence with the encapsulation framework of [7], in this section we temporarily employ a vector $\overrightarrow{b} = \mathbf{b}^{-*}$. Then converted to this required vector characterization within the encapsulation framework, but using the geometric algebra bivector exponentiation, Eq. (27) reads as:

$$R \boxplus_{\mathrm{b}} \overrightarrow{b} \equiv \mathrm{e}^{\log(\mathrm{R}) + \overrightarrow{b}^{*}}, \tag{28}$$

with the subscript 'b' on $\boxplus$ denoting 'bivector' (as opposed to the 'arc' of Sect. 3.1). The accompanying $\boxminus_{\mathrm{b}}$ is:

$$R_2 \boxminus_{\mathrm{b}} R_1 \equiv \big(\log(R_2) - \log(R_1) \big)^{-*} \tag{29}$$

The associated metric is:

$$d_{\mathrm{b}}(R_2, R_1) = \| R_2 \boxminus_{\mathrm{b}} R_1 \| = \| \log(R_2) - \log(R_1) \|. \tag{30}$$

We have again omitted the dualization, since in geometric algebra this norm can be computed directly on the bivectors by Eq. (4).

In the title of this paper, we have preferred to refer to this metric as based on the *axis*, rather than its *bivector*, since that is a more familiar term and more likely to be used in searches. In the body we prefer 'bivector', for possible generalization to n-D and because arc, bivector and chord are nicely denotable by a, b and c.

Surprisingly, this often used metric (in the sense that many use perturbations in the axis vector, with the angle as its length, as a measure of noise) is not among the six 3D rotation metrics evaluated in [8].

3.3 Chord Metric

We will find it convenient to introduce and use a distance measure $\|R_2 - R_1\|$, the magnitude of the difference between two rotors. In older quaternion literature, this metric is not uncommon, see e.g. [8].

At first sight, characterizing a rotation difference by such a 'rotor chord' would appear to be constrained, since the converse operation of adding a chord-like vector to a rotor must somehow be forced to end on the manifold. We now show that this metric can still be characterized by an unconstrained vector from the tangent space, as the framework of [7] demands.

To incorporate the chord metric into this encapsulation framework, let us reverse engineer the $\boxminus$ and $\boxplus$ from the desired metric d_c. Our chord metric is

$$\begin{aligned} d_c^2(R_1, R_2) &= \|R_2 - R_1\|^2 \\ &= \langle (R_2 - R_1)(\widetilde{R}_2 - \widetilde{R}_1) \rangle_0 \\ &= \langle R_2\widetilde{R}_2 + R_1\widetilde{R}_1 - R_1\widetilde{R}_2 - R_2\widetilde{R}_1 \rangle_0 \\ &= 2 - \langle R_2\widetilde{R}_1 + R_1\widetilde{R}_2 \rangle_0 \\ &= 2(1 - \langle R_2\widetilde{R}_1 \rangle_0) \\ &= 2\big(1 - \cos(\theta/2)\big) \\ &= 4\sin^2(\theta/4), \end{aligned} \tag{31}$$

where θ is the relative rotation angle of the two rotors, defined from $R_2\widetilde{R}_1 = \exp(-\mathbf{I}\theta/2)$, so $\mathbf{I}\theta = -2\log(R_2/R_1)$.

We expect the quantity d_c^2 to be the squared norm of a vector (for the encapsulation framework) or bivector (in the 3D geometric algebra perspective). Starting with the latter, let us employ the usual geometric algebra overloading of the sine function to a 2-blade $\mathbf{X}$ (with negative scalar square $\mathbf{X}^2 < 0$), a map from bivectors to bivectors defined as:

$$\sin(\mathbf{X}) \equiv \sin(\|\mathbf{X}\|)\frac{\mathbf{X}}{\|\mathbf{X}\|}, \tag{32}$$

which follows from the Taylor expansion of the exp-function in geometric algebra. Note that we can then write d_c^2 as $\|2\,\mathbf{I}\,\sin(\theta/4)\|^2 = \|2\,\sin(\mathbf{I}\theta/4)\|^2$. Since this should equal $\|R_2 \boxminus_c R_1\|^2$, we should define as our $\boxminus_c$ operation (with the subscript 'c' denoting 'chord') the Euclidean vector of which the relevant bivector is the dual. This suggests the definition:

$$R_2 \boxminus_{\mathrm{c}} R_1 = -2 \left(\sin \left(\tfrac{1}{2} \log(R_2/R_1) \right) \right) I_3. \tag{33}$$

(As before, the final undualization converts the bivector to a vector, as is required for the strictly Euclidean framework of [7]; but other than that it is cosmetic, we could and will work with the linear 3D bivector space directly.) Working back through the encapsulation framework, it follows that the corresponding $\boxplus_{\mathrm{c}}$ is the Euclidean parametrization:

$$R \boxplus_{\mathrm{c}} \overrightarrow{c} = \mathrm{e}^{-2\,\mathrm{asin}(\overrightarrow{c}^{\,*}/2)}\, \mathrm{R}, \tag{34}$$

employing dualization to convert the Euclidean vector $\overrightarrow{c}$ to the bivector $\mathbf{c} = \overrightarrow{c}^{\,*}$ so that it can be used in the geometric algebra arcsine function (defined as the local inverse of the geometric algebra sine function, so $\mathrm{asin}(\mathbf{X})$ for a 2-blade $\mathbf{X}$ is defined as $\mathrm{asin}(\|\mathbf{X}\|)\,\mathbf{X}/\|\mathbf{X}\|$). Then the corresponding Euclidean metric is indeed as desired:

$$d_{\mathrm{c}}(R_2, R_1) = 2\,|\sin(\theta/4)|. \tag{35}$$

This embedding in the encapsulation framework endows the seemingly naive 'difference of rotors' metric construction with sufficient legitimacy, and permits us to relate it to the other two parametrizations. This metric also occurs in as Φ_2 in [8], who refers to a traditional source for this quaternion difference measure.

Effectively, $\overrightarrow{c}$ is a vector perpendicular to the plane of relative rotation (and hence determines that plane), and the length of the vector is equal to the length of the chord of the (unit circle) arc a vector moves on, thus determining the relative angle. This vector $\overrightarrow{c}$ as parametrization of the relative rotation is therefore fully unconstrained; while the exponentiation in Eq. (34) ensures that the rotor chord nevertheless ends on the unit sphere of rotors.

3.4 The Parametrizations Related

The arc and chord parametrizations by $\mathbf{a} = \overrightarrow{a}^{\,*}$ and $\mathbf{c} = \overrightarrow{c}^{\,*}$ can be easily related, directly from the bivectors of their defining additional rotors:

$$\mathbf{c} = 2\,\sin(\mathbf{a}/2). \tag{36}$$

It is clear that for small values of the perturbation magnitude, these characterizations are equivalent to second order. Moreover, the relationship is monotonic, which will relate the optimal estimators based on them in a simple manner.

The relationship of the additive bivector parametrization by $\mathbf{b} = \overrightarrow{b}^{\,*}$ to the other two is more involved and we cannot give it in closed form, due to non-commutative properties of the exponentials involved. We compute a first order approximation $\mathbf{a} \approx \psi_{\mathbf{B}}(\mathbf{b})$ in detail in the next section. The outcome is:

$$\mathbf{a} \approx \psi_{\mathbf{B}}(\mathbf{b}) \equiv \mathbf{b}_{\parallel} + \mathbf{b}_{\perp}\,\mathrm{sinc}(\mathbf{B})\,\mathrm{e}^{-\mathbf{B}}. \tag{37}$$

Here $\mathbf{b}_{\parallel} = \frac{1}{2}(\mathbf{b} - \mathbf{B}\,\mathbf{b}\,\mathbf{B}^{-1})$ is the component of $\mathbf{b}$ parallel to $\mathbf{B}$, and $\mathbf{b}_{\perp} = \frac{1}{2}(\mathbf{b} + \mathbf{B}\,\mathbf{b}\,\mathbf{B}^{-1})$ the perpendicular component. This split in the roles of parallel and perpendicular parts is understandable: the former is purely the change of the rotation angle, the latter is the change of attitude of the rotation plane, which only partly contributes to the total change.

The mapping $\psi_{\mathbf{B}}(\mathbf{b})$ is linear in $\mathbf{b}$, but strongly non-linear in its dependence on the local value of $\mathbf{B}$, the bivector of the current rotor. We thus find that the most commonly employed characterization of a rotation by an additive axis vector is actually geometrically the most involved. This will have consequences for modeling noise, as we will see.

3.5 *First-Order Conversion of Arc and Bivector Parametrizations*

This section derives Eq. (37) in detail. It may be skipped at first reading.

In the arc parametrization of rotations, it is natural to treat the noise in rotation (or orientation) as *a small rotor applied after the main rotation*:

$$r\,R = \mathrm{e}^{\mathbf{a}}\mathrm{e}^{\mathbf{B}} \approx \mathrm{e}^{\mathbf{B}} + \mathbf{a}\,\mathrm{e}^{\mathbf{B}}, \tag{38}$$

where $\mathbf{B}$ and $\mathbf{a}$ are the large and small bivectors characterizing these rotors. This is not different in principle from doing the noisy rotor *before* the main rotor. For since $R\,r \approx \mathrm{e}^{\mathbf{B}} + \mathrm{e}^{\mathbf{B}}\,\mathbf{a} = \mathrm{e}^{\mathbf{B}} + (\mathrm{e}^{\mathbf{B}}\,\mathbf{a}\,\mathrm{e}^{-\mathbf{B}})\mathrm{e}^{\mathbf{B}}$, that change of order merely differs by a simple linear mapping of the infinitesimal rotor argument, and we know how to deal with the covariance of such transformed noise.

In the bivector parametrization of rotations, it is more natural to encode the noise through a small *additive term in the bivector of the rotor*, so a rotor $R = \mathrm{e}^{\mathbf{B}}$ becomes perturbed as

$$\mathrm{e}^{\mathbf{B}} \;\rightarrow\; \mathrm{e}^{\mathbf{B}+\mathbf{b}}. \tag{39}$$

This appears quite natural, since we have seen before that the space of bivectors which parameterize the rotor is linear. This is also how noise is modeled in the geometric algebra paper [14], and (in its dual classical axis vector parametrization) traditionally by many others (such as [1, 5, 9, 11]).

This additive bivector characterization of the noise is really different from Eq. (38), though we can establish a linear relationship between this 2-blade noise $\mathbf{b}$ and the rotor noise $\mathbf{a}$ used above. That correspondence is also derived in [14] (their equation (39)) using multivector differentiation, but we give our own (brief) derivation to make the present paper self-contained, and show the underlying first-order assumption

explicitly (so that it could be refined). We later propagate the consequences to the covariances used in the Kalman update combination, in Sect. 6.4.

In the following, assume that the rotor argument $\mathbf{B}$ is a pure 2-blade (which squares to a scalar) and invertible, and that all terms of order 2 and higher in $\mathbf{b}$ (and $\mathbf{a}$) can be neglected.

$$
\begin{aligned}
e^{\mathbf{B}+\mathbf{b}} &\approx 1 \\
&\quad + \frac{1}{1!}(\mathbf{B}+\mathbf{b}) \\
&\quad + \frac{1}{2!}\left(\mathbf{B}^2 + \mathbf{B}\mathbf{b} + \mathbf{b}\mathbf{B}\right) \\
&\quad + \frac{1}{3!}\left(\mathbf{B}^3 + 2\mathbf{B}^2\mathbf{b} + \mathbf{B}\mathbf{b}\mathbf{B}\right) \\
&\quad + \frac{1}{4!}\left(\mathbf{B}^4 + 2\mathbf{B}^3\mathbf{b} + 2\mathbf{B}^2\mathbf{b}\mathbf{B}\right) \\
&\quad + \cdots \\
&= 1 + \qquad\qquad \frac{1}{2}\mathbf{B}^{-1}(\mathbf{B}^0\mathbf{b} - \mathbf{b}\mathbf{B}^0) \\
&\quad + \frac{1}{1!}\left(\mathbf{B}^1 + \frac{1}{2}\mathbf{B}^0(\mathbf{b} + \mathbf{B}^{-1}\mathbf{b}\mathbf{B}) + \frac{1}{2}\mathbf{B}^{-1}(\mathbf{B}^1\mathbf{b} - \mathbf{b}\mathbf{B}^1)\right) \\
&\quad + \frac{1}{2!}\left(\mathbf{B}^2 + \frac{2}{2}\mathbf{B}^1(\mathbf{b} + \mathbf{B}^{-1}\mathbf{b}\mathbf{B}) + \frac{1}{2}\mathbf{B}^{-1}(\mathbf{B}^2\mathbf{b} - \mathbf{b}\mathbf{B}^2)\right) \\
&\quad + \frac{1}{3!}\left(\mathbf{B}^3 + \frac{3}{2}\mathbf{B}^2(\mathbf{b} + \mathbf{B}^{-1}\mathbf{b}\mathbf{B}) + \frac{1}{2}\mathbf{B}^{-1}(\mathbf{B}^3\mathbf{b} - \mathbf{b}\mathbf{B}^3)\right) \\
&\quad + \frac{1}{4!}\left(\mathbf{B}^4 + \frac{4}{2}\mathbf{B}^3(\mathbf{b} + \mathbf{B}^{-1}\mathbf{b}\mathbf{B}) + \frac{1}{2}\mathbf{B}^{-1}(\mathbf{B}^4\mathbf{b} - \mathbf{b}\mathbf{B}^4)\right) \\
&\quad + \cdots \\
&= \mathrm{e}^{\mathbf{B}} + \tfrac{1}{2}\mathrm{e}^{\mathbf{B}}\left(\mathbf{b} + \mathbf{B}^{-1}\mathbf{b}\mathbf{B} + \mathbf{B}^{-1}(\mathrm{e}^{\mathbf{B}}\mathbf{b} - \mathbf{b}\mathrm{e}^{\mathbf{B}})\right)
\end{aligned}
\tag{40}
$$

Equating the two expressions Eq. (38) and Eq. (40) involving $\mathbf{b}$ and $\mathbf{a}$ gives

$$
\mathbf{a} \equiv \psi_{\mathbf{B}}(\mathbf{b}) = \frac{1}{2}\mathrm{e}^{\mathbf{B}}\,(\mathbf{b} + \mathbf{B}^{-1}\,\mathbf{b}\,\mathbf{B})\mathrm{e}^{-\mathbf{B}} + \frac{1}{2}\mathbf{B}^{-1}(\mathrm{e}^{\mathbf{B}}\,\mathbf{b}\,\mathrm{e}^{-\mathbf{B}} - \mathbf{b}) \tag{41}
$$

In 3D, the formula cleans up by defining the component $\mathbf{b}_{\parallel}$ of $\mathbf{b}$ which is contained in the $\mathbf{B}$-plane, and the component $\mathbf{b}_{\perp}$ perpendicular to it:

$$
\mathbf{b}_{\parallel} \equiv \frac{1}{2}\mathbf{B}^{-1}(\mathbf{B}\,\mathbf{b} + \mathbf{b}\,\mathbf{B}) \quad \mathbf{b}_{\perp} \equiv \frac{1}{2}\mathbf{B}^{-1}(\mathbf{B}\,\mathbf{b} - \mathbf{b}\,\mathbf{B}) \tag{42}
$$

Then

$$
\mathbf{a} = \mathrm{e}^{\mathbf{B}}\mathbf{b}_{\parallel}\mathrm{e}^{-\mathbf{B}} - \frac{1}{2}\,\mathbf{b}_{\perp}\mathbf{B}^{-1}(\mathrm{e}^{-2\mathbf{B}} - 1) = \mathbf{b}_{\parallel} + \mathbf{b}_{\perp}\frac{\mathrm{e}^{\mathbf{B}} - \mathrm{e}^{-\mathbf{B}}}{2\mathbf{B}}\,\mathrm{e}^{-\mathbf{B}}. \tag{43}
$$

Defining the correspondence as $\mathbf{a} = \psi_{\mathbf{B}}(\mathbf{b})$, we thus obtain:

$$
\mathbf{a} = \psi_{\mathbf{B}}(\mathbf{b}) \equiv \mathbf{b}_{\parallel} + \mathbf{b}_{\perp}\,\mathrm{sinc}(\mathbf{B})\,\mathrm{e}^{-\mathbf{B}}, \tag{44}
$$

where we introduced the sinc function of bivector arguments. By complete analogy to the definition in real analysis, it is defined as $\sin(\mathbf{B})/\mathbf{B}$, and it is easy to show that it equals the scalar $\sin(|\mathbf{B}|)/|\mathbf{B}|$.

Thus $\psi_{\mathbf{B}}()$ is a linear correspondence between the characterization of a perturbation by a small bivector $\mathbf{b}$ and a small arc $\mathbf{a}$. The final form shows that the in-plane component of the bivector (which signifies the change in angle) is unchanged, but that the out-of-plane component (the change in rotation plane attitude) rotates clockwise by $e^{-\mathbf{B}}$ while shrinking by $\mathrm{sinc}(\mathbf{B})$. This is illustrated in Fig. 3, for a normal vector representation of the bivector: the in-plane component of the normal vector of the bivector $\mathbf{b}$ 'winds up' around the normal vector of the $\mathbf{B}$-plane, shortening as it does so.

The inverse mapping of $\psi_{\mathbf{B}}()$ is:

$$\mathbf{b} = \psi_{\mathbf{B}}^{-1}(\mathbf{a}) \equiv \mathbf{a}_{\parallel} + \mathbf{a}_{\perp} \frac{e^{\mathbf{B}}}{\mathrm{sinc}(\mathbf{B})}. \tag{45}$$

Near $|\mathbf{B}| = \pi$, this grows unbounded and the assumption that the bivector $\mathbf{b}$ is small is no longer valid. *This suggests that the 'post-rotor' characterization is better behaved than the 'additive bivector' characterization.* However, a rotor angle of π implies a rotation angle of 2π, so in practice usage of the equations in this region is presumably avoided by a proper disambiguation of the rotor representation, resetting all to be around the zero angle.

We will also need the adjoint $\overline{\psi}_{\mathbf{B}}$ when propagating the covariances. This is simply:

$$\overline{\psi}_{\mathbf{B}} = \psi_{-\mathbf{B}} \tag{46}$$

Derivation: for arbitrary bivectors $\mathbf{b}$ and $\mathbf{y}$, we have (defining the scalar product $X * Y \equiv \langle X\, Y \rangle_0$)

$$\begin{aligned}
\mathbf{y} * \psi_{\mathbf{B}}(\mathbf{b}) &= \mathbf{y} * \left(\mathbf{b}_{\parallel} + \mathbf{b}_{\perp} e^{-\mathbf{B}}\, \mathrm{sinc}(\mathbf{B}) \right) \\
&= \mathbf{y} * \mathbf{b}_{\parallel} + \mathbf{y} * (\mathbf{b}_{\perp} e^{-\mathbf{B}})\, \mathrm{sinc}(\mathbf{B}) \\
&= \mathbf{b}_{\parallel} * \mathbf{y} + \mathbf{b}_{\perp} * (e^{-\mathbf{B}} \mathbf{y})\, \mathrm{sinc}(\mathbf{B}) \quad \text{(by cyclic re-ordering of scalar product)} \\
&= \mathbf{b} * \mathbf{y}_{\parallel} + \mathbf{b} * (e^{-\mathbf{B}} \mathbf{y}_{\perp})\, \mathrm{sinc}(\mathbf{B}) \quad \text{(apply cyclic re-ordering again)} \\
&= \mathbf{b} * \left(\mathbf{y}_{\parallel} + \mathbf{y}_{\perp} e^{\mathbf{B}}\, \mathrm{sinc}(-\mathbf{B}) \right) \quad \text{(anti-commutation, and symmetry of sinc)} \\
&= \mathbf{b} * \psi_{-\mathbf{B}}(\mathbf{y})
\end{aligned}$$

The linearity of the mapping $\psi_{\mathbf{B}}$ means that statistics of $\mathbf{b}$ and $\mathbf{a}$ can be related, but since they are so different (by Eq. (44)), it should be possible to decide which characterization is most suitable for any specific measurement process.

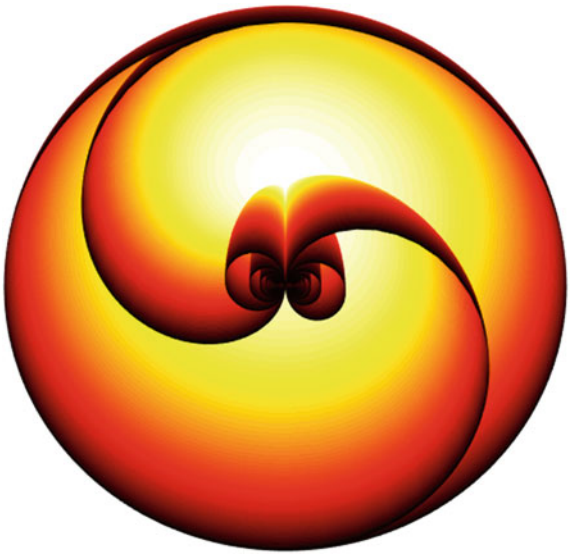


Fig. 3 A comparison of noisy rotor characterization by **a** and **b**, as $e^{\mathbf{a}}e^{\mathbf{B}}$ or $e^{\mathbf{b}+\mathbf{B}}$. This is a 3-dimensional surface of which the points denote the solution for **a** in terms of **b**, as in Eq. (44). Since the relationship scales proportionally, the bivector coefficients of **b** (or if you will, the normal vectors to the bivectors) are made to range over a unit sphere. The sphere is shown cut by the **B**-plane of the main rotor $e^{\mathbf{B}}$ for clarity. Equation (44) causes the perpendicular component of **b** to swirl in the way indicated; the parallel component of **b** is unaffected by this (but is still determined by the fact that **b** is a unit bivector). You may study this swirly shape in 3D in Matlab as: `ezsurf ('cos(s)*sin(t)/ t*sin(t)','cos(s)* sin(t)/t*cos(t)', 'sin(s)*ones(size(t))', [0,pi,0.01,6*pi],300)`

4 Averaging Orientations

The orientational distance measures we introduced have two important properties: first, only the *relative rotation* matters (so that the distance measure is uniform over the whole rotor manifold); second, of the relative rotation only the *angular aspect* contributes to the distance measure, not its plane. Figure 4 shows a rotor (as an arc on the rotation sphere), and some rotors that are all equally different from it for the chord metric d_{c}. Their endpoints all lie on a circle of equal chord length from the result. You can see that some differ mainly in angle, others mainly in rotation plane. It is comforting that a distance measure $\|R_2 - R_1\|$ derived on the rotor manifold has such a clear interpretation for the rotational arcs (and a similar picture can be drawn for the arc metric d_{a}). You should realize, though, that the figure is slightly too specific: since the arc representation can be freely slid along the great circle, we only get *all* rotations with the same distance to the given rotation if we perform this small-circle-construction everywhere along the great circle indicated.

Note that for small rotational differences, the distances of Eq. (26) and Eq. (35) are proportional to the arclength $|\theta/2|$, and this is perhaps half of what one would have expected (by contrast, the rotation examples in [7] use double this value). For

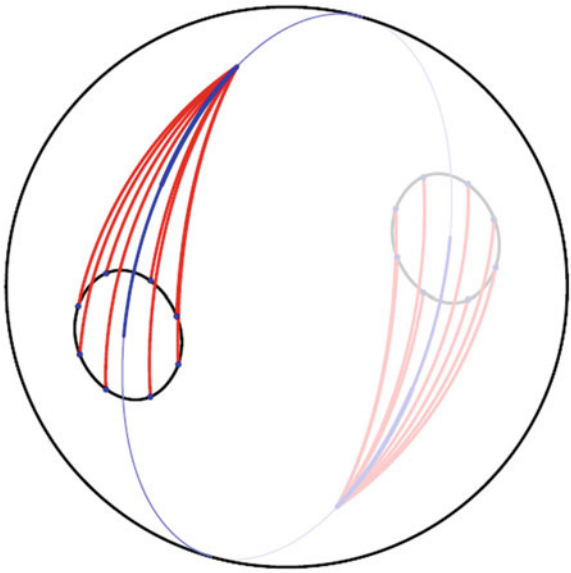


Fig. 4 The chord length metric on rotations

the finite rotation over angle π, the distance is $\sqrt{2}$, for a rotation over 2π the measure is 2, which is its maximum, and only for a rotation of 4π do we get a rotor of distance zero to the original rotation.

This is strange, but strangely correct. It is related to the property of rotors to describe relative rotations rather than absolute rotations (what are those anyway?). You can see this in a 'Balinese dance experiment'. Stand up straight, with your hand immediately in front of your left shoulder, flat, palm up; now make the motions necessary to make your hand turn 2π in its own plane, around its center (you may have to move your torso a little to maintain a planar plate rotation); you find that your elbow sticks up in the air; continue turning the hand in the same direction; when you have reached 4π, you are (surprise!) back in your original pose and ready to do this once more. The moral: the periodicity of the relative rotation of the hand was 4π, not 2π; the maximally different orientation was halfway the experiment, at the 2π rotation, after that the rotational distance decreased again. This is what our distance measure gives as well.

Incidentally, the Balinese experiment also illuminates our earlier remark on the physical significance of the apparent sign multiplicity of the rotor representation. Merely observing the hand, its orientation might just as well be characterized by $R = \mathrm{e}^{-\mathbf{I}\theta/2}$ as by $R' = -R = -\mathrm{e}^{-\mathbf{I}\theta/2}$. However, the latter equals $R' = \mathrm{e}^{-\mathbf{I}\pi}\,\mathrm{e}^{-\mathbf{I}\theta/2} = \mathrm{e}^{-\mathbf{I}(2\pi+\theta)/2}$, so it is more properly interpreted as the rotation over $(2\pi + \theta)$. For the hand, there is no difference, but the elbow stance demonstrates the non-equivalence when one considers the connection of the rotated element to a fixed basis. One could use R and $-R$ as clockwise and counterclockwise ways of achieving the same orientation.

With this intuition in place, we can study the various ways of producing average rotations under the three metrics. So suppose we have a set of measured rotors $\{R_i\}_{i=1}^n$, all with the same accuracy (we will refine this in Sect. 6.3 to 'independently identically distributed'), and let us find the rotor that has the minimum total squared distance to all of them.

4.1 The Minimal Chord Estimator

For the distance measure of the chord metric, we are looking for the rotor R_c that minimizes

$$\begin{aligned}\sum_i d_c^2(R_i, R_c) &= \sum_i \langle (R_i - R_c)\tilde{}\,(R_i - R_c)\rangle_0 \qquad (47)\\ &= 2\sum_i \langle 1 - \widetilde{R}_i\, R_c\rangle_0 \\ &= 2(n - \langle (\sum_i \widetilde{R}_i)\, R_c\rangle_0) \\ &= 2(n - \langle \widetilde{S}\, R_c\rangle_0). \qquad (48)\end{aligned}$$

This derivation involves a rewriting similar to that in the derivation of Eq. (31), and the bilinearity of the scalar product. The quantity $S \equiv \sum_i R_i$, is not a rotor, so let us normalize it to become one:

$$\overline{R} \equiv \frac{S}{\|S\|} = \frac{\sum_i R_i}{\|\sum_i R_i\|} \qquad (49)$$

Now we can rewrite the quantity to be minimized to: $2(n - \|S\|\langle \widetilde{\bar{R}} R_c\rangle_0)$. The first term is a given constant, the second is maximal when $R_c = \bar{R}$, so that is the solution.

The least-square chord length error estimate $R_c \equiv \mathrm{argmin}_R \sum_i \|R - R_i\|^2$ *of a set of independent identically distributed rotors* R_i *is their normalized sum*

$$R_c = \frac{\overline{R}}{\|\overline{R}\|}, \qquad (50)$$

with $\overline{R} \equiv \frac{1}{n}\sum_i R_i$.

This is the proper formula for the 'mean rotor' of an ensemble under the chord metric. The form of the resulting estimator is not that different from translations, where the best linear estimate of a set of measurements $\{\mathbf{x}_i\}_{i=1}^n$ of the same position would be the mean $\hat{\mathbf{x}} = (\sum_i \mathbf{x}_i)/n$.

Some care is required when averaging these rotors, due to the double representation of orientations, in which $-R_i$ may represent the same rotation as R_i (the issue

we mentioned at the end of Sect. 2.3). If you have been sloppy in casting your measurement data into rotors, you may have both forms and should not average over them; then you first need to multiple all R_i by the sign of their scalar part, to ensure that $\langle R_i \rangle_0 \geq 0$.

4.2 The Minimal Bivector Estimator

In the bivector metric, we measured the magnitude of the squared difference in characterizing bivectors of rotors. Finding an estimator that minimizes this least-square criterion is easy: it is just achieved by averaging the bivectors of the ensemble and exponentiating.

The least-square bivector error estimate $R_\mathrm{b} \equiv \mathrm{argmin}_R \sum_i \| \log(R) - \log(R_i) \|^2$ *of a set of independent identically distributed rotors* R_i *is the exponential of the average bivector*

$$R_\mathrm{b} = \exp(\overline{\log(R)}), \tag{51}$$

with $\overline{\log(R)} \equiv \frac{1}{n} \sum_i \log(R_i)$.

This is the direct consequence of the standard quadratic form of the criterion in terms of $\log R$ and the fact that the $\log()$ is a monotonic function which does not affect the optimum. For, taking the directional derivative of the criterion we find (by the rules of geometric differentiation, see e.g. [3]):

$$\begin{aligned} 0 &= (A * \partial_R) \sum_i \| \log(R) - \log(R_i) \|^2 \\ &= 2 \sum_i \big(\log(R) - \log(R_i) \big)^\sim (A * \partial_R) \log(R) \\ &= 2n \big(\log(R) - \overline{\log(R)} \big)^\sim (A * \partial_R) \log(R) \end{aligned}$$

and since this needs to hold for all A, the result follows.

The considerations on standardizing rotors and their bivectors of the ensemble to the principal values that we mentioned above for the estimator R_c also apply to R_a, for functionally equivalent rotors represented by angles that are not in the same range would cause problems when averaging the corresponding bivectors. So again, ensure that $\langle R_i \rangle_0 \geq 0$ beforehand, and use the principal value of the logarithm, returning an angle in the range $(-\pi, \pi)$.

4.3 The Minimal Arc Estimator

Another reasonable criterion to minimize is the sum of squared arc lengths d_a^2, from the estimated rotor to all noisy rotors:

$$R_{\mathrm{a}} = \mathrm{argmin}_R \sum_i \| \log(R\, \widetilde{R}_i) \|^2. \tag{52}$$

We define $B_i = \log(R_i)$ and $X = \log(R)$, then perform a differentiation by varying the rotor R through its bivector X, so as to maintain the rotor nature. We derive a first-order approximation to the estimator by demanding that it extremizes the criterion, to first order in the perturbation. The non-commutative products make a closed form solution hard, so we employ the first few terms of the Baker-Campbell-Hausdorff (BCH) formula:

$$\mathrm{e}^X \mathrm{e}^Y \approx \exp(X + Y + X \times Y + \tfrac{1}{3}\big(X \times (X \times Y) + Y \times (Y \times X)\big) + \text{higher order commutators} \tag{53}$$

with $X \times Y \equiv \frac{1}{2}(X\,Y - Y\,X)$. We cut this after the term $X \times Y$; this is allowed if X and Y are small, since otherwise the series is not convergent. We can therefore not use it to bring $R\,\widetilde{R}_i$ into a single exponential of which could then take the logarithm, but have to proceed more subtly. As a shorthand, let us denote the bivector $\log(\mathrm{e}^{\mathbf{X}}\,\mathrm{e}^{-\mathbf{B}_i})$ by $\mathbf{r}_i$, and assume it to be small (not infinitesimal, but small enough to apply it in BCH). In the directional derivative, another bivector $\varepsilon\mathbf{A}$ occurs; since we are taking the limit for vanishing ε, that is definitely small. Now compute:

$$\begin{aligned}
0 &= (\mathbf{A} * \partial_{\mathbf{X}}) \sum_i \| \log(\mathrm{e}^{\mathbf{X}}\,\mathrm{e}^{-\mathbf{B}_i}) \|^2 \\
&= \lim_{\varepsilon\to 0} \frac{1}{\varepsilon} \sum_i \Big(\| \log(\mathrm{e}^{\mathbf{X}+\varepsilon\mathbf{A}}\,\mathrm{e}^{-\mathbf{B}_i}) \|^2 - \| \log(\mathrm{e}^{\mathbf{X}}\,\mathrm{e}^{-\mathbf{B}_i}) \|^2 \Big) \\
\overset{\text{Eq. (37)}}{=} &\lim_{\varepsilon\to 0} \frac{1}{\varepsilon} \sum_i \Big(\| \log(\mathrm{e}^{\psi_{\mathbf{X}}(\varepsilon\mathbf{A})}\mathrm{e}^{\mathbf{X}}\,\mathrm{e}^{-\mathbf{B}_i}) \|^2 - \| \log(\mathrm{e}^{\mathbf{X}}\,\mathrm{e}^{-\mathbf{B}_i}) \|^2 \Big) \\
&= \lim_{\varepsilon\to 0} \frac{1}{\varepsilon} \sum_i \Big(\| \log(\mathrm{e}^{\psi_{\mathbf{X}}(\varepsilon\mathbf{A})}\mathrm{e}^{\mathbf{r}_i}) \|^2 - \| \log(\mathrm{e}^{\mathbf{r}_i}) \|^2 \Big) \\
\overset{\text{BCH}}{\approx} &\lim_{\varepsilon\to 0} \frac{1}{\varepsilon} \sum_i \Big(\| \log(\mathrm{e}^{\psi_{\mathbf{X}}(\varepsilon\mathbf{A})+\mathbf{r}_i+\psi_{\mathbf{X}}(\varepsilon\mathbf{A})\times\mathbf{r}_i}) \|^2 - \| \log(\mathrm{e}^{\mathbf{r}_i}) \|^2 \Big) \\
&= \lim_{\varepsilon\to 0} \frac{1}{\varepsilon} \sum_i \Big(\| \psi_{\mathbf{X}}(\varepsilon\mathbf{A}) + \mathbf{r}_i + \psi_{\mathbf{X}}(\varepsilon\mathbf{A}) \times \mathbf{r}_i \|^2 - \| \mathbf{r}_i \|^2 \Big) \\
&= \sum_i \Big(\mathbf{r}_i \left(\psi_{\mathbf{X}}(\mathbf{A}) + \psi_{\mathbf{X}}(\mathbf{A}) \times \mathbf{r}_i \right) + \left(\psi_{\mathbf{X}}(\mathbf{A}) + \psi_{\mathbf{X}}(\mathbf{A}) \times \mathbf{r}_i \right) \mathbf{r}_i \Big) \\
&\approx \sum_i \Big(\mathbf{r}_i\, \psi_{\mathbf{X}}(\mathbf{A}) + \psi_{\mathbf{X}}(\mathbf{A})\, \mathbf{r}_i \Big) \\
&= \overline{\mathbf{r}}\, \psi_{\mathbf{X}}(\mathbf{A}) + \psi_{\mathbf{X}}(\mathbf{A})\, \overline{\mathbf{r}}
\end{aligned}$$

where the $\approx$ equalities employ the smallness of $\mathbf{r}_i$. This needs to be zero for all $\mathbf{A}$. Since $\psi_{\mathbf{X}}(\mathbf{A})$ is a linear function, containing both components parallel and perpen-

dicular to $\mathbf{X}$, and hence commutating and anti-commutating with it, the only way to satisfy the equation is to demand that $\bar{\mathbf{r}} = 0$, which implies for the arc-based estimator R_a:

$$\sum_i \log(R_\mathrm{a}\, \widetilde{R}_i) = 0. \tag{54}$$

So, the sum of the bivectors of the relative rotors to the optimal arc rotor R_a should be zero. Since the criterion of Eq. (52) is non-negative, demanding it to be zero would certainly give a minimum; therefore the approximations we had to make in the derivation do not jeopardize this result. It is interesting that this solution involves both the scalar quantitative aspects (the arc lengths) and the spatial aspects (the orientations of the relative rotation planes).

Yet I do not know how to solve Eq. (54) in closed form, so at this moment we can only state:

The least-square arc error estimate $R_\mathrm{a} \equiv \mathrm{argmin}_R \sum_i \| \log(R/R_i) \|^2$ *of a set of independent identically distributed rotors* R_i *is the solution* R_a *of*

$$\sum_i \log(R_\mathrm{a}\, \widetilde{R}_i) = 0. \tag{55}$$

The best we can do at this point is to use the closeness of chord and arc for the small angles involved, and use the straightforward minimum chord estimator R_c as a proxy for the minimum arc estimator R_a. In an iterative solution to the defining equation Eq. (55), it should be a good starting point. The above derivation then provides the directional derivative for a gradient descent method to converge to the true optimum.

Simulations using 100 noisy rotors with a reasonable Gaussian noise level of 0.2 rad standard deviation on the bivector angle show that the normalized average rotor R_c, designed to minimize the sum of squared chord lengths, indeed has a very small total arc sum with a net angle of the order of 5×10^{-4}. The minimum chord estimator R_c is therefore a very good approximation to optimal arc rotor R_a.

4.4 Averages Compared

We can consider the estimators R_a, R_b, R_c from the point of view of perturbation, and this provides some additional insights. With an ensemble of rotors $R_i = \exp(\mathbf{a}_i)\, R$ perturbing the original rotor R (in the arc-noise perturbation representation), the average chord rotor R_c amounts to normalizing the average rotor $\overline{R} = \overline{\exp(\mathbf{a})}\, R$. Normalization to construct R_c from $\overline{R}$ requires division by

$$
\begin{aligned}
\|\overline{\exp(\mathbf{a})}\, R\| &= \|\overline{\exp(\mathbf{a})}\| \\
&= \sqrt{(1+\bar{\mathbf{a}}+\tfrac{1}{2}\overline{\mathbf{a}^2}+\cdots)\cdot(1-\bar{\mathbf{a}}+\tfrac{1}{2}\overline{\mathbf{a}^2}+\cdots)} \\
&= \sqrt{1+(\overline{\mathbf{a}^2}-\bar{\mathbf{a}}^2)+\cdots} \\
&= \sqrt{1+\mathrm{var}(\mathbf{a})+\cdots} \\
&\approx 1+\tfrac{1}{2}\mathrm{var}(\mathbf{a})+\cdots,
\end{aligned}
$$

It follows that, to second order:

$$
\begin{aligned}
R_\mathrm{c} &= \bar{R}/\|\bar{R}\| \\
&\approx (1-\tfrac{1}{2}\mathrm{var}(\mathbf{a}))\,\overline{\exp(\mathbf{a})}\, R \\
&= \big(1-\tfrac{1}{2}(\overline{\mathbf{a}^2}-\bar{\mathbf{a}}^2)+\cdots\big)\,(\mathbf{a}+\bar{\mathbf{a}}+\tfrac{1}{2}\overline{\mathbf{a}^2}+\cdots)\, R \\
&= \big(1+\bar{\mathbf{a}}+\tfrac{1}{2}\bar{\mathbf{a}}^2+\cdots\big)\, R \\
&\approx \exp(\bar{\mathbf{a}})\, R.
\end{aligned}
$$

By computing the normalized average rotor R_c, we are therefore effectively averaging the bivectors of the perturbing rotors (to second order). Since their mean $\bar{\mathbf{a}}$ is expected to be zero (with a small standard error of the mean of $\sqrt{\mathrm{var}(\mathbf{a})/n}$), we expect this estimator R_c of R based on the data R_i to be quite precise.

We can establish the relationship of the minimum bivector rotor R_b to this way of considering the chord rotor R_c. Its analysis can be performed by first converting the individual $R_i = \exp(\mathbf{a}_i)R$ to the bivector representation, by the first-order method of Eq. (37). Defining $\mathbf{B} = \log(R)$ this is, to first order in the small quantity $\mathbf{a}_i$:

$$
\begin{aligned}
R_i &= \exp(\mathbf{a}_i)\, R \\
&= \exp(\mathbf{a}_i)\, \exp(\mathbf{B}) \\
&\approx \exp\big(\mathbf{B}+\psi_\mathbf{B}^{-1}(\mathbf{a}_i)\big).
\end{aligned}
$$

Now averaging the bivectors in the exponents of the R_i (done by taking the log, averaging, and re-exponentiating) leads to the average bivector-based optimal rotor R_b. Using the linearity (to first order) of the mapping $\psi_\mathbf{B}^{-1}()$, we find:

$$
\begin{aligned}
R_\mathrm{b} &= \exp\big(\mathbf{B}+\overline{\psi_\mathbf{B}^{-1}(\mathbf{a})}\big) \\
&= \exp\big(\mathbf{B}+\psi_\mathbf{B}^{-1}(\bar{\mathbf{a}})\big) \\
&\approx \exp(\bar{\mathbf{a}})\, R,
\end{aligned}
$$

where we converted back using the inverse mapping of $\psi_\mathbf{B}()$ (again to first order).

The two estimators 'normalized average rotor' R_c (which minimizes the chords) and the 'average bivector rotor' R_b (which minimizes the bivector norm) therefore agree to first order. This is confirmed by comparing them directly, for instance by computing their difference:

$$
\begin{aligned}
R_\mathrm{b} - R_\mathrm{c} &\approx \left(\exp(\bar{\mathbf{a}}) - (1 - \tfrac{1}{2}\mathrm{var}(\mathbf{a}))\, \overline{\exp(\mathbf{a})} \right) R \\
&\approx \left((1 + \bar{\mathbf{a}} + \tfrac{1}{2}\bar{\mathbf{a}}^2) - (1 - \tfrac{1}{2}\mathrm{var}(\mathbf{a}))\, (1 + \bar{\mathbf{a}} + \tfrac{1}{2}\overline{\mathbf{a}^2}) \right) R \\
&= \left(\tfrac{1}{2}\bar{\mathbf{a}}^2 - \tfrac{1}{2}\overline{\mathbf{a}^2} + \tfrac{1}{2}\mathrm{var}(\mathbf{a})\, (1 + \bar{\mathbf{a}} + \tfrac{1}{2}\overline{\mathbf{a}^2}) \right) R \\
&= \left(-\tfrac{1}{2}\,\mathrm{var}(\mathbf{a}) + \tfrac{1}{2}\,\mathrm{var}(\mathbf{a})\, (1 + \bar{\mathbf{a}} + \tfrac{1}{2}\overline{\mathbf{a}^2}) \right) R \\
&\approx \tfrac{1}{2}\mathrm{var}(\mathbf{a})\, \bar{\mathbf{a}}\, R.
\end{aligned}
$$

Since the noisy bivectors are assumed to be unbiased, $\bar{\mathbf{a}}$ will be very small (the standard error of the mean is $\sqrt{\mathrm{var}(\mathbf{a})/n}$), and hence so is the difference of the estimators.

5 Interpolating Rotations

In many applications, notably in computer graphics, one needs to interpolate between given orientations. Here representation matters greatly: interpolating between rotation matrices gives many undesirable effects. The introduction of quaternions into the field by Shoemake [15] showed how such issues are much more easily tractable in that representation, and boosted their rapid adoption. We briefly repeat the basic derivation, now using rotors, to set up our Kalman combination in the next section.

5.1 Interpolation: The slerp

If we would have two position vectors $\mathbf{x}_0$ and $\mathbf{x}_1$, we would linearly interpolate between them using the familiar

$$
\mathsf{lerp}(\mathbf{x}_0, \mathbf{x}_1; \lambda) = (1 - \lambda)\mathbf{x}_0 + \lambda \mathbf{x}_1, \tag{56}
$$

with λ running from 0 to 1. For interpolation between two rotors R_0 and R_1, the analogous formula (i.e. $(1 - \lambda)R_0 + \lambda R_1$) would not work since the result would not be a rotor; and renormalization would make the interpolation non-uniform.

We would like to have a rotation that interpolates rotation in the sense that it divide the total rotation between R_0 and R_1 in a linear manner into smaller incremental rotations. Visualizing this on the sphere, the intermediate rotations have rotors which are pictured as arcs to evenly spaced intermediate points on the completing side of the spherical triangle, as in Fig. 5. So we need to find the relative rotation and split it up into equal parts. We write the relative rotation as:

$$
R_1 \widetilde{R}_0 \equiv \mathrm{e}^{-\mathbf{I}\theta/2} = \cos(\theta/2) - \mathbf{I}\sin(\theta/2), \tag{57}
$$

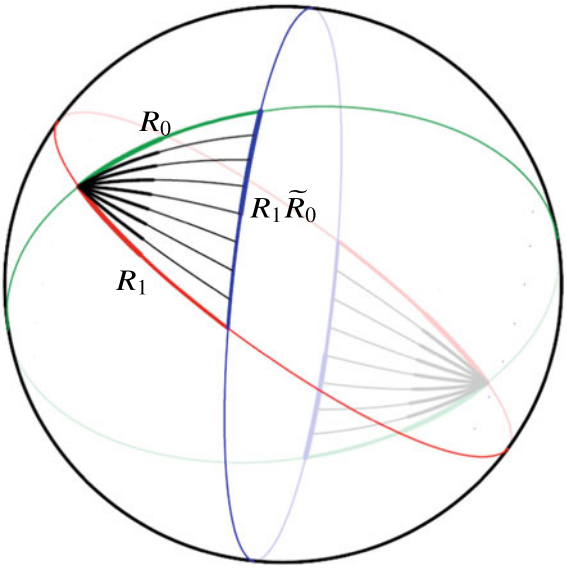


Fig. 5 Illustration of the interpolation of rotors R_0 and R_1, by intermediate rotors

so that $\mathbf{I}$ denotes the relative rotation plane, θ is the relative rotation angle, and $\theta/2$ is the relative rotor angle. This gives $\mathbf{I}R_0 = (R_1 - R_0\cos(\theta/2))/\sin(\theta/2)$. Then the linear interpolation is achieved by rotation over a fraction $\lambda(\theta/2)$ of the angle, from R_0 towards R_1. That is the rotor $\mathrm{e}^{-\lambda\mathbf{I}\theta/2}\mathrm{R}_0$, which may be expressed as:

$$\begin{aligned} R_\lambda &= \big(\cos(\lambda\theta/2) - \mathbf{I}\sin(\lambda\theta/2)\big)\, R_0 \\ &= \frac{R_0\sin(\theta/2)\cos(\lambda\theta/2) - R_0\cos(\theta/2)\sin(\lambda\theta/2) + R_1\sin(\lambda\theta/2)}{\sin(\theta/2)} \\ &= \frac{\sin((1-\lambda)\theta/2)}{\sin(\theta/2)}\, R_0 + \frac{\sin(\lambda\theta/2)}{\sin(\theta/2)}\, R_1 \equiv \mathsf{slerp}(R_0, R_1; \lambda) \end{aligned} \tag{58}$$

This is the linear interpolation formula for rotations, which we abbreviate as $\mathsf{slerp}[R_0, R_1; \lambda]$ (following [15] who derived it for quaternions). It is slightly more involved than linear interpolation of vectors since the linear interpolation is now 'under the sine'.[2] Please note that the angles involved in the interpolation are the rotor angles, which are *half* the rotation angles. When R_0 and R_1 are not very different, $\theta/2$ is small, and Eq. (58) becomes a linear interpolation of rotors: $\mathsf{slerp}(R_0, R_1; \lambda) \approx \mathsf{lerp}(R_0, R_1; \lambda)$.

In computer graphics, more general interpolations (such as Bézier) between a set of key rotations are done by extending this principle. The fact that this was demonstrated to be possible in [15] was one of the motivations for that field to begin using quaternions rather than rotation matrices for their representation of rotations

[2] Yet comparing to Eq. (56), there is really nothing new under the sine—or one could remember that the lerp is now 'living in sin( )'.

and orientations. In geometric algebra, one could naturally perform the identical techniques in the linear space of bivectors of the relative rotors.

For purposes later on, we can rewrite the slerp formula in terms of the sum and difference of the two rotors, by use of some trig identities[3]:

$$\begin{aligned}\mathsf{slerp}(R_0, R_1; \lambda) &= \frac{\sin((1-\lambda)\theta/2)}{\sin(\theta/2)} R_0 + \frac{\sin(\lambda\theta/2)}{\sin(\theta/2)} R_1 \\ &= \frac{\cos\left((2\lambda-1)\,\theta/4\right)}{\cos(\theta/4)} \frac{R_1+R_0}{2} + \frac{\sin\left((2\lambda-1)\,\theta/4\right)}{\sin(\theta/4)} \frac{R_1-R_0}{2} \\ &= \frac{\cos(\rho\,\theta/4)}{\cos(\theta/4)} \frac{R_1+R_0}{2} + \frac{\sin(\rho\,\theta/4)}{\sin(\theta/4)} \frac{R_1-R_0}{2}, \end{aligned} \tag{59}$$

introducing $\rho \equiv 2\lambda - 1$ varying from -1 to 1 as λ varies from 0 to 1.

5.2 Chord-Based Interpolation

In practice, one often applies chord-based interpolation, to avoid the sine evaluation occurring in Eq. (58) or Eq. (59). This involves simply performing a linear interpolation of the end-points of a chord; and then rescaling the result to be an equal-radius rotation. This operation is based on *points*, not on the rotors themselves, and therefore leads to a sequence of approximately-equi-angle-rotated points.

In the more powerful applications of geometric algebra, one really would like to interpolate at the rotor level, since the interpolated rotors can then be used to rotate arbitrary elements. To compute the chord-interpolated rotor, linearly interpolate between the rotors 1 and $R_2\widetilde{R}_1 = \cos(\theta/2) - I\sin(\theta/2)$. This gives:

$$\begin{aligned} R_{\mathrm{c}}(\lambda) &= \mathrm{normalize}\big((1-\lambda)1 + \lambda\,(\cos(\theta/2) - I\,\sin(\theta/2)\big) \\ &= \mathrm{normalize}\big(1 - 2\lambda\,\sin^2(\theta/4) - 2\lambda\,I\,\sin(\theta/4)\,\cos(\theta/4)\big) \\ &= \mathrm{normalize}\big(1 - 2\lambda\,\sin(\theta/4)(\sin(\theta/4) + I\cos(\theta/4)\big) \\ &= \mathrm{normalize}\big(1 - 2\lambda\,\sin(\theta/4)\,I\exp(-I\theta/4)\big) \\ &= \frac{1 - 2\lambda\,\sin(\theta/4)\,I\exp(-I\theta/4)}{\sqrt{1 + 4\lambda^2\,\sin^2(\theta/4)}}. \end{aligned} \tag{60}$$

This computation of the approximate rotor involves one trig function, and a square root; it is perhaps somewhat more efficient than the exact angle-interpolated rotor. But since one expects to compute it in a situation where the rotor is to be applied to multiple elements, the savings are hardly worth it.

[3]Like $\sin((1-\lambda)\theta/2) + \sin(\lambda\theta/2) = \sin(\theta/2)\cos(\lambda\theta/2) - \cos(\theta/2)\sin(\lambda\theta/2) + \sin(\lambda\theta/2) = 2\sin(\theta/4)\cos(\theta/4)\cos(\lambda\theta/2) + 2\sin^2(\theta/4)\sin(\lambda\theta/2) = 2\sin(\theta/4)\cos(\theta/4 - \lambda\theta/2)$.

As for the point-based interpolation, for larger angles θ one can first compute a mid-rotor $\sqrt{R}$ (in 3D, the normalized $(1 + R)$, in n-D more involved [4]), and then perform the chord interpolation twice on the half chords (repeating the process if accuracy requires).

5.3 Bivector-Based Interpolation and Filtering

For the bivector metric, one would perform the difference $\boxminus_{\mathrm{b}}$ of Eq. (29) in linearly interpolated fashion. This produces the intermediate rotor:

$$\begin{aligned} R_{\mathrm{b}}(\lambda) &= \exp\big((1-\lambda)\,\log(R_1) + \lambda\,\log(R_2)\big) \\ &= \exp\big(-(1-\lambda)\,\mathbf{I}_1\theta_1/2 - \lambda\,\mathbf{I}_2\theta_2/2\big) \\ &= \exp\Big(-\tfrac{1}{2}\big(\frac{\mathbf{I}_1\theta + \mathbf{I}_2\theta_2}{2} - \rho\,\frac{\mathbf{I}_2\theta_2 - \mathbf{I}_1\theta_1}{2}\big)\Big), \end{aligned} \tag{61}$$

with $\rho = 2\lambda - 1$. Since the bivectors form a linear space (even in n-D), this is well-defined and the outcome is an intermediate rotor.

Because of the linearity of the bivector space, more general interpolation techniques can be applied as well, such as rational Bézier employing Bernstein polynomials on bivectors (adapting them from being applied to axis vectors by duality), see e.g. [10] for a quaternion formulation. One can even lift the intuition and techniques from linear filtering to the exponential argument of the rotor (or quaternion). This is a powerful way to, say, 'smooth' a sequence of orientations, or to 'sharpen' it. Lee and Shin [12] give the explicit computation of the coefficients for such quaternion filters based on applying the classical linear techniques in the tangent space to the unit quaternion manifold.

6 Statistical Processing of Orientations

To predict orientations, we look at how an orientation is evolving in its rotational aspects, and give a good guess of what it could be at the next time instant. Our derivation of the interpolation formula shows that this can also be used for extrapolation: just extend the interpolated arc from R_0 to R_1 beyond R_1, by taking $\lambda > 1$. Once we have decided how far to extend it, a prediction for some subsequent rotation R_2 has been obtained. If we also have measurement data available about the actual value of R_2, with some noise, then we would like to combine these two optimally. This is the update step of any discrete-time Kalman filter, where the optimal combination is done statistically, to produce a minimum variance unbiased linear estimate [13]. In this section, we produce the optimal weighting procedure for prediction and

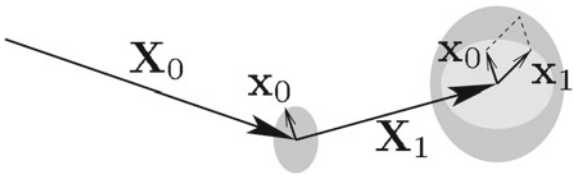


Fig. 6 Propagation of errors in translations

observation of orientations. To develop this weighting, we obviously first need to understand how to represent and compute confidence measures in orientations.

6.1 Propagating Orientational Noise

Let us recall how we usually combine noise in translation measurements. Suppose we have a translation vector $\mathbf{X}_0$, and a translation $\mathbf{X}_1$, both with some noise. To see what the distribution of the result $\mathbf{X}_0 + \mathbf{X}_1$ is, we need to do some averaging: take an error vector $\mathbf{x}_0$ (located at the tangent space at the endpoint of $\mathbf{X}_0$) and an error vector $\mathbf{x}_1$ (located at the tangent space at the endpoint of $\mathbf{X}_1$); these produce the error vector $\mathbf{x}_0 + \mathbf{x}_1$ at the endpoint of $\mathbf{X}_0 + \mathbf{X}_1$. Drawing $\mathbf{X}_0$ and $\mathbf{X}_1$ head to tail, we have basically 'transferred' the error $\mathbf{x}_0$ to the end of $\mathbf{X}_0 + \mathbf{X}_1$, and added, see Fig. 6. This is permissible: translations are commutative, and the tangent spaces are everywhere isomorphic, and even isomorphic to the Euclidean space itself so that this transfer also works for non-infinitesimal displacements. In particular, we are allowed to write $(\mathbf{x}_1 + \mathbf{X}_1) + (\mathbf{x}_0 + \mathbf{X}_0)$ as

$$(\mathbf{x}_1 + \mathbf{x}_0) + (\mathbf{X}_1 + \mathbf{X}_0) \equiv \mathbf{x}_t + \mathbf{X}_t, \tag{62}$$

so that the 'quantity plus error' retains its form, and the total error $(\mathbf{x}_1 + \mathbf{x}_0)$ is independent of the data $\mathbf{X}_0$ and $\mathbf{X}_1$. If we now want to average this for distributions of $\mathbf{x}_0$ and $\mathbf{x}_1$, then the total distribution becomes the convolution of the contributing distributions, integrating for a given error $\mathbf{x}_t$ the ways in which it can be produced from $\mathbf{x}_0$ and $\mathbf{x}_1$:

$$p_t(\mathbf{x}_t) = \int p_0(\mathbf{x}_0)\ p_1(\mathbf{x}_t - \mathbf{x}_0)\ d\mathbf{x}_0 \equiv (p_0 * p_1)(\mathbf{x}_t). \tag{63}$$

For noise characterization, the Gaussian family of distributions is particularly suitable, since convolution of two Gaussians is again a Gaussian, merely coarser. To be precise, denoting the Gaussian with covariance matrix C by $\mathcal{G}[\mathsf{C}]$, the resulting distribution is simply:

$$\mathcal{G}[\mathsf{C}_0] * \mathcal{G}[\mathsf{C}_1] = \mathcal{G}[\mathsf{C}_0 + \mathsf{C}_1]. \tag{64}$$

In an isotropic situation, the equi-probability lines of the resulting pdf are equidistant lines of the metric induced by the covariance matrix.

Compare these straightforward results for translations to the situation with rotations. The main issue is that rotations are *not* commutative, so we have to choose where to represent the noise. Let us first represent it by a small rotor applied after the main rotor, bearing in mind Fig. 4 of 'similar rotors'. So for rotor R_0, we multiply by a small rotor r_0 performed 'after' the main rotation to obtain $r_0 R_0$. Now apply a second rotor R_1 with similar noise and attempt to rewrite to a form showing the resulting rotor $R_1 R_0$, accompanied by some little noise rotor:

$$(r_1\, R_1)\,(r_0\, R_0) = (r_1\, R_1\, r_0\, \widetilde{R}_1)\,(R_1\, R_0) \equiv r_t\, R_t, \tag{65}$$

implying that $r_t = r_1\, R_1\, r_0\, \widetilde{R}_1$. This shows that while the resulting noise is still a small rotor, the initial main rotor R_1 plays an essential role in how the noise transfers. It implies that the composition of distributions of rotors is not simply a convolution, but assumes the form:

$$p_t(r_t) = \int p_0(r_0)\; p_1(r_t\, R_1\, \widetilde{r}_0\, \widetilde{R}_1\,)\; dr_0 \tag{66}$$

That the pdfs are defined in terms of rotors is less of a problem than the complicated argument of p_1. This composition is not a convolution, and not closed for the Gaussian family.

It is perhaps possible to find a 'natural' noise model with parameters that transform simply under this transfer law (but the simple and common von Mises distribution does not; I checked). I do not know how to treat this composition in general. In an approximation for small noise, though, the composition becomes tractable, and this is naturally the approach taken by many (such as [1, 9]). We will follow that approach in the remainder of this paper.

6.2 Composition of Narrow Orientation Distributions

If the noise rotors are small, we can approximate them and study their 'infinitesimal interaction', which is simpler than the general interaction. For a small rotor r_0 in our metric, the distance between it and the identity should be small, and therefore its rotor angle θ_0 is small. This implies that we can write, to first order[4]:

$$r_0 = \mathrm{e}^{-\mathbf{I}_0\theta_0/2} = \cos\theta_0/2 - \mathbf{I}_0 \sin\theta_0/2 \approx 1 - \mathbf{I}_0\theta_0/2 \equiv 1 - \mathbf{i}_0, \tag{67}$$

where we defined $\mathbf{i}_0$ as the small bivector $\mathbf{I}_0\theta_0/2$, and we define $\mathbf{i}_1$ an $\mathbf{i}_t$ similarly. (This approximation is rather good, for 10° noise in rotation angle the error is about 0.4%;

[4]The first order equivalence of all three metrics introduced implies that this model, based on the arc-characterization, also applies to the others.

5% error is reached for 36°.) Expanding the composition of a total rotor $R_t \equiv R_1 R_0$ and its noise r_t now gives:

$$r_t\, R_t = (r_1\, R_1)\,(r_0\, R_0) \approx (1 - \mathbf{i}_1 - R_1\, \mathbf{i}_0\, \widetilde{R}_1)\,(R_1 R_0) \equiv (1 - \mathbf{i}_t)\, R_t. \tag{68}$$

It follows that for small amounts of noise, the propagation is simply:

$$\mathbf{i}_t = \mathbf{i}_1 + R_1\, \mathbf{i}_0\, \widetilde{R}_1 = \mathbf{i}_1 + \mathsf{R}_1 \mathbf{i}_0 \tag{69}$$

We rewrote this outcome using the linear mapping R_1 (the rotation, in sans serif font), to simplify notation for the following derivation. Of course, Eq. (69) involves bivectors; but since these elements are merely dual to vectors, defining a distribution over them is straightforward.

If we have a distribution p_0 of $\mathbf{i}_0$ and an independent distribution p_1 of $\mathbf{i}_1$, this induces a distribution p_t on $\mathbf{i}_t$. Using $\mathbf{i}_0$ as the parametrization, this distribution can be integrated as:

$$p_t(\mathbf{i}_t) = \int p_0(\mathbf{i}_0)\; p_1(\mathbf{i}_t - \mathsf{R}_1 \mathbf{i}_0)\; d\mathbf{i}_0 \equiv (p_0 *_{\mathsf{R}_1} p_1)(\mathbf{i}_t). \tag{70}$$

The symbol $*_{\mathsf{R}_1}$ denotes that this resembles, but is different from, a convolution: we have rotation R_1 in the second argument, so it is a 'convolution with a turn'.

Although it is not quite a convolution, the Gaussian family of functions is still closed under this composition, and can therefore be used in our thinking about covariance. In fact, denoting the Gaussian with covariance matrix C by $\mathcal{G}[\mathsf{C}]$, we have:

$$\mathcal{G}[\mathsf{C}_0] *_{\mathsf{R}} \mathcal{G}[\mathsf{C}_1] = \mathcal{G}[\mathsf{R}\mathsf{C}_0\overline{\mathsf{R}}] * \mathcal{G}[\mathsf{C}_1] = \mathcal{G}[\mathsf{R}\mathsf{C}_0\overline{\mathsf{R}} + \mathsf{C}_1] \tag{71}$$

where $\overline{\mathsf{R}}$ is the adjoint of R, representable by the transpose of the matrix of R. This result is immediate from the combination of covariances of distributions (see e.g. [13]), since in Eq. (69) $\mathbf{i}_t$ is a linear combination of $\mathbf{i}_0$ and $\mathbf{i}_1$. The sandwiching between rotation mappings is just how covariance matrices rotate, through the usual linear algebra of bilinear forms. Notice that the exponent occurring in the computation of the convolution can be rewritten (omitting a factor $-\frac{1}{2}$):

$$\mathbf{i}_0^T \mathsf{C}_0^{-1} \mathbf{i}_0 + (\mathbf{i}_t - \mathsf{R}\mathbf{i}_0)^T \mathsf{C}_1^{-1} (\mathbf{i}_t - \mathsf{R}\mathbf{i}_0) = (\mathsf{R}\mathbf{i}_0)^T (\mathsf{R}\mathsf{C}_0^{-1}\overline{\mathsf{R}})(\mathsf{R}\mathbf{i}_0) + (\mathbf{i}_t - \mathsf{R}\mathbf{i}_0)^T \mathsf{C}_1^{-1} (\mathbf{i}_t - \mathsf{R}\mathbf{i}_0). \tag{72}$$

By a change of variables from $\mathsf{R}\mathbf{i}_0$, this is just the exponent occurring in the usual convolution of two Gaussians, with covariances $\mathsf{R}\mathsf{C}_0\overline{\mathsf{R}}$ and C_1, resulting in a Gaussian with covariance $\mathsf{R}\mathsf{C}_0\overline{\mathsf{R}} + \mathsf{C}_1$.

It is comforting that Gaussians are still natural in this 'small noise' setting, since we may expect them to arise in practice whenever we have a reasonable number of rotations with independent errors (the ensemble will then tend to a Gaussian distribution by the central limit theorem). In the small noise approximation we are considering (less than, say, 30°), the more usual von Mises-Fisher or Fisher-Bingham

distributions of orientation quaternions are virtually equivalent to the Gaussian [7], so that we may extend this practical result to their corresponding parametrizations.

6.3 Optimal Combination of Orientations

Before we tackle rotation combinations, let us recall the procedure for optimally combining translation estimates.

The linear combination of two estimates of a translation is a lerp function. The total covariance of the lerp of Eq. (56) is the 'qerp' (quadratic interpolation, our term):

$$\mathsf{qerp}(\mathsf{C}_0, \mathsf{C}_1; \lambda) \equiv (1-\lambda)^2 \mathsf{C}_0 + \lambda^2 \mathsf{C}_1. \tag{73}$$

Suppose we desire to do an update in a Kalman filter, optimally combining prediction and measurement by weighting them. One of the equivalent formulations of this step, and particularly convenient for us, is that the Kalman filter is the *minimum variance unbiased linear estimate* [13]; for Gaussian distributions this linear estimate is optimal. The updated estimate is then effectively found by minimizing the *total error covariance*, defined as the *trace of the error covariance matrix* (the trace of the matrix equals the divergence of the corresponding linear mapping, so we minimize the 'uncertainty leakage' in this time step). Since the trace is a linear function, the optimization involves differentiation of the weighting factors in Eq. (73). Let us denote the trace of C_i by τ_i, then we find: $\tau_\lambda = \mathsf{qerp}(\tau_0, \tau_1; \lambda) = (1-\lambda)^2\,\tau_0 + \lambda\,\tau_1$, and the value of λ that minimizes this is

$$\lambda_* = \frac{\tau_0}{\tau_0 + \tau_1}, \quad \text{so} \quad \mathbf{x}_* = \frac{\tau_1\,\mathbf{x}_0 + \tau_0\,\mathbf{x}_1}{\tau_0 + \tau_1}, \tag{74}$$

the well-known classical result for the optimal combination of two translation estimates. For comparison with the rotor Kalman update step below, we rewrite this into a form that shows how this estimate deviates from the mean, by reparametrizing using $\tau_* \equiv \tanh(\frac{1}{2}\,\log(\tau_1/\tau_0)) = (\tau_1 - \tau_0)/(\tau_1 + \tau_0)$:

$$\lambda_* = \tfrac{1}{2} - \tfrac{1}{2}\tau_*, \quad \text{so} \quad \mathbf{x}_* = \frac{\mathbf{x}_0 + \mathbf{x}_1}{2} + \tau_* \frac{\mathbf{x}_0 - \mathbf{x}_1}{2}. \tag{75}$$

This is the Kalman filter's update step, as a minimal covariance combination of translations (or other vectorial quantities).

Now let us combine two rotations R_0 and R_1 to an intermediate interpolated rotor R_λ, as before, which has the form of the 'spherical linear interpolation' $R_\lambda = \mathsf{slerp}(R_0, R_1; \lambda)$ of Eq. (58). We have established in Sect. 6.2 that for small amounts of noise, we can work with covariance matrices. The consequences of this linear combination of R_0 and R_1 for the covariance matrices is easily established, and the covariance for the intermediate rotation R_λ is

$$\mathsf{C}_\lambda = \left(\frac{\sin((1-\lambda)\theta/2)}{\sin(\theta/2)}\right)^2 \mathsf{C}_0 + \left(\frac{\sin(\lambda\theta/2)}{\sin(\theta/2)}\right)^2 \mathsf{C}_1 \quad \equiv \mathsf{sqerp}[\mathsf{C}_0, \mathsf{C}_1; \theta/2, \lambda] \tag{76}$$

The notation 'sqerp' is the 'spherical quadratic interpolation' (our term again), and we need the relative rotor angle $\theta/2$ (half the rotation angle) as an argument. We are again interested in the optimal way of combining these distributions, i.e. in a choice of λ which is 'best' in a well-defined sense. And again viewing the Kalman filter update step as the minimum variance unbiased linear estimate helps. For Gaussian distributions this linear estimate is optimal, and as we have seen in Sect. 6.2 there are algebraic and statistical reasons for us to focus on Gaussian distributions of small rotation errors.

As in the translation case, the *total error variance* is the *trace of the error covariance matrix*. Denoting the trace of C_i by τ_i, we have that the interpolation for traces is $\tau_\lambda = \mathsf{sqerp}[\tau_0, \tau_1; \theta/2, \lambda]$. Differentiating, we find the optimum:

$$\begin{aligned}
0 = \frac{d\tau_\lambda}{d\lambda} &= \frac{d}{d\lambda}\left(\left(\frac{\sin((1-\lambda)\theta/2)}{\sin(\theta/2)}\right)^2 \tau_0 + \left(\frac{\sin(\lambda\theta/2)}{\sin(\theta/2)}\right)^2 \tau_1\right) \\
&= \frac{\theta/2}{\sin^2\theta/2}\left(-2\tau_0 \sin((1-\lambda)\theta/2)\cos((1-\lambda)\theta/2) + 2\tau_1 \sin(\lambda\theta/2)\cos(\lambda\theta/2)\right) \\
&= \frac{\theta/2}{\sin^2\theta/2}\left(\tau_0 \sin((\lambda-1)\theta) + \tau_1 \sin(\lambda\theta)\right) \\
&= \frac{\theta/2}{\sin^2(\theta/2)}\left(\sin(\lambda\theta)(\tau_0\cos\theta + \tau_1) - \cos(\lambda\theta)\tau_0 \sin\theta\right)
\end{aligned}$$

Therefore the minimum is reached at:

$$\lambda_* = \frac{1}{\theta}\,\mathrm{atan}\left(\frac{\sin\theta}{\cos\theta + \tau_1/\tau_0}\right) \tag{77}$$

We bring this in a more convenient form using $\tau_* = \tanh(\frac{1}{2}\log(\tau_1/\tau_0)) = (\tau_1 - \tau_0)/(\tau_1 + \tau_0)$ and some ingenious rewriting[5]

$$\lambda_* = \tfrac{1}{2} - \tfrac{1}{2}\frac{\mathrm{atan}\big(\tau_*\,\tan(\theta/2)\big)}{\theta/2} \equiv \tfrac{1}{2} + \tfrac{1}{2}\,\rho_* \tag{78}$$

defining $\rho_* = 2\lambda_* - 1$ in terms of the parameters θ and τ_1/τ_0. With this the interpolation formula gives (using the same rewriting as before for the slerp equation Eq. (59)):

$$R_*(\lambda) = \frac{\cos(\rho_*\theta/4)}{\cos(\theta/4)}\,\frac{R_0 + R_1}{2} + \frac{\sin(\rho_*\theta/4)}{\sin(\theta/4)}\,\frac{R_1 - R_0}{2}. \tag{79}$$

[5] Using $\tan\left(\frac{\theta}{2} - \mathrm{atan}(\frac{\sin(\theta)}{\cos(\theta)+\tau})\right) = \left(\frac{\sin(\theta/2)}{\cos(\theta/2)} - \frac{\sin(\theta)}{\cos(\theta)+\tau}\right)/\left(1 + \frac{\sin(\theta/2)}{\cos(\theta/2)}\,\frac{\sin(\theta)}{\cos(\theta)+\tau}\right) = \frac{\sin(\theta/2)\left(2\cos^2(\theta/2)+\tau-1\right)-2\sin(\theta/2)\cos^2(\theta/2)}{\cos(\theta/2)\left(-2\sin^2(\theta/2)+\tau+1\right)+2\cos(\theta/2)\sin^2(\theta/2)} = \tan(\theta/2)\,\frac{\tau-1}{\tau+1}$.

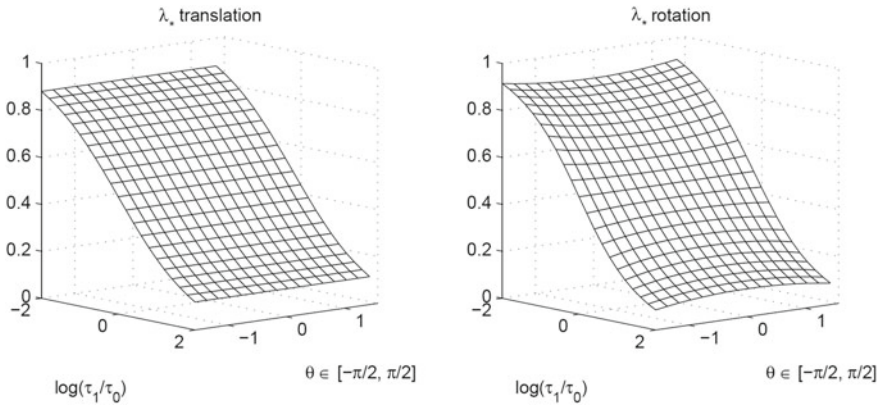


Fig. 7 Minimum total covariance combination of estimates: λ_* as a function of the trace ratio τ_1/τ_0. Left: translation estimates according to Eq. (75). Right: rotation estimates according to Eq. (78) (in the approximation of relatively small noise)

Just to verify our basic understanding, the ratio τ_1/τ_0 has the special cases 0 (no noise in R_1), 1 (equal noise in both) and ∞ (no noise in R_0). These give, as they should, $\lambda_* = 1, \frac{1}{2}$, and 0, so $\rho_* = -1, 0, 1$, corresponding to the rotors $R_*(\lambda) = R_1$, $(R_0 + R_1)$ (normalized) and R_0, respectively. That equal noise case can be generalized to the averaging formula of Eq. (47), which is now indeed seen to be valid for identically distributed data.

Although different in form, Eq. (79) is not that different from Eq. (75) in substance. This is illustrated in the plots of Fig. 7. Indeed, when we are using λ_* in a Kalman filter to optimally blend prediction and observation, we can expect the relative angle θ between them to be small. In that case,

$$\lambda_* \approx \frac{1}{1+\tau_1/\tau_0} = \frac{\tau_0}{\tau_0+\tau_1}, \quad \text{so} \quad R_* \approx \frac{\tau_1 R_0 + \tau_0 R_1}{\tau_0+\tau_1}. \tag{80}$$

This coincides in form with the translation case, since the local tangent space to the unit rotor manifold is flat. But unless there is demand for excessive speed, there is no reason to make this linear approximation, the computation of Eq. (78) and applying the proper interpolation by Eq. (79) is simple enough.

6.4 *Converting Covariances*

Due to the linear relationship between the arc and bivector characterizations in first order, propagation of the statistics of either characterization is straightforward: if the multiplicative **a**-characterization has a covariance of C, then the additive **b**-characterization has a covariance C' of

$$\mathsf{C}' = \overline{\psi}_{\mathbf{B}}\, \mathsf{C}\, \psi_{\mathbf{B}} = \psi_{-\mathbf{B}}\, \mathsf{C}\, \psi_{\mathbf{B}}, \tag{81}$$

with $\psi_{\mathbf{B}}$ as defined in Eq. (37). The Kalman filter update computation of the previous section is now valid for this alternative noise characterization, if we convert the noise covariances C'_0 and C'_1 in the additive bivectors first. This gives

$$\mathsf{C}_0 = \overline{\psi}_0^{-1}\, \mathsf{C}'_0\, \psi_0^{-1} \;\text{ and }\; \mathsf{C}_1 = \overline{\psi}_1^{-1}\, \mathsf{C}'_1 \psi_1^{-1}, \tag{82}$$

where the ψ_i are the conversion functions as in Eq. (37), different for each rotor. The consequence for the relationship between the traces is

$$\mathrm{tr}(\mathsf{C}_0) = \mathrm{tr}\left(\overline{\psi}_0^{-1} \mathsf{C}'_0\, \psi_0^{-1}\right) = \mathrm{tr}\left(\psi_0^{-1} \overline{\psi}_0^{-1}\, \mathsf{C}'_0\right) = \mathrm{tr}(\Psi_0\, \mathsf{C}'_0), \tag{83}$$

with

$$\Psi_0(\mathbf{b}) \equiv \psi_0^{-1}(\overline{\psi}_0^{-1}(\mathbf{b})) = \mathbf{b}_{\parallel} + \mathbf{b}_{\perp} \frac{\mathrm{e}^{-\mathbf{B}}}{\mathrm{sinc}(-\mathbf{B})} \frac{\mathrm{e}^{\mathbf{B}}}{\mathrm{sinc}(\mathbf{B})} = \mathbf{b}_{\parallel} + \mathbf{b}_{\perp}/\mathrm{sinc}^2(\mathbf{B}_0), \tag{84}$$

where $\mathbf{b}_{\parallel}$ and $\mathbf{b}_{\perp}$ are defined relative to $\mathbf{B}_0$. The net effect is to increase the contribution of the perpendicular component $\mathbf{b}_{\perp}$ to the trace τ_0. In 3D, the trace of this bivector-to-bivector mapping $\Psi_0(\,)$ is $1 + 2/\mathrm{sinc}^2(\mathbf{B}_0)$.

A similar equation, but involving $\mathbf{B}_1$, computes τ_1. Substituting these in Eq. (78) gives the optimal λ, which in this additive bivector characterization therefore involves the absolute orientation bivectors $\mathbf{B}_0$ and $\mathbf{B}_1$ of the interpolants R_0 and R_1. As a consequence, *the additive bivector noise characterization is not rotationally invariant*. This makes it rather less likely as a model for reasonable noise in applications than the multiplicative characterization we have used throughout the paper.

6.5 *Procrustes Orientation Estimation, and Extensions to Kalman*

Applying the Kalman update combination procedure clearly presumes knowledge of the covariance of rotation estimates. For the common Procrustes method, which gives an optimal estimate of the orientation to make two labeled point clouds coincide, we have reported this covariance in [5]. Combining those results with this paper provides the backbone for a Kalman filter for the orientational aspects of the Procrustes method for aligning point clouds.

In that paper [5], the covariances in rotation are characterized classically by an additive distribution over the rotation axis vector, rather than a bivector, but the conversion is simple by duality. This is therefore an example of the additive noise (in axis or bivector); in the classical representation of the Procrustes formalism, that

seemed the more natural characterization. We have now found other characterizations that may be preferable, and the corresponding conversions from the earlier solution.

The main conclusions from [5] are that it is the 'harmonized inertia' of the point cloud that becomes the covariance of the rotation. For a point cloud characterized in its principal frame by its major covariances as $\mathrm{diag}(\lambda_1, \lambda_2, \lambda_3)$, the harmonized inertia is computed from this as $\mathrm{diag}(1/(\lambda_2+\lambda_3), 1/(\lambda_3+\lambda_1), 1/(\lambda_1+\lambda_2))$. This implies that isotropic and rod-like shaped point clouds convey most of the intuition on the orientational effects of distributions. For details we refer to the paper; the present paper is the 'Kalman filter on pose estimation' referred to in its Conclusions section.

In a true application to attitude estimation, one usually also tracks the velocity of change and uses that as part of the state. In a quaternion representation, this leads to a nonlinear state propagation by the geometric or quaternion product, and this necessitated [11] to process the data by an unscented Kalman filter. It may be possible to avoid this in the rotor bivector representation.

7 Conclusions

This paper has reviewed three ways of measuring rotation (or orientation, or attitude) differences in 3D: by arc, bivector or chord. These three metrics were codified within the encapsulation framework of Hertzberg et al. [7], which couples a metric uniquely to a choice of parametrization.

By means of geometric algebra, we could perform these parametrizations (and found that the chord metric was a bit surprising in its geometry, see Eq. (34)). Geometric algebra rotors and their bivectors were instrumental in the unification (and introduced in brief tutorial fashion). While in some sense equivalent to quaternions, their full membership of a geometric algebra provides useful extended tools for computation and analysis, as well as visualization.

Having the three metrics thus embedded and related, we investigated averaging, interpolation and optimal updating of rotations.

- We found closed form solutions to the optimal average for arc and chord metrics, though not for the bivector metric. We showed analytically how all are very similar, to second order.
- The arc and bivector based interpolations recover the well-known slerp of [15], while the bivector interpolation connects to known Bézier techniques on quaternions.
- Optimal updating based on minimizing the variance for distributions of rotations was performed to first order, and led to a closed form Kalman update formula. We have shown that the very commonly used 'axis vector perturbation' treatment of noise (use e.g. in [1, 5, 9, 11] and many others), which involves the bivector metric, is not rotationally invariant. It therefore may not correspond to a sensible

noise model, and one should consider modeling perturbations in the arc or chord parametrizations instead.

When we started this work we thought that the chord metric (basically, norm of the difference of rotors—or quaternions) was a naive, non-geometrical way of comparing rotations. Yet our detailed comparison of rotations combinations shows that it is as good as the others for averaging (to second order), and certainly more suited to statistical processing than the common axis vector perturbation. For interpolation, we still prefer the slerp form which the arc or bivector parametrizations provide.

In summary, the arc angle parametrization of rotations appears to be the most suitable for all combinations, with chord parametrization (which is easier to compute) a good second choice.

An additional advantage of using geometric algebra to perform this comparative analysis is that the techniques used are already in a form that permits treatment beyond 3D. Bivectors are the geometric algebra encoding of general Lie algebras [2]. Reformulating the encapsulation framework [7] in terms of bivectors (as we have done here for 3D rotations) should extend its scope as a practical tool for the combination of operators.

References

1. Bourmaud, G., Mégret, R., Arnaudon, M., Giremus, A.: Continuous-discrete extended Kalman filter on matrix Lie groups using concentrated Gaussian distributions. J. Math. Imag. Vis. **51**(1), 209–228 (2015)
2. Doran, C., Hestenes, D., Sommen, F., Van Acker, N.: Lie groups as spin groups. J. Math. Phys. **34**(8), 3642–3669 (1993)
3. Dorst, L., Fontijne, D., Mann, S.: Geometric Algebra for Computer Science: An Object-oriented Approach to Geometry. Morgan Kaufman, San Francisco (2009)
4. Dorst, L., Valkenburg, R.J.: Square root and logarithm of rotors in 3D conformal geometric algebra using polar decomposition. In: Dorst, L., Lasenby, J. (eds.) Guide to Geometric in Practice, pp. 81–104. Springer (2011)
5. Dorst, L.: First order error propagation of the Procrustes method for 3-D attitude estimation. IEEE-PAMI **27**(2), 221–229 (2005)
6. Fontijne, D., Dorst, L.: Reconstructing rotations and rigid body motions from exact point correspondences through reflections. In: Dorst, L., Lasenby, J. (eds.) Guide to Geometric in Practice, pp. 63–78. Springer (2011)
7. Hertzberg, C., Wagner, R., Frese, U., Schröder, L.: Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds. Inf. Fusion **14**(1), 57–77 (2013)
8. Huynh, D.Q.: Metrics for 3D rotations: comparison and analysis. J. Math. Imag. Vis. **35**, 155–164 (2009)
9. Kanatani, K.: Lie ALgebra Method for Pose Optimization Computation, pp. 293–319. Springer, Cham (2020)
10. Kim, M.-J., Kim, M.-S., Shin, S.: General construction scheme for unit quaternion curves with simple high order derivatives. In: Proceedings of the ACM SIGGRAPH Conference on Computer Graphics, pp. 369–376. ACM, January 1995
11. Kraft, E.: A quaternion-based unscented Kalman filter for orientation tracking. In: Proceedings 6th Int. Conf. on Information Fusion, pp. 47–54. ISIF (2003)

12. Lee, J., Shin, S.Y.: General construction of time-domain filters for orientation data. IEEE Trans. Vis. Comput. Graph. **8**, 119–128 (2002)
13. Maybeck, P.S.: The Kalman filter: an introduction to concepts. In: Cox, I.J., Wilfong, G.T. (eds.) Autonomous Robot Vehicles, pp. 194–204. Springer, New York (1990)
14. McRobie, F.A., Lasenby, J.: Simo-Vu Quoc rods using Clifford algebra. Int. J. Num. Methods Eng. **45**, 377–398 (1999)
15. Shoemake, K.: Animating rotation with quaternion curves. Proc. ACM SIGGRAPH **19**(3), 245–254 (1985)

Space-Bending Lattices: Parameterization and Rationalization of Cyclidic Volumes Through Conformal Transformation of Principal Contact Elements

Pablo Colapinto

Abstract Using 3D conformal geometric algebra, smoothly deformable curved volumes of space can be constructed with simple expressions, purely in terms of tangents to the space. Rationalization is achieved through continuous transformation of spheres, which are principal contact surfaces of the space. The resulting triply orthogonal curvilinear coordinate system enables lattice bending deformations within a hexahedral volume. An inverse mapping is used to connect the coordinate systems of neighboring volumes. The development of these parameterizations offers a new but foundational tool for computer aided design of machine parts, animated 3D characters, or freeform architecture.

1 Introduction

The embedding of 3D Euclidean geometry in Conformal Geometric Algebra provides a precise, expressive, and unifying mathematical structure for computer aided design. It is precise in that the discretization of surfaces quickly converges to the continuous case using closed-form equations with spherical primitives. It is expressive in that complicated geometric structures can be intuitively parameterized with just a few elements. Finally, it is unifying in that it helps bridge disparate branches of applied mathematics, and offers its practitioners an intuitive way to generalize expressions into higher dimensions.

As an example of these features, in a previous work [3] we composed conformal transformations of principal contact spheres into a rationalization of *cyclidic surfaces*, which exhibit valuable geometric properties of use in engineering and design. Most notably, these analytic mappings benefit from reduced torsion while their composition from circles offers a closer approximation to continuous differential geometry,

P. Colapinto (✉)
Los Angeles, USA

S. Xambó-Descamps (ed.), *Systems, Patterns and Data Engineering with Geometric Calculi*, SEMA SIMAI Springer Series 13, https://doi.org/10.1007/978-3-030-74486-1_5

which can allow for simplified deformation and fabrication techniques. This reformulation of the model of curvature-line parameterized surfaces and volumes developed by Bobenko and Huhnen-Venedey [2, 12] was motivated by the demonstration by Dorst and Valkenburg [6] and Dorst [7] that all 3D conformal motions can be decomposed into two orthogonal bivector generators of the transformation. The decomposition itself is an application of the bivector split uncovered by Hestenes and Sobczyk [11].

In the present work, we extend this composition to more carefully detail the construction of *cyclidic volumes* of space, by leaning explicitly on the geometric expressivity of principal **tangent** contact elements, obviating the need for supporting circles (or cones) found in our own (and other) earlier formulations. These tangent elements can be considered unit normals at a particular location in space. Dually, they can be considered infinitesimal surface area—a circle that has shrunk to zero radius—though in the expressions that follow we deal with them as an "arrow at a point in space". As we will see, we can create a discretizable volume of curved space by relying on these contact elements exclusively.

By evaluating coordinates within a hexahedral cyclidic volume, we provide a novel technique for constructing deformations, as depicted in Fig. 1. We also provide a novel inverse mapping algorithm, which is necessary for joining these elements together into more complex lattices. The resulting formulations are compact, coherent, and generalizable to higher dimensions.

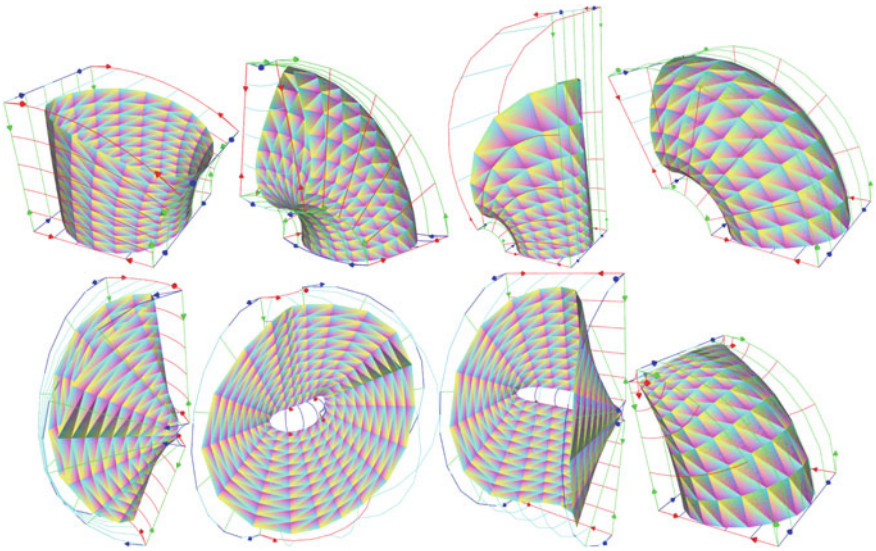


Fig. 1 Various cyclidic lattice deformations of a cylinder. To define the transformation of a vertex $v \mapsto v'$, the coordinates of the original cylinder vertex v are used as inputs to the mapping outlined in this paper.

2 Cyclidic Volumes

A generic six-sided—hexahedral—volume, where the individual edge boundary lines lie on circles and the faces are patches of Dupin cyclides, we call a cyclidic volume (sometimes referred to in the literature as a cyclidic *cube* or hexahedron—see Fig. 2). The cyclidic volume is an extension of a cyclidic surface patch, a 2D manifold so-called because it lies on a Dupin cyclide, which is any torus, cylinder, or cone under a Möbius transformation [9]. The aforementioned references [6] and [7] show that all 3D conformal transformations generate orbits that lie on these cyclidic surfaces. Such surfaces can therefore be rationalized with orthogonal and commuting generators of conformal transformations.

The original use of cyclidic patches as elements in computational geometry stems from Martin's PhD thesis [13], where he developed the use of these principal patches to create a surface modelling technique through direct manipulation of tangent elements. The formulation provides an at-times more intuitive, albeit more restrictive, alternative to the prevalent NURBS-based modelling (Non-Uniform Rationalized Basis Spline), and enables the creation of isotropic surfaces which exhibit some useful properties.

In particular, working with discretized cyclidic surfaces and volumes is useful in engineering and design for their precision and consistency. For one, the edges of a surface patch are circular segments, and, as discussed in [2], discretizing space with curved elements instead of flat ones leads to closer approximations to the continuous case, a desirable feature of any discrete differential system. In [1], freeform architectural constructions are faciliated with simplified edges and reduced nodal complexity, effectively eliminating nodal torsion and reducing construction costs. Conformal surface deformations are also useful in computer graphics, where the elimination of shearing forces will more consistently warp a uv-texture in a way that preserves local features, as explored in [5] and [15]. In [14], we see this type of construction provides a mechanism for solving difficult modelling tasks, such as hole-filling, finding and generating canal surfaces, and shape blending.

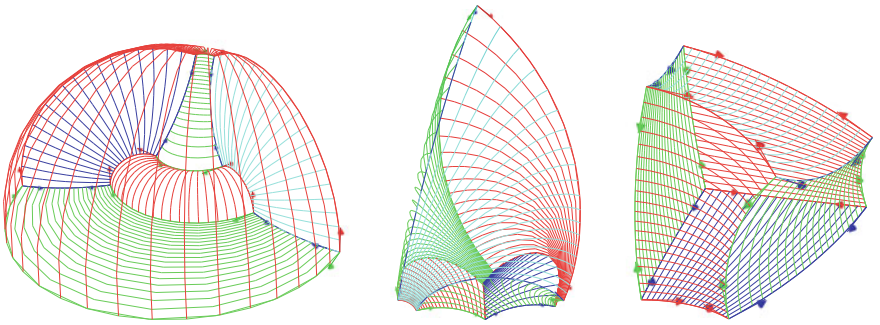


Fig. 2 A few six-sided cyclidic volumes. Each face is itself a cyclidic patch composed of circular segments.

3 Conformal Geometric Algebra

In the references [7, 8] and [4], readers can find a clear introduction to geometric algebra and the conformal model. Here we summarize the notation to be used in the expressions that follow.

The model we use in our parameterization is the geometric algebra of metric space $\mathbb{R}^{4,1}$ with basis elements $\{e_1, e_2, e_3, e_+, e_-\}$, null origin basis vector $n_o = \frac{1}{2}(e_- + e_+)$ and null infinity basis vector $n_\infty = e_- - e_+$. Euclidean vectors in bold $\boldsymbol{x}$ are sums of Euclidean subspace basis $\{e_1, e_2, e_3\}$ and conformal points p are defined through inverse stereographic projection: $p = n_0 + \boldsymbol{x} + \frac{1}{2}\boldsymbol{x}^2 n_\infty$. We write the outer product of null basis vectors as $E = n_o \wedge n_\infty$.

The application of a continuous rotor $R = e^{-tB}$ with scalar variable t and bivector generator B to transform a multivector X is written $X \mapsto R[X] = RX\tilde{R}$.

A ratio of elements $\frac{B}{A}$ is calculated in the order BA^{-1} and in many cases this entire ratio is noted as being normalized with a hat symbol (as in $\widehat{(\frac{B}{A})}$) which in the current text should not be mistaken to signify an involution (to try to avoid confusion, the involution operation is not used anywhere in this paper). Normalization is calculated by dividing out the reverse norm, as in $X_{\text{normalized}} = X/\|X\|$, where the reverse norm is:

$$\|X\| = \begin{cases} \sqrt{X \cdot \tilde{X}} & \text{if } X \cdot \tilde{X} \geq 0 \\ -\sqrt{-(X \cdot \tilde{X})} & \text{if } X \cdot \tilde{X} < 0 \end{cases} \tag{1}$$

Brackets $\langle X \rangle_k$ around a multivector extract only the grade k component of X, with the absence of a subscript, as in $\langle X \rangle$, signifying the grade 0 scalar component.

In this 5D metric space, the dual operator X^* effectively multiplies X by the negative of the pseudoscalar $I = e_{123}E$, whereas the undual operator X^{-*} does the same but without the minus sign, hence:

$$X^* = -X^{-*} = -XI \tag{2}$$

4 A Tangential Approach

A summary of the details below can be found in the "cast of characters" reference of Table 1.

A **translator** $T_{\boldsymbol{p}}$ in direction $\boldsymbol{p}$ is a rotor constructed through exponentiation of a direction vector $\boldsymbol{p}n_\infty$ where $\boldsymbol{p}$ is a Euclidean vector:

$$T_{\boldsymbol{p}} = e^{-\frac{\boldsymbol{p}n_\infty}{2}} = 1 - \frac{\boldsymbol{p}n_\infty}{2}, \tag{3}$$

which stems from a result of Taylor expansion (see also Eq. 23 below).

A **tangent** τ at a point p is constructed from a unit Euclidean vector $\boldsymbol{v}$ representing a direction, wedged with the null origin n_o, which together have been translated by a translator $T_{\boldsymbol{p}}$ to position $p = n_o + \boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^2 n_\infty$:

$$\tau = T_{\boldsymbol{p}}[n_0 \boldsymbol{v}] \tag{4}$$

Such a tangent element, or "arrow-at-a-point" is sometimes called a **null point pair**, as it is equivalent to a 0-sphere with zero radius, and it is dual to a **null circle**, or "surface-at-a-point", which is a 1-sphere with zero radius.

A tangent at p is orthogonal (normal) to a flat surface **dual plane** π through p constructed with the simple form:

$$\pi = n_\infty \cdot \tau \tag{5}$$

The flat surface π (with normal τ at p) is bent into a curved surface **sphere** (also with normal τ at p) with scalar curvature k by using the tangent itself as a **generator** to create a curvature inducing rotor S:

$$\sigma = S[\pi] \tag{6}$$

with

$$S = e^{-\frac{k\tau}{2}} = 1 - \frac{k\tau}{2} \tag{7}$$

which is an expression resulting from the Taylor expansions of Eq. 23 below, given that $\tau^2 = 0$. Here k really is the **curvature**: i.e. the inverse of the radius of the resulting sphere. It is important to note that the expressivity of the tangent allows us to use it both to construct the flat surface and to generate the transformation that bends it. All that is needed is the tangent τ and the curvature scalar k. The expression for creating a surface sphere σ with normal τ and curvature k in precisely just those terms is then:

$$\begin{aligned}
\sigma(\tau, k) &= (1 - \frac{k}{2}\tau)(n_\infty \cdot \tau)(1 + \frac{k}{2}\tau) \\
&= ((n_\infty \cdot \tau) - (\frac{k}{2}\tau)(n_\infty \cdot \tau))(1 + \frac{k}{2}\tau) \\
&= ((n_\infty \cdot \tau) + (n_\infty \cdot \tau)(\frac{k}{2}\tau))(1 + \frac{k}{2}\tau) \\
&= (n_\infty \cdot \tau)(1 + \frac{k}{2}\tau)(1 + \frac{k}{2}\tau) \\
&= (n_\infty \cdot \tau)(1 + k\tau)
\end{aligned} \tag{8}$$

Another simple and useful expression for constructing a curved surface σ with a normal τ is to use another surface $\sigma_\perp$ as an orthogonality constraint to give

$$\sigma = \sigma_\perp \cdot \tau \tag{9}$$

which constructs a sphere σ that is orthogonal to $\sigma_\perp$ and has normal τ. When two spheres are orthogonal, at every point along their intersection, the normals of one lie tangent to the other. Equation 9 is a versatile general expression: when $\sigma_\perp$ is a point p, the resulting sphere goes through p. And we have already seen in Eq. 5 that when $\sigma_\perp$ is the point at infinity, the resulting sphere is a plane. We will also encounter this basic expression in Sect. 9 in the form $p \cdot \kappa$, where κ is a pair of contact surfaces, and p is a point we would like to inverse map.

Finally, the following expression for extracting the tangent τ normal at a point p on a surface σ will be useful during cyclidic constructions.

$$\tau = \sigma \wedge p \tag{10}$$

The use of the above spheres in rationalization is further described in Sect. 8 and Table 1.

5 The Sense of Spheres

The curvature-parameterized spheres of Eq. 8 will have a positive or negative signed **sense** depending on whether the curvature coefficient is, respectively, positive or negative (see Fig. 3). This curvature will be initially encoded in the n_o coefficient of the sphere. In practice, we normalize the spheres by dividing out the absolute value of this coefficient, so that spheres have an orientation of 1 or -1. We can also define this sense s algebraically as the trivector direction $e_{123}n_\infty$,

$$s = (-n_\infty \cdot \sigma^{-*}) \wedge n_\infty \tag{11}$$

which is a formulation found in [8].

As shown in Fig. 3, in our curvature parameterization a positive sense means the generating tangent points inwards towards the center of the sphere, and hence, towards the direction of curvature, concave upward. With a negative sense the tangent points outwards, the curvature being concave downward. In this representation, the dot product of a point p with a sphere σ is positive if a point is on the same side of the sphere surface pointed to by the tangent's direction and negative if it is on the other side. The same separation holds true if the curvature is 0 and σ is in fact a plane.

While this is a consistent representation, it is "inverted" in that positively-sensed spheres have normals that face inwards, and points that are on the same side of a sphere as the direction of its normals have a positive dot product with it. Furthermore, the spheres of Eq. 8 will have the opposite sense than the spheres they are orthogonal to. We can easily change this if desired by negating the expressions of Eqs. 6 and 9,

$$\sigma = S[-n_\infty \cdot \tau] \tag{12}$$

$$\sigma = -\sigma_\perp \cdot \tau \tag{13}$$

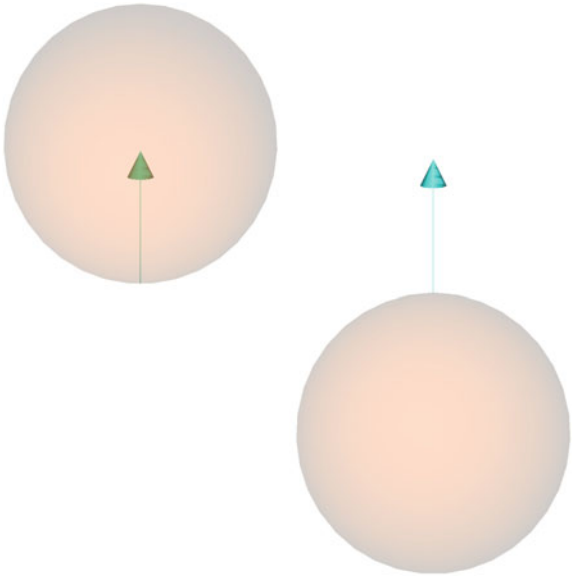


Fig. 3 The sense (sometimes called direction, or orientation) of spheres in the curvature parameterization of Eq. 8 is (left image) positive if the curvature is positive, concave upward, and (right image) negative if the curvature is negative, concave downward. Both spheres in the image above, despite having opposing absolute sense, actually have similar *relative sense* (positive) with respect to their generating tangents. Table 2 demonstrates how relative sense can be used to specify an appropriate transformation between two intersecting spheres.

but we need not do so, as it will not impact the formulations in this paper.

In practice, what is more relevant than the absolute sense of a sphere σ is the sense relative to a given tangent arrow τ orthogonal to its surface, which we define as the sense of the sphere $\sigma \cdot \tau$. We call this important measure the **relative sense** as it is the sense with respect to a specific direction, telling us whether the arrow points towards the positive or negative side of the sphere. Similarly, we use the signed scalar value $p \cdot \sigma$ to determine the relative direction of a point p with respect to the sphere σ, which tells us on which side of the surface the point lies. In Sect. 8, the sign of these measures is essential to enabling us to precisely control the bivector logarithm of the transformation that takes a tangent arrow on one surface to a tangent arrow on the other (see Table 2).

6 Establishing the Tangent Frame

With our basic geometric constructions prescribed, let us be more explicit about what these represent. We will be constructing a triply orthogonal curved coordinate system, providing a **mapping** $f : (u_x, v_y, w_z) \mapsto \mathbb{R}^3$ from scalar coordinate variables u_x, v_y, w_z to a point $p_{(u_x, v_y, w_z)}$ in a 3D volume. The value of the coordinate we write in the subscript, such that u_0 means $u = 0$. Working in the conformal model of 3D Euclidean space, an orthonormal frame at coordinate (u_x, v_y, w_z) is written $F_{(u_x, v_y, w_z)}$. This **tangent frame** is encoded by the location and orientation of three orthonormal tangents at p: τ_u, τ_v, τ_w. In generic notation we will write τ_i,

with $i = u, v$ or w specifying our principal directions. Our initial volume will be **hexahedral** (six-sided), like a cube, and so there are eight frames in all $F_{(u_x, v_y, w_z)}$ with $x, y, z \in \{1, 0\}$, as depicted in Fig. 6.

Through use of the simple expressions of Eqs. 6 and 9, we will identify spheres σ^i that have normals τ_i which represent a **constant coordinate surface**: moving on the surface σ^i is equivalent to *not* moving in the orthogonal direction encoded by τ_i. Again, the superscript notation in σ^i specifies the coordinate that is held constant, with $i = u, v$ or w. Likewise, the subscript in τ_i indicates the coordinate value that is increased in direction τ_i.

With these tangent elements, six spheres written σ^i_j (with normals τ_i) are constructed from, respectively, six curvature coefficients k^i_j. These represent coordinate surfaces of constant i with respect to moving in the j direction. So, for instance, the sphere σ^u_v represents the surface of constant coordinate u as we move in the v direction, whereas the sphere σ^u_w represents the surface of constant coordinate u as we move in the w direction. In flat spaces these two surfaces would be the same plane, but this is not necessarily the case in curvilinear coordinates, where there are now two distinct constant coordinate surfaces for each direction of travel: σ^u_v, σ^w_v, σ^u_w, σ^v_w, σ^v_u, and σ^w_u.

It becomes geometrically clear that each tangent τ lies along a circular **coordinate curve** K specifying a linearly independent direction of movement, and that each circular coordinate curve trajectory K lies on the intersection of the two spherical constant coordinate surfaces (see Fig. 4).[1]

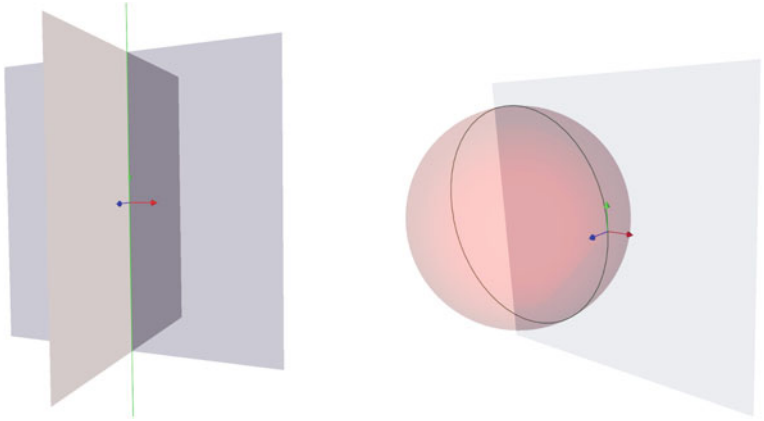


Fig. 4 Two examples where a surface of constant u in the v direction, σ^u_v, intersects with a surface of constant w in the v direction, σ^w_v, to define a coordinate curve K_v. On the left, both surfaces are flat and the curve is straight. On the right, σ^u_v is round and the curve is as well.

[1] *Dupin's theorem* relates that the principal curvature lines lie at the intersection of a pair of coordinate surfaces in a triply orthogonal system. Hestenes provides a Geometric Calculus approach to analyzing the various components of these differential manifolds in [10].

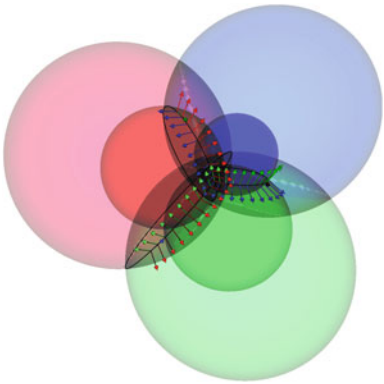


Fig. 5 Six spheres define all the various constant coordinate surfaces of one triply orthogonal tangent frame. In red: σ_v^u and σ_w^u. In green: σ_u^v and σ_w^v. In blue, σ_u^w and σ_v^w. Coordinate curves lie upon the intersection of spheres with common subscripts, e.g. $(\sigma_v^u \wedge \sigma_v^w)^{-*}$, $(\sigma_u^v \wedge \sigma_u^w)^{-*}$, and $(\sigma_w^u \wedge \sigma_w^v)^{-*}$. A transformation along that curve is then defined using Eq. 17.

$$K_k = (\sigma_k^i \wedge \sigma_k^j)^{-*} \tag{14}$$

To specify points along this coordinate curve we apply a conformal rotor to a point translated some distance along the line $\tau^i n_\infty$. Given a point along this line we transform it to lie on the coordinate curve using the rotor C_k, which is constructed by summing the two generators of constant surfaces i and j:

$$C_k = e^{-\frac{1}{2}(k_k^i \tau_i + k_k^j \tau_j)} = 1 - \frac{1}{2}(k_k^i \tau_i + k_k^j \tau_j) \tag{15}$$

The selection of the point along the line is also defined as a rotor in terms of a scalar distance d_k and, once again, our tangent element itself:

$$T_{d_k} = e^{-\frac{d}{2}(n_\infty \cdot \tau_k) \wedge n_\infty} \tag{16}$$

such that our point p' on curve K_k is completely defined in terms of tangents as a translation followed by a conformal rotation:

$$p' = C_k T_{d_k}[p] \tag{17}$$

where p is the location of the orthonormal frame F (see Fig. 5).

7 Completing the Coordinate System

So far we have fully defined the six constant coordinate surfaces of one "corner" of our volume of space, using a tangent frame and six curvature coefficients. To actually navigate the space we need to be able to specify the tangent frames at the other seven corners.

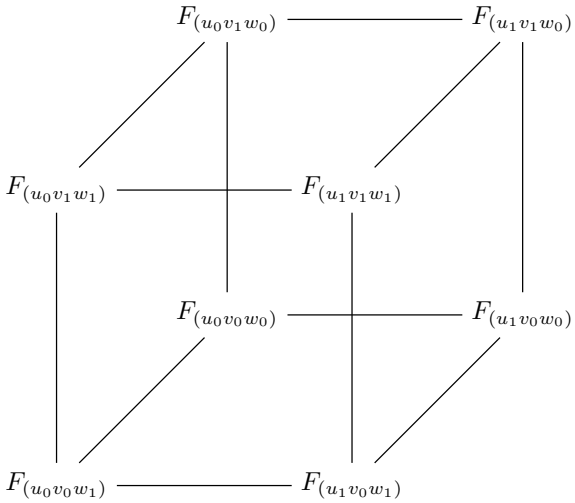


Fig. 6 A hexahedral (six-sided) lattice consists of eight tangent frames and can be completely specified with nine curvature coefficients, forming 24 constant coordinate surfaces (see Sect. 7).

Before we do so, it is already apparent we will need to keep track of more variables in our notation, as the constant coordinate surfaces at one corner of our hexahedral volume need to be distinguishable from those at the other. We extend the previous notation with the following scheme: $\sigma_{j_m}^{i_n k_l}$ is to be understood as the surface of constant $i = n$ as we move in the j direction, starting at frame $F_{(i_n, j_m, k_l)}$, where $i = n$, $j = m$ and $k = l$. Thus, this encoding specifies the actual frame where our surface exists. If a subscript or superscript is missing, then it should be assumed to be equal to 0.

For instance, $\sigma_{v_0}^{u_0 w_0}$ is the surface of constant u as we move along v, starting at frame $F_{(u_0, v_0, w_0)}$, whereas $\sigma_{v_0}^{u_1 w_0}$ is the surface of constant u as we move along v, starting at frame $F_{(u_1, v_0, w_0)}$.

With the dimensions of the volume given, to fully define a hexahedral volume we will need to create a total of 24 constant coordinate surfaces (two for each edge of the volume):

$$\begin{aligned}
&\sigma_{v_0}^{u_0 w_0}, \sigma_{w_0}^{u_0 v_0}, \sigma_{u_0}^{v_0 w_0}, \sigma_{w_0}^{v_0 u_0}, \sigma_{u_0}^{w_0 v_0}, \sigma_{v_0}^{w_0 u_0},\\
&\sigma_{v_0}^{u_1 w_0}, \sigma_{w_0}^{u_1 v_0}, \sigma_{u_0}^{v_1 w_0}, \sigma_{w_0}^{v_1 u_0}, \sigma_{u_0}^{w_1 v_0}, \sigma_{v_0}^{w_1 u_0},\\
&\sigma_{v_0}^{u_0 w_1}, \sigma_{w_0}^{u_0 v_1}, \sigma_{u_0}^{v_0 w_1}, \sigma_{w_0}^{v_0 u_1}, \sigma_{u_0}^{w_0 v_1}, \sigma_{v_0}^{w_0 u_1},\\
&\sigma_{v_0}^{u_1 w_1}, \sigma_{w_0}^{u_1 v_1}, \sigma_{u_0}^{v_1 w_1}, \sigma_{w_0}^{v_1 u_1}, \sigma_{u_0}^{w_1 v_1}, \sigma_{v_0}^{w_1 u_1}.
\end{aligned}$$

Similarly, we will also want to keep better track of our tangent frames, as there will be eight total, one for each corner of the hexahedral volume. We refer to the originating frame tangents as before, τ_u, τ_v, and τ_w, and specify other frames with specific values: e.g. $\tau_u^{(u_1, v_0, w_0)}$ is the u-direction tangent element at frame $F_{(u_1, v_0, w_0)}$, and $\tau_u^{(u_1, v_1, w_0)}$ the u-direction tangent at frame $F_{(u_1, v_1, w_0)}$.

7.1 Choosing Three More Curvature Coefficients to Close the Volume

By specifying three distinct distances, d_u, d_v, and d_w (one for each principal direction) and applying the resulting rotor from Eq. 17 to the tangents τ_u, τ_v, and τ_w, we can transform our original tangent frame $F_{(u_0,v_0,w_0)}$ to the corners $F_{(u_1,v_0,w_0)}$, $F_{(u_0,v_1,w_0)}$, and $F_{(u_0,v_0,w_1)}$ as labelled in Fig. 6.

Our task is now to identify the tangent frames and surface spheres at the remaining unknown corners, $F_{(u_1,v_1,w_0)}$, $F_{(u_0,v_1,w_1)}$, $F_{(u_1,v_0,w_1)}$ and $F_{(u_1,v_1,w_1)}$. To do so we first specify three more curvature coefficients at our newly found frames to identify three more surfaces: $\sigma_{u_0}^{v_1w_0}$, $\sigma_{w_0}^{u_1v_0}$, and $\sigma_{v_0}^{w_1u_0}$, as depicted in Fig. 7. We then use Eq. 9 to create the surfaces that are orthogonal to them, namely:

sphere	is orthogonal to	has normal
$\sigma_{v_0}^{w_0u_1}$	$\sigma_{u_0}^{v_1w_0}$	$\tau_w^{(u_1,v_0,w_0)}$
$\sigma_{v_0}^{u_1w_0}$		$\tau_u^{(u_1,v_0,w_0)}$
$\sigma_{u_0}^{v_0w_1}$	$\sigma_{w_0}^{u_1v_0}$	$\tau_v^{(u_0,v_0,w_1)}$
$\sigma_{u_0}^{w_1v_0}$		$\tau_w^{(u_0,v_0,w_1)}$
$\sigma_{w_0}^{u_0v_1}$	$\sigma_{v_0}^{w_1u_0}$	$\tau_u^{(u_0,v_1,w_0)}$
$\sigma_{w_0}^{v_1u_0}$		$\tau_v^{(u_0,v_1,w_0)}$

With these constructed, we use them to continue constructing three more:

sphere	is orthogonal to	has normal
$\sigma_{u_0}^{w_0v_1}$	$\sigma_{v_0}^{w_0u_1}$	$\tau_v^{(u_0,v_1,w_0)}$
$\sigma_{w_0}^{v_0u_1}$	$\sigma_{u_0}^{v_0w_1}$	$\tau_u^{(u_1,v_0,w_0)}$
$\sigma_{v_0}^{u_0w_1}$	$\sigma_{w_0}^{u_0v_1}$	$\tau_w^{(u_0,v_0,w_1)}$

Finally, in the following section, we will calculate the transformation that takes one surface to its opposing one, and apply this transformation to the tangents themselves.

8 Conformal Rotors from Ratios of Constant Coordinate Surfaces

With 18 of the orthogonal surfaces now constructed, we are able to create conformal transformations that take spheres to spheres. This transformation allows us to build three new tangent frames $F_{(u_1,v_1,w_0)}$, $F_{(u_0,v_1,w_1)}$, and $F_{(u_1,v_0,w_1)}$. For instance, $F_{(u_1,v_1,w_0)}$ can be found by transforming the tangents at $F_{(u_1,v_0,w_0)}$ by the conformal rotor that takes constant coordinate surface $\sigma_{u_0}^{v_0w_0}$ to $\sigma_{u_0}^{v_1w_0}$. The action of this rotor is depicted in Fig. 8. We will be using these tangent frames to produce additional surface spheres using Eq. 9.

We write the point pair generator of a conformal surface transformation as

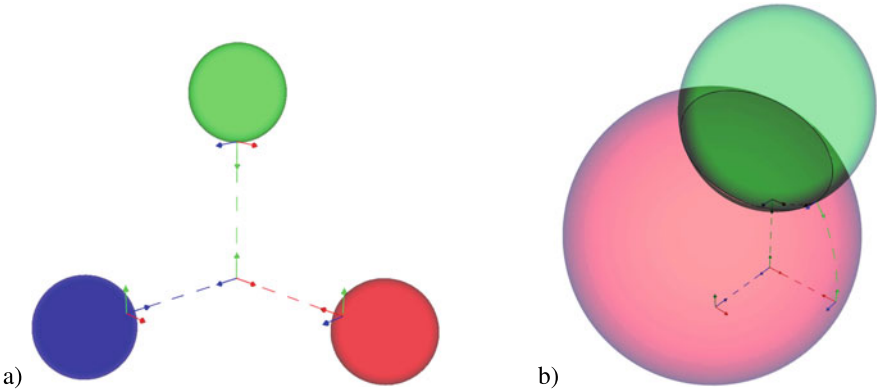


Fig. 7 a) Surfaces $\sigma_{u_0}^{v_1 w_0}$ (green), $\sigma_{w_0}^{u_1 v_0}$ (red), and $\sigma_{v_0}^{w_1 u_0}$ (blue) are created by applying Eq. 6 to the tangents $\tau_v^{(u_0,v_1,w_0)}$, $\tau_u^{(u_1,v_0,w_0)}$, and $\tau_w^{(u_0,v_0,w_1)}$, respectively. b) Given $\sigma_{u_0}^{v_1 w_0}$ (green), we define $\sigma_{v_0}^{u_1 w_0}$ as the sphere that must be orthogonal to it, with normal $\tau_u^{(u_1,v_0,w_0)}$, using Eq. 9. While not pictured, we also define $\sigma_{v_0}^{w_0 u_1}$ in this way, with normal $\tau_w^{(u_1,v_0,w_0)}$.

Table 1 Cast of characters: proposed algebraic notation for curvilinear coordinate systems using conformal geometric algebra, built entirely with tangent elements. The hat symbol $\widehat{\ }$ here means "normalized" (and not involution).

Notation	Expression(s)	Geometric meaning
τ_i	$T[n_0 \boldsymbol{v}_i]$	Unit normal to constant surface π_j^i
π_j^i	$n_\infty \cdot \tau_i$	Surface (flat) of constant $i = 0$ in direction j
S_j^i	$e^{-\frac{1}{2}k_j^i \tau_i}$	Rotor to create surface with scalar curvature k_j^i
σ_j^i		Surface of constant $i = 0$ in direction j
	$S_j^i[\pi_j^i]$	Round surface from transformed flat surface
$\sigma_j^{i_n}$		Surface of constant $i = n$ in direction j
$\sigma_{j_m}^{i_n k_l}$		Surface of constant $i = n$ in direction j, at $j = m, k = l$
$\sigma_{i_{n-1}}^{j_{m+1} k_l} \cdot \tau_i^{(i_n,j_m,k_l)}$ $\sigma_{k_{l-1}}^{j_{m+1} i_n} \cdot \tau_i^{(i_n,j_m,k_l)}$		Surface with normal τ orthogonal to surface σ
$\kappa_j^{i_n}$	$\frac{1}{2}\log(\pm\frac{\widehat{\sigma_j^{i_{n+1}}}}{\sigma_j^{i_n}})$	Generator to transform $\sigma_j^{i_n}$ to $\sigma_j^{i_{n+1}}$
$R_j^{i_{(n+t)}}$	$e^{t\kappa_j^{i_n}}$	Rotor transforming $\sigma_j^{i_n}$ to $\sigma_j^{i_{n+t}}$, with t in range (0, 1)
$R_{j_m}^{i_n k_l}$		Rotor transforming $\sigma_{j_m}^{i_n k_l}$ to $\sigma_{j_m}^{i_{n+t} k_l}$ and $\sigma_{j_m}^{k_l i_n}$ to $\sigma_{j_m}^{k_l i_{n+1}}$
C_k	$e^{-\frac{1}{2}(k_k^i \tau_i + k_k^j \tau_j)}$	Rotor that bends a straight line into a k-direction curve
K_k	$(\sigma_k^i \wedge \sigma_k^j)^{-*}$	A k-direction curve

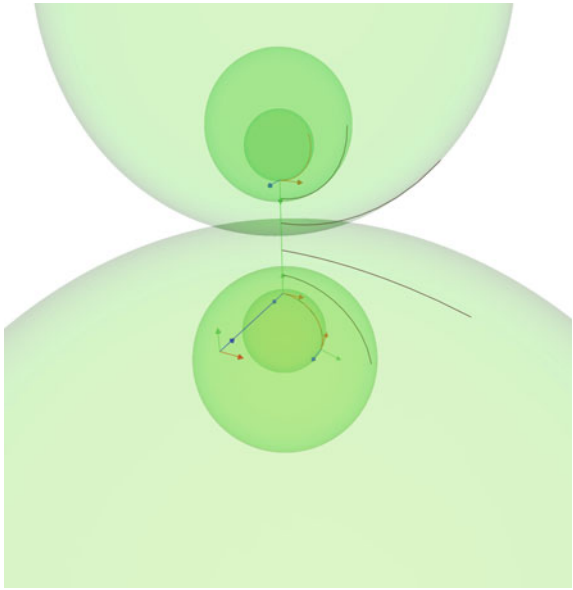


Fig. 8 The conformal rotor $R_{u_0}^{v_0 w_0}$ is a continuous transformation that takes the surface $\sigma_{u_0}^{v_0 w_0}$ to $\sigma_{u_0}^{v_1 w_0}$. It can be applied to tangents, points, etc.—any element of the algebra. Applying it to tangents allows us to identify the corners of our hexahedral lattice.

$$\kappa_j^{i_n} = \frac{1}{2}\log(\pm\frac{\widehat{\sigma_j^{i_{n+1}}}}{\sigma_j^{i_n}}) \tag{18}$$

with the hat here symbolizing normalization of the entire ratio (not involution). In practice, we use the minus sign if the relative sense of the spheres with respect to their tangents at the beginning and end of the transformation are opposite (see the bottom row of Table 2 for visual explanation of this distinction).

As explained in [6], the log itself is

$$\log(R) = \text{atanh2}(\langle R\rangle_2, \langle R\rangle) \tag{19}$$

with

$$\text{atanh2}(s, c) = \begin{cases} \frac{\text{asinh}(\sqrt{s^2})}{\sqrt{s^2}} s & \text{if } s^2 > 0 \\ s & \text{if } s^2 = 0 \\ \frac{\text{atan2}(\sqrt{-s^2}, c)}{\sqrt{-s^2}} s & \text{if } -1 \leq s^2 < 0 \end{cases} \tag{20}$$

In [3] we proposed an additional case to this logarithm: when s^2 is negative, we can provide an alternate sense of transformation to approach a sphere from the opposite direction (see the first column of Table 2 for when we want to do so):

$$\frac{\text{atan2}(\sqrt{-s^2}, c) - 2\pi}{\sqrt{-s^2}} s \tag{21}$$

The rotor itself is then the continuous transformation

$$R_j^{i_{(n+t)}} = e^{t\kappa_j^{i_n}} \tag{22}$$

with n an integer greater than 0 and t a scalar value in the range of 0 to 1. The laws of exponentiation follow from Taylor expansion:

$$e^{tB} = \begin{cases} \cosh(t) + \sinh(t)B & \text{if } B^2 = 1 \\ \cos(t) + \sin(t)B & \text{if } B^2 = -1 \\ 1 + tB & \text{if } B^2 = 0. \end{cases} \tag{23}$$

Applying the rotors to the tangents leaves out only $F_{(u_1,v_1,w_1)}$, the position of which can be found as the intersection of three planes, and the tangents themselves using Eq. 10.

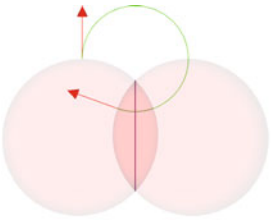
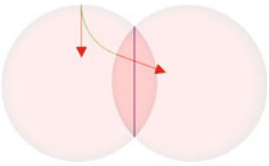
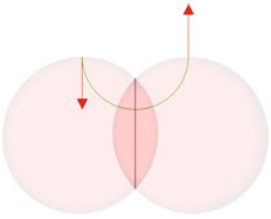
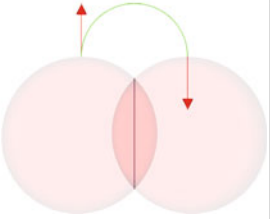
With these tangent frames constructed, we can find the 6 remaining constant coordinate surfaces.

sphere	through point	has normal
$\sigma_{w_0}^{u_1 v_1}$	$p_{(u_1,v_1,w_1)}$	$\tau_u^{(u_1,v_1,w_0)}$
$\sigma_{w_0}^{v_1 u_1}$		$\tau_v^{(u_1,v_1,w_0)}$
$\sigma_{u_0}^{v_1 w_1}$		$\tau_v^{(u_0,v_1,w_1)}$
$\sigma_{u_0}^{w_1 v_1}$		$\tau_w^{(u_0,v_1,w_1)}$
$\sigma_{v_0}^{w_1 u_1}$		$\tau_w^{(u_1,v_0,w_1)}$
$\sigma_{v_0}^{u_1 w_1}$		$\tau_u^{(u_1,v_0,w_1)}$

8.1 Triply Orthogonal Coordinate System

The discretization of this curved space depends only on the constant coordinate surface spheres defined in the previous section. This compact representation allows one to continue working directly with elements tangent to the space itself, without extraneously constructed supporting elements. With our surfaces specified, we can use Eq. 18 to define the generators $\kappa_{j_m}^{i_n k_l}$ of the conformal transformation $R_{j_m}^{i_n k_l}$ that takes $\sigma_{j_m}^{i_n k_l}$ to $\sigma_{j_m}^{i_{n+1} k_l}$. There are 12 such generators, which are used to transform one constant coordinate surface into another on the opposing face. There are two generators across each face of the hexahedral volume—i.e. four generators for each of the three directions a face can be extended to form a cube. Figure 9 diagrams four such generators.

Table 2 Given two intersecting spheres σ_L and σ_R, there are four ways to generate a transformation that takes a point on one sphere to a point on the other. Two boolean parameters determine which path should be made – rows of the table consider whether starting and ending tangents of the path have similar relative sense with respect to their spheres, and the columns consider whether the starting point has a similar sense as the ending tangent.

Sign of relative senses are ...	Equation	Starting point and ending tangent are ... similar	Starting point and ending tangent are ... opposite
		$\log = \frac{\text{atan2}(\sqrt{-s^2}, c) - 2\pi}{\sqrt{-s^2}} s$	$\log = \frac{\text{atan2}(\sqrt{-s^2}, c)}{\sqrt{-s^2}} s$
... similar	$\kappa = \frac{1}{2}\log(\widehat{\frac{\sigma_R}{\sigma_L}})$		
... opposite	$\kappa = \frac{1}{2}\log(-\widehat{\frac{\sigma_R}{\sigma_L}})$		

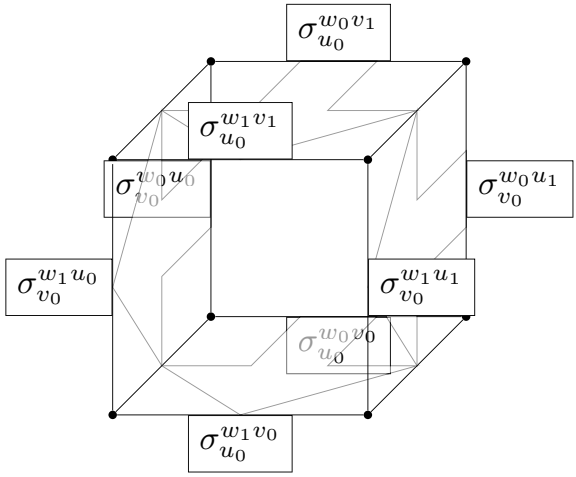


Fig. 9 The arrows on four faces of the hexahedral volume here represent the four transformation generators κ^{w_0} bringing constant surfaces σ^{w_0} to σ^{w_1}. These w-direction transformations are just 4 of the 12 possible generators $\kappa^{i_n k_l}_{j_m}$ formed by the ratio of constant coordinate surface spheres using Eq. 18. The other 8 consist of v-direction transformations taking σ^{v_0} to σ^{v_1} and u-direction transformations taking σ^{u_0} to σ^{u_1}.

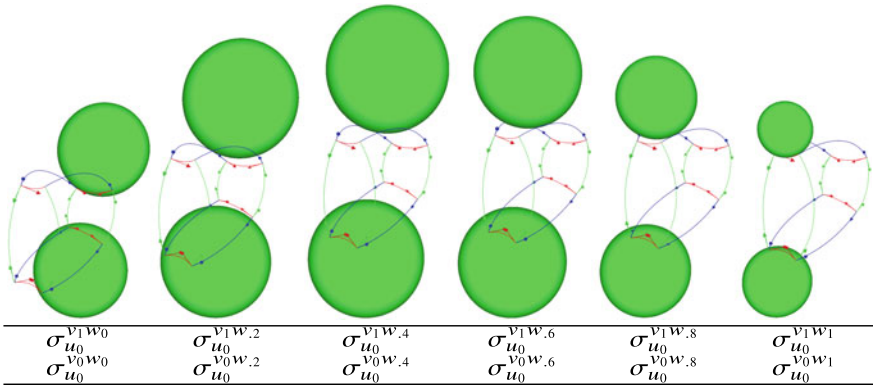


Fig. 10 The principal surface spheres $\sigma_{u_0}^{v_0 w_0}$ and $\sigma_{u_0}^{v_1 w_0}$ slide along the volume in the w-direction to become $\sigma_{u_0}^{v_0 w_1}$ and $\sigma_{u_0}^{v_1 w_1}$ by application of the $R_{u_0}^{w_z v_0}$ and $R_{u_0}^{w_z v_1}$ rotors, respectively, with z in the range $(0, 1)$.

We have explained that the rotor $R_{v_0}^{w_0 u_0}$ takes sphere $\sigma_{v_0}^{w_0 u_0}$ to $\sigma_{v_0}^{w_1 u_0}$ and that the rotor $R_{v_0}^{w_0 u_1}$ takes sphere $\sigma_{v_0}^{w_0 u_1}$ to $\sigma_{v_0}^{w_1 u_1}$. But these rotors also, respectively, transform the spheres $\sigma_{v_0}^{u_0 w_0}$ and $\sigma_{v_0}^{u_1 w_0}$ into the spheres $\sigma_{v_0}^{u_0 w_1}$ and $\sigma_{v_0}^{u_1 w_1}$, effectively sliding them along the outside of the volume. In general the $R_{j_m}^{i_n k_l}$ rotor transforms $\sigma_{j_m}^{i_n k_l}$ to $\sigma_{j_m}^{i_{n+1} k_l}$ and also transforms $\sigma_{j_m}^{k_l i_n}$ to $\sigma_{j_m}^{k_l i_{n+1}}$. Figure 10 shows these principal surfaces sliding along the w-direction.

We desire a consistent mapping that allows to move first along the w coordinate curve by an amount z, and then find our surfaces of constant $\sigma_{v_0}^{u_0 w_z}$, $\sigma_{v_0}^{u_1 w_z}$, $\sigma_{u_0}^{v_0 w_z}$, and $\sigma_{u_0}^{v_1 w_z}$ at that location, to effectively determine the uv patch at $w = z$. While Fig. 10 shows the principal contact elements can slide along the tangent space of the volume, there is no guarantee that such contact surfaces will slide along opposing surfaces at a rate that maintains the orthogonality condition. Instead, we rely on the orthogonality condition to find these surfaces constructively. We first transform the tangents τ_u and τ_v themselves by $R_{v_0}^{w_z u_0}$ (so, by amount z along the w coordinate curve) to define $\tau_u^{(u_0 v_0 w_z)}$ and $\tau_v^{(u_0 v_0 w_z)}$, and then across the full volume in the u and v directions:

$$\tau_u^{(u_1 v_0 w_z)} = R_{w_0}^{u_0 v_0}[\tau_u^{(u_0 v_0 w_z)}]$$

$$\tau_v^{(u_0 v_1 w_z)} = R_{w_0}^{v_0 u_0}[\tau_v^{(u_0 v_0 w_z)}]$$

With these four tangents defined at $w = z$ we can construct the constant coordinate spheres at $w = z$ by maintaining orthogonality with our known constant coordinate spheres:

$$\sigma_{v_0}^{u_0 w_z} = \sigma_{w_0}^{v_1 u_0} \cdot \tau_u^{(u_0 v_0 w_z)}$$

$$\sigma_{v_0}^{u_1 w_z} = \sigma_{w_0}^{v_1 u_1} \cdot \tau_u^{(u_1 v_0 w_z)}$$

$$\sigma_{u_0}^{v_0 w_z} = \sigma_{w_0}^{u_1 v_0} \cdot \tau_v^{(u_0 v_0 w_z)}$$

$$\sigma_{u_0}^{v_1 w_z} = \sigma_{w_0}^{u_1 v_1} \cdot \tau_v^{(u_0 v_1 w_z)}$$

We can then define the transformation between these spheres using Eq. 18 to find the generators $\kappa_{v_0}^{u_0 w_z}$ and $\kappa_{u_0}^{v_0 w_z}$. Interpolating these generators by amounts x and y respectively, we now have a complete mapping at (u_x, v_y, w_z):

$$R_{(u_x, v_y, w_z)} = R_{u_0}^{v_y w_z} R_{v_0}^{u_x w_z} R_{v_0}^{w_z u_0} \tag{24}$$

The above conformal rotor first performs a transformation of amount z along the w-direction generator, and then of amount x along the w-transformed u-direction generators and amount y along the w-transformed v-direction generators. Thus, here we are extending the uv surface along the w-direction (see Fig. 11)—we should note that a different transformation, and therefore mapping, would result if we instead extended the uw face along the v-direction or the vw face along the u-direction. The application of such a mapping to a cylinder is demonstrated in Figs. 12 and 13.

9 Connecting Volumes with Inverse Mapping

A challenge remains to connect the coordinate systems of two discretizable volumes or surface patches. This is a necessary step with the conformal discretization, as neighboring patches do not rationalize at the same rate across the cyclidic arcs (see Fig. 14), and so the coordinate systems do not line up.

An **inverse mapping** is needed here $f : \mathbb{R}^3 \mapsto (u_x, v_y, w_z)$, such that given a point p on a border, we can identify the coordinates of the two different systems to which it belongs, i.e. the corresponding coordinate values of u, v, w. This is simplified by the fact that we need only the inverse mapping at the boundary (i.e. face) where two volumes meet, so one of the coordinate values will already be known.

Given a point p on a surface patch—one of the faces of the curved hexahedron—we can find the constant coordinate surfaces on which it lies using the surface-finding expression;

$$\sigma^{i_{(n+t)}} = -p \cdot \kappa^{i_n} \tag{25}$$

where κ^{i_n} is a surface generator taking patch boundary surfaces σ^{i_n} to $\sigma^{i_{n+1}}$ and t is the coordinate value offset in the range (0, 1) we will next seek to identify (see Fig. 13). Note that this is the same simple expression used for defining surfaces throughout

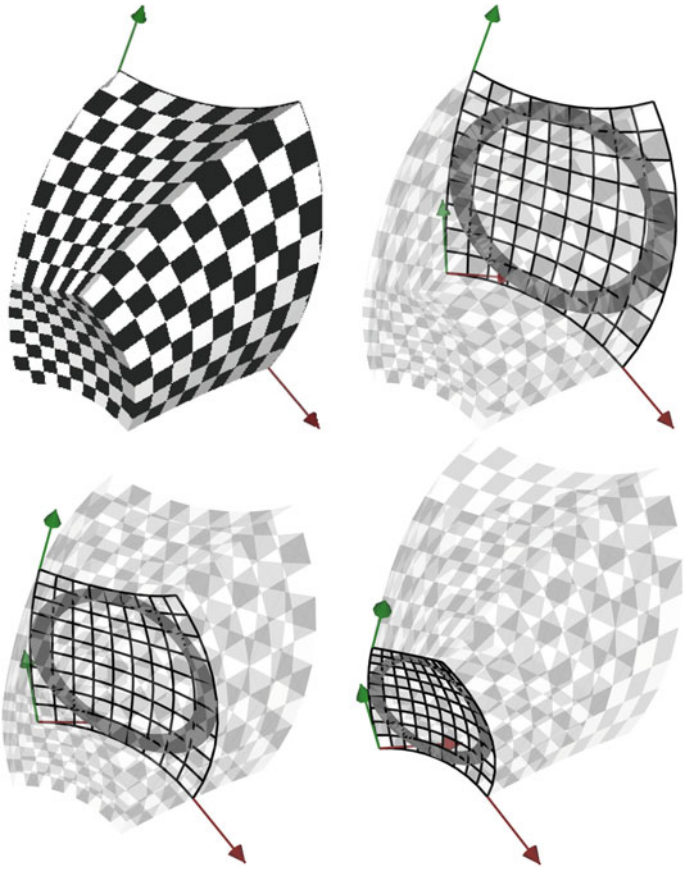


Fig. 11 A uv surface is transformed along the w direction, to define a 3D mapping that rationalizes the cyclidic volume. Above we draw the mapping of a circle at $w = z$ with $z \in \{0, .5, 1\}$. The surfaces of constant u and v at $w = z$, which are used to define the mapping, are found by first transforming the tangent vectors themselves and then relying on the orthogonality condition to extract the principal contact spheres.

this paper. The inverse mapping requires extracting the scalar coordinate value t which can be done by finding the ratio of surface transformation generators:

$$t = \kappa^{i_{n(t)}} \cdot (\kappa^{i_n})^{-1} \tag{26}$$

with

$$\kappa^{i_{n(t)}} = \frac{1}{2}\log((\widehat{\frac{\sigma^{i_{(n+t)}}}{\sigma^{i_n}}})) \tag{27}$$

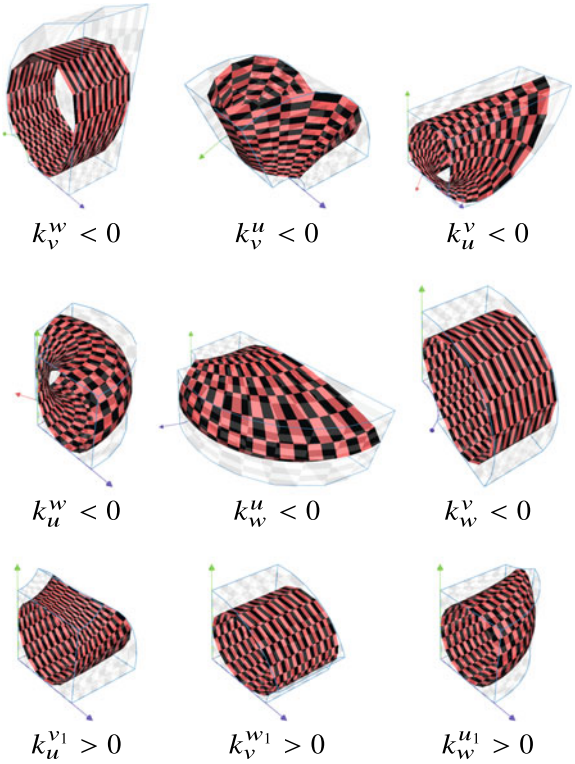


Fig. 12 A cyclidic volume defines a mapping which is here applied to a cylinder. Below each image we note the sign of the specific curvature coefficient controlling the deformation.

In this way, given a point on a face, we can identify the coordinate in both of the volume systems to which it belongs, enabling a way to traverse smoothly from one volume to the next. The ability to smoothly navigate connected volumes allows for warping more complex surfaces and topologies. Figure 15 demonstrates a step in this direction, by applying the lattices to deform a cylinder.

The above expression will return 0 if the two opposing surfaces are parallel planes. A more general expression is:

$$t = (n_o \cdot \kappa^{i_n(t)}) \cdot (n_o \cdot \kappa^{i_n})^{-1} \tag{28}$$

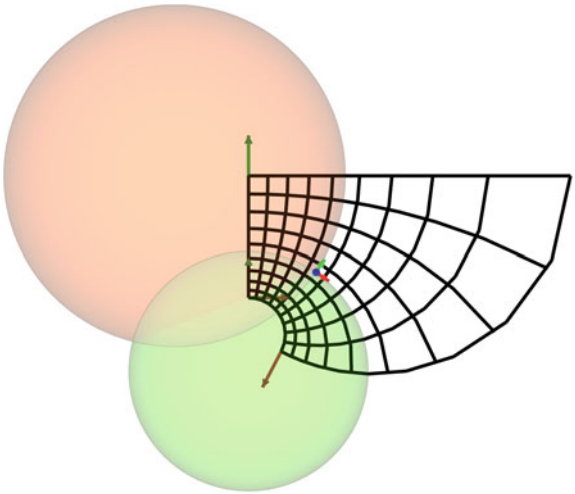


Fig. 13 Given a point p on the $w = 0$ surface patch, the sphere $-p \cdot \kappa_v^u$ (in red) and $-p \cdot \kappa_u^v$ (in green) are the surfaces of constant u and v at p. These surfaces can be used in Eqs. 26 and 27 to find the inverse mapping, which specifies a uv-coordinate value for p. In this example, p is in the middle of the patch, where $u = .5$ and $v = .5$.

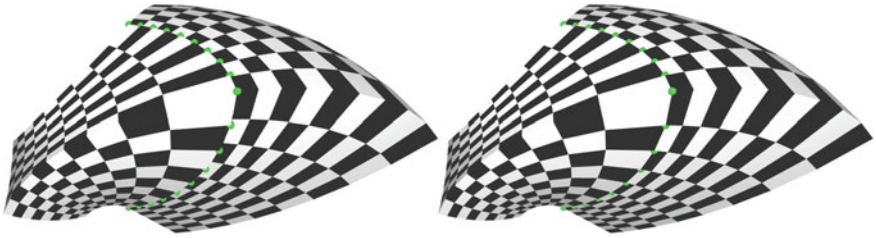


Fig. 14 With the conformal rotor rationalization outlined in Sect. 8, neighboring volumes will not coherently rationalize at the same rate, as visualized here with patches whose coordinate curves do not line up (left image). To account for this, the inverse mapping at a boundary can be used to connect the coordinate systems of neighboring patches (right image).

Fig. 15 Various deformations of a series of cylinders across multiple lattice volumes. On the left-most image we see in green the points that are inverse mapped in order to connect the series of volumes.

10 Conclusion

The precise, expressive, and unifying nature of conformal geometric algebra allows us to design carefully parameterized spatial systems with geometric primitives and versatile expressions. In this work, we began by explicitly encoding the concept of curvature at a point in space, and proceeded to walk through a novel algorithmic technique for defining a curved hexahedral volume of space using nine curvature coefficients and direct manipulation of principal tangent contact elements. To rationalize this space into a curvilinear coordinate system, we demonstrated that there are 24 principal surface spheres tangent to such a volume—4 for each face (e.g. 2 for each edge)—and that a composition of conformal transformations of one principal surface across to the opposing face can be applied to rationalize the volume. This provides us with a coordinate mapping $f : (u_x, v_y, w_z) \mapsto \mathbb{R}^3$ which can be used to warp the contents of the space, providing a potentially useful mechanism for computer aided design. To connect the coordinate systems of two adjacent volumes, we provide an inverse mapping $f : \mathbb{R}^3 \mapsto (u_x, v_y, w_z)$ to extract the coordinate values at a boundary. This inverse mapping can assist the construction of a network of lattices to bend and design more elaborate forms. With these techniques firmly in hand, we can now explore the range of motions and meshes facilitated by this space-bending approach to design.

Acknowledgements The author would like to thank Professor Leo Dorst for his essential corrections and comments during the preparation of this text. Of course, any omissions or errors are the author's alone.

References

1. Bo, P., Pottmann, H., Kilian, M., Wang, W., Wallner, J.: Circular arc structures. ACM Trans. Graph. **30**(4), 101:1–101:12 (2011)
2. Bobenko, A.I., Huhnen-Venedey, E.: Curvature line parametrized surfaces and orthogonal coordinate systems: discretization with Dupin cyclides. Geometriae Dedicata **159**(1), 207–237 (2012)
3. Colapinto, P.: Composing surfaces with conformal rotors. Adv. Appl. Clifford Algebras **27**(1), 453–474 (2017)
4. Colapinto, P.: Articulating space: geometric algebra for parametric design – symmetry, kinematics, and curvature. Ph.D. thesis, Media Arts and Technology Program, University of California Santa Barbara, March 2016
5. Crane, K., Pinkall, U., Schröder, P.: Spin transformations of discrete surfaces. ACM TOG 30, 4, 104:1–10 (2011)
6. Dorst, L., Valkenburg, R.: Square root and logarithm of rotors in 3D conformal geometric algebra using polar decomposition. In: Dorst, L., Lasenby, J. (eds.) Guide to Geometric Algebra in Practice, pp. 81–104. Springer (2011)
7. Dorst, L.: The construction of 3D conformal motions. Math. Comput. Sci. **10**(1), 97–113 (2016). https://doi.org/10.1007/s11786-016-0250-8
8. Dorst, L., Fontijne, D., Mann, S.: Geometric algebra for computer science: an object-oriented approach to geometry. Elsevier, Morgan Kaufmann, Amsterdam, San Francisco (2007)

9. Dupin, C.: Applications de géométrie de mécanique à la marine, aux ponts et chaussées; pour faire suite aux développements de géométrie. Paris (1822)
10. Hestenes, D.: New Foundations for Mathematical Physics, Chapter 2. http://geocalc.clas.asu.edu/html/NFMP.html
11. Hestenes, D., Sobczyk, G.: Clifford algebra to geometric calculus: a unified language for mathematics and physics. D. Reidel; Distributed in the U.S.A. and Canada by Kluwer Academic Publishers, Dordrecht (1984)
12. Huhnen-Venedey, E.: Curvature line parametrized surfaces and orthogonal coordinate systems. Discretization with Dupin cyclides. Master's thesis. Technische Universität Berlin, Institut für Mathematik, Berlin (2007)
13. Martin, R.R.: Principal patches for computational geometry. Ph.D. thesis. Pembroke College, Cambridge University (1983)
14. Krasauskas, R.: Unifying theory of Pythagorean-normal surfaces based on geometric algebra. Adv. Appl. Clifford Algebras **27**, 491–502 (2017)
15. Vaxman, A., Müller, C., Weber, O.: Conformal mesh deformations with Mobius transformations. ACM Trans. Graph. **34**(4), Article 55 (2015). https://doi.org/10.1145/2766915.

Exploring Lazy Evaluation and Compile-Time Simplifications for Efficient Geometric Algebra Computations

Leandro A. F. Fernandes

Abstract Mathematical function libraries for scientific computation play an essential role in scientific development. These libraries allow researchers to focus their efforts on solving higher-level problems while the implementations provided by the libraries make good use of available computer resources. The Geometric Algebra Template Library (`GATL`) is a C++ library of data structures and mathematical functions for arbitrary Geometric Algebras (GAs). `GATL` uses template metaprogramming to implement a lazy evaluation strategy at compile-time. This way, `GATL` is capable of performing optimizations on the programs during the compilation of executable files, reducing the computational cost that programs will have at runtime. More specifically, we have designed `GATL` to automatically execute low-level algebraic manipulation in the procedures described by the programmer using GA operations. The aim of `GATL` at compile-time is to simplify each described procedure by performing symbolic optimizations on expressions, leading to more efficient programs.

1 Introduction

It is well-known that Geometric Algebra (GA) is a powerful mathematical system encompassing concepts like Complex Numbers, Quaternion Algebra, Grassmann-Cayley Algebra, and Plücker Coordinates under the same framework [9, 16, 18, 21]. GA is mainly based on Clifford Algebra, but with a strong emphasis on geometric interpretation. As such, it is an appropriate mathematical tool for modeling and solving geometric problems in physics, chemistry, engineering, and computer science.

L. A. F. Fernandes (✉)
Instituto de Computação, Universidade Federal Fluminense (UFF),
Av. Gal. Milton Tavares de Souza, Niterói, Rio de Janeiro 24210-346, Brazil
e-mail: laffernandes@ic.uff.br

S. Xambó-Descamps (ed.), *Systems, Patterns and Data Engineering with Geometric Calculi*, SEMA SIMAI Springer Series 13, https://doi.org/10.1007/978-3-030-74486-1_6

This contribution discusses the application of the lazy evaluation strategy at compile time as a promising approach for the implementation of efficient programs based on GA. Lazy evaluation defers the evaluation of expressions until other computations need the expressions' results. We present the Geometric Algebra Template Library (GATL) as proof of concept of how to explore lazy evaluation at compile time to reduce both computational cost and memory footprint of programs. GATL is a high-level C++ library that includes a data structure that represents GA expressions that operate multivectors in arbitrary metric spaces and dozens of operations of this algebra. In contrast to other lazy solutions such as Gaalet [25], GATL conducts symbolic optimizations on expressions during compilation. Solutions like Gaalop [5, 17] also execute algebraic manipulations at compile time to simplify expressions. In this case, the solution uses an external tool to perform symbolic optimizations and replaces the original source code by the optimized counterpart version of it. GATL, on the other hand, performs symbolic optimizations by the ingenious use of the metaprogramming template capabilities of C++.

In Sect. 2, we present an overview of the implementation strategies that have been employed in defining code optimizers, libraries, and library generators for GA. Section 3 describes the internal structure of GATL and how it implements lazy evaluation and compile-time simplification. The performance of other solutions for GA is compared to GATL in Sect. 4. Finally, we draw our conclusions in Sect. 5.

! Attention

It is assumed that the reader is familiar with GA. Please refer to Dorst *et al.* [9], Kanatani [18], and Perwass [21] for a complete reference on the subject. Here you find the notational conventions used throughout this contribution:

$\mathcal{C}\ell_{r,p,q}$	Clifford Algebra with signature (p, q, r), where $n = p + q + r$.
e_i	i-th Unit basis vector that square to $+1$ in an orthonormal basis.
e_+, e_-	Extra unit basis vectors that square to $+1$ and -1, respectively.
n_o, n_∞	Null basis vectors interpreted, respectively, as origin point and point at infinity in conformal model.
ϕ	Angle in radians.
x, y, z	Scalar value, typically coordinates in 3-dimensional spaces.
a, b	General vector (1-blade).
$\mathcal{R}$	General rotor.
A, B	General multivector.
AB	Geometric product of A and B.
$A \wedge B$	Outer product of A and B.
$\widetilde{A}$	Reverse of A.

2 Overview of Implementation Strategies for Geometric Algebra

One can classify the solutions that provide data structures and mathematical functions to work with GA in terms of their type as *code optimizers*, *libraries*, or *library generators*. The solutions can employ two different strategies for evaluating the implemented operations. Namely, *lazy evaluation* and *eager evaluation*. We call *implementation strategy* the combination of a type of solution with an evaluation strategy. Table 1 relates some existing solutions with their classification regarding their type, evaluation strategy, supported data types at runtime, and programming languages.

Section 1 introduces the lazy evaluation as the strategy that defers the evaluation of expressions until other computations need the expressions' results. In eager evaluation, the arguments to a function are always evaluated completely before the function is applied. It is important to emphasize that the actual evaluation strategy is intrinsic to the programming language. All programming languages mentioned in this section employ the eager evaluation strategy. But in all of them, it is possible to implement data structures that simulate laziness. Therefore, the solutions indicated as "lazy" make use of implementation tricks to simulate laziness in eager languages.

Table 1 Classification of solutions for GA regarding their type, evaluation strategy, supported data types at runtime, and the programming languages

Solution	Type	Evaluation Strategy	Runtime Data types	Programming languages
`clifford` [1]	Library	Eager	Numeric	Python 3
`Gaalet` [25]	Library	Lazy	Numeric, Symbolic†	C++
`Gaalop` [5, 17]	Code Optimizer	Eager	Numeric	C, C++, CUDA, OpenCL, MATLAB
`Gaigen` [13]	Library Generator	Eager	Numeric	C, C++, C#, Java
`galgebra` [3, 23]	Library	Eager	Symbolic	Python 2, Python 3
`Gallant` [8, 12]	Library	Eager	Numeric	Java
`ganja.js` [7]	Library Generator	Eager	Numeric	C++, C#, Javascript, Python 3, Rust
`Garamon` [2]	Library Generator	Eager	Numeric	C++
`GATL`	Library	Lazy	Numeric, Symbolic	C++
`GluCat` [19]	Library	Eager	Numeric	C++, Python 2
`GMac` [10]	Code Optimizer	Eager	Numeric	C++, C#, VB.NET, F#, IronPython
`Grassmann.jl` [24]	Library	Eager	Numeric, Symbolic	Julia
`Klein` [20]	Library	Eager	Numeric	C++
`Liga` [4]	Library	Eager	Numeric	Julia
`TbGAL` [26]	Library	Eager	Numeric, Symbolic†	C++, Python 2, Python 3
`Versor` [6]	Library	Eager	Numeric, Symbolic†	C++

†As a template-based C++ solution, it is likely to support symbolic data types for multivector coefficients at runtime. However, this capability has not been asserted by the authors

Code Optimizers. When a programmer uses a code optimizer, he/she writes snippets of the source code of his/her program using the representation language provided by the optimizer. The representation language is not necessarily the same used to write the rest of the program. Before compiling the whole program, the optimizer analyzes the snippets of specialized code, maps the input variables to symbols, and converts the calls to operations of the algebra into mathematical expressions. Such expressions, in turn, are manipulated algebraically in order to simplify them. The result of the manipulation is then translated into source code in the programming language chosen by the programmer, and compiled with the rest of the program.

`Gaalop` [17] is an example of code optimizer for GA. It is a software that expects procedures implemented using `CLUScript` [22] as input and converts them into simpler C, C++, CUDA, OpenCL, or MATLAB code. `Gaalop` uses the `Maxima` system for the manipulation of symbolic expressions. Charrier *et al.* [5] developed a `Gaalop` Pre-Compiler for C++, CUDA, and OpenCL that takes `CLUScript` snippets declared in `pragma` directives [27] and optimize them producing inline code for a given source file. The source code produced by `Gaalop` is evaluated eagerly. Nevertheless, in theory, it does not show any performance issues at runtime since the simplification process runs before the compiling process is triggered.

`GMac` [10] is another code optimizer that produces code fragments from a description of an algorithm in a domain-specific language. It can be configured to generate textual code files using an API that is accessible through any .NET language, including C++, C#, VB.NET, F#, and IronPython. Currently, `GMac` depends on `Wolfram Mathematica` for symbolic processing. The generated code is evaluated eagerly.

Libraries. When using libraries, the programmer declares variables whose types are data structures provided by the solution and calls subroutines that represent GA operations implemented by the library. `clifford` [1], `galgebra` [3, 23], `Gallant` [8, 12], `GluCat` [19], `Grassmann.jl` [24], `Klein` [20], `Liga` [4], `TbGAL` [26], and `Versor` [6] are examples of GA libraries that employ the eager evaluation strategy. For eager libraries, the only possible optimizations are those that affect individual calls of subroutines, because eager evaluation solves each subroutine call before passing its result as an argument for the next call.

`Versor` uses the metaprogramming capabilities of C++ templates to perform compile-time specialization of subroutines based on the arguments passed to them, producing results that compute and store only the multivector components whose coefficients may be non-zero at runtime. To perform such simplification at compile-time, `Versor` assumes that the coefficients stored by the input multivectors are non-zero values since the actual values will only be known at runtime. In contrast, the basis blade associated with each coefficient is known at compile time. Through the algebraic manipulation of the basis blades of the input multivectors, the library can predict which components of the resulting multivector will be equal to zero and which will be different from zero, eliminating the need for calculating the former and maintaining the storage and computation of the latter. However, as illustrated in the following code,

eager evaluation prevents the library from suppressing intermediate results that would be unnecessary when considering a sequence of operations.

Program Code

Example of source code using the `Versor` library. The comments on the right present the state of each variable at runtime:

```
1  #include <vsr/space/vsr_ega3D_types.h>
2  #include <vsr/detail/vsr_generic_op.h>
3
4  using namespace vsr::ega;
5  using namespace vsr::nga;
6
7  int main() {
8    double x, y, z, phi_deg;
9    std::cin >> x >> y >> z; // x: 10, y: 20, z: 30
10   std::cin >> phi_deg;     // phi_deg: 90
11
12   double phi = phi_deg * (M_PI/180.0);    // phi: 1.5708
13
14   auto a = Vec(x, y, z);                  // a: 10*e1 + 20*e2 + 30*e3
15   auto R = Gen::rot(Biv::xy * (-phi/2)); // R: 0.7071 - 0.7071*e12
16   auto b = R * a * ~R;                    // b: -20*e1 + 10*e2 + 30*e3 + 0*e123
17
18   return EXIT_SUCCESS;
19 }
```

The $*$ and $\sim$ operators implement the geometric product and the reverse, respectively.

Discussion The rotation of a vector $a = x_1e_1 + y_1e_2 + z_1e_3$ made by a sandwiching product with a rotor $\mathcal{R} = \cos(\phi/2) - \sin(\phi/2)e_1 \wedge e_2$ under Euclidean metric, where ϕ and $e_1 \wedge e_2$ are, respectively, the rotation angle and the unit rotation plane, produces a vector $b = \mathcal{R}a\widetilde{\mathcal{R}} = x_2e_1 + y_2e_2 + z_2e_3$. By the grade preservation property of outermorphisms, the multivector components with grade different than 1 will always be zero in b. But the example shows that `Versor` does not suppress the computation of the component of grade 3 in `b` (line 16). In eager solutions, suppression is only possible by implementing specialized routines, such as `lhs.spin(rhs)`, which returns the multivector resulting from applying the rotor `rhs` to `lhs`.

Other eager libraries presented in Table 1 do not simplify subroutine according to their arguments at compile time. The only exception is `clifford` [1], which uses the just-in-time (JIT) compilation feature of Python to improve the performance of subroutine calls. The JIT functionality was extended for $C\ell_{4,1}$ by `Gajit` [15].

Libraries for GA that employ the lazy evaluation strategy provide two types of data structures to represent multivectors. The more straightforward kind of data structure represents *concrete multivectors*, *i.e.,* multivectors that store their components in memory at runtime. The other kind of data structure encodes (lazy) *multivector expressions*, *i.e.,* expressions obtained by operating concrete multivectors and other expressions. Typically, a multivector expression `arg` can be evaluated implicitly by assigning it to an existing concrete multivector variable or explicitly by calling functions like `eval(arg)` and `arg.eval()` that return a concrete multivector.

To the best of our knowledge, Gaalet [25] and GATL are the only libraries for GA that employ the lazy evaluation strategy. By combining C++ templates metaprogramming and lazy evaluation, both solutions extend the compile-time specialization capability of Versor to include the suppression of unnecessary computations over sequences of operations, inline function calls, and avoid storage of temporary values. The main difference between Gaalet and GATL is that GATL's lazy evaluation system of template expressions implements an algebraic manipulator that conducts symbolic manipulations equivalent to those performed by Gaalop, but without the need for an external tool. Also, unlike Gaalet and Versor, GATL allows multivector coefficients to assume known values at compile-time, thus reducing storage cost at runtime. This latter feature is especially useful in representing points with unit homogeneous coordinate and constant multivector.

The code examples that follow illustrate, respectively, the use of Gaalet and GATL in solving the same rotation case presented for Versor.

Program Code

Example of source code using the Gaalet library. The comments on the right present the state of each concrete variable at runtime, or the expression encoded by the non-concrete multivector variable b_ as nested operations:

```
1  #include <gaalet.h>
2
3  using ga3e = gaalet::algebra<gaalet::signature<3, 0>, double>;
4
5  int main() {
6    double x, y, z, phi_deg;
7    std::cin >> x >> y >> z; // x: 10, y: 20, z: 30
8    std::cin >> phi_deg;     // phi_deg: 90
9
10   double phi = phi_deg * (M_PI/180.0); // phi: 1.5708
11
12   ga3e::mv<1, 2, 4>::type a{x, y, z};              // a: 10*e1 + 20*e2 + 30*e3
13   ga3e::mv<0, 3>::type R{cos(phi/2), -sin(phi/2)}; // R: 0.7071 - 0.7071*e12
14
15   auto b_ = grade<1>(R * a * ~R); // b_: grade<1, gp<gp<mv<0, 3>, mv<1, 2, 4> >,
16                                   //        reverse<mv<0, 3> > > >
17
18   auto b = eval(b_);              // b: -20*e1 + 10*e2 + 30*e3
19
20   return EXIT_SUCCESS;
21 }
```

The $*$ and $\sim$ operators implement the geometric product and the reverse, respectively.

Discussion This example is equivalent to the one presented for Versor. However, notice in line 15 that the variable b_ represents a multivector expression instead of a concrete multivector. The last step in this expression is the extraction of the grade 1 components of the multivector resulting from the application of the rotor $\mathcal{R}$ to vector a. The nested structure of the expression in this variable is defined at compilation time. At runtime, variable b (line 18) receives a concrete multivector that does not include the calculation of the suppressed grade 3 component.

Program Code

Example of source code using the GATL library. The comments on the right present the state of each concrete variable at runtime, or the expression encoded by the non-concrete multivector variables a_, phi_, R_, and b_:

```
1  #include <gatl/ga3e.hpp>
2
3  using namespace ga3e;
4
5  int main() {
6    double x, y, z, phi_deg;
7    std::cin >> x >> y >> z; // x: 10, y: 20, z: 30
8    std::cin >> phi_deg;     // phi_deg: 90
9
10   double phi = phi_deg * (M_PI/180.0); // phi: 1.5708
11
12   auto a = vector(x, y, z);            // a: 10*e1 + 20*e2 + 30*e3
13
14   auto lazy = make_lazy_context(a, scalar(phi));
15   auto [a_, phi_] = lazy.arguments();  // a_: x_*e1 + y_*e2 + z_*e3, phi_: phi_
16
17   auto R_ = cos(phi_/c<2>) - sin(phi_/c<2>)*e1^e2; // R_: cos(phi_/2)
18                                                    //       - sin(phi_/2)*e12
19   auto b_ = R_ * a_ * ~R_; // b_: (cos(phi_)*x_ - sin(phi_)*y_)*e1
20                            //       + (sin(phi_)*x_ + cos(phi_)*y_)*e2 + z_*e3
21
22   auto b = lazy.eval(b_);  // b: -20*e1 + 10*e2 + 30*e3
23
24   return EXIT_SUCCESS;
25 }
```

The *, ^, and ~ operators implement the geometric product, outer products, and reverse, respectively. The helper functions `vector(arg1, ...)` and `scalar(arg)` in lines 12 and 14 create multivector structures from arithmetic types or scalar multivectors. The `make_lazy_context(arg1, ...)` function in line 14 takes a set of multivectors as input arguments and creates a data structure that relates each concrete coefficient and basis blade in the components of the input to symbols composing the multivector expressions returned by the `lazy.arguments()` function in line 15. Details of its operation will be presented in Sect. 3. The helper template `c<arg>` in line 17 initializes a scalar multivector that wraps a static constant value.

Discussion This example is equivalent to those presented for `Versor` and `Gaalet`. By comparing the expressions presented here for `b_` with the expression presented for the variable `b_` in the `Gaalet`'s example, we observe that, unlike `Gaalet`, `GATL` does not store expressions by nesting operations. The existence of the lazy context (line 14) allows algebraic manipulations to be performed at compile time while defining each new expression, leading to simpler computations (line 19) when calling the `lazy.eval(arg)` function at runtime (line 22).

Library Generators. For this kind of solution, the programmer first defines the parameters of his/her programming language and GA of interest in the generator software. The generator then produces implementations of data structures and GA operations from scratch. This set of implementations, in turn, can be used by the programmer like conventional libraries in writing his/her programs. `Gaigen` [13], `ganja.js` [7], and `Garamon` [2] are examples of library generators for GA (Table 1). The three solutions generate eager libraries but only `Gaigen` implements an optimization mechanism

for individual subroutines based on use cases. When using `Gaigen`, the programmer must generate the initial library without optimizations and use it in his/her program with the profiling functionality enabled. Profiling data is then interpreted by `Gaigen`, which produces an optimized version of the GA library by pruning unused multivector components. The final compilation of the program must be done considering the optimized version of the generated library.

3 The Geometric Algebra Template Library (`GATL`)

`GATL` is a C++17 template library defined in its headers files. There is no binary library to link to, no configured header file, or dependencies to external libraries. Therefore, if you want to use `GATL`, you can use the header files right away. Section 3.1 presents an overview of `GATL`'s front-end, *i.e.,* the set of data structure and subroutines available to the programmer while using the library. The internal organization and implementation of the library correspond to the back-end presented in Sect. 3.2.

3.1 `GATL`'s Front-End

According to the `GATL`'s conventions, the root directory for the header files that the programmer will include in his/her source files is the `gatl` folder. The header file that encloses all `GATL` implementations is `gatl/ga.hpp`. From this header, the programmer has access to the `ga` namespace, which is the main namespace of the `GATL` library. In C++, namespaces are declarative regions that provide scope to the names of the types, subroutines, and constants inside it. The following classes correspond to the most important data structures in `GATL`'s front-end:

`clifford_expression<CoefficientType, Expression>`
: A Clifford expression. The template parameter `CoefficientType` can be either a native arithmetic type (*e.g.,* `double`, `float`, `int`) or third-party classes implementing arbitrary-precision arithmetic or symbolic computation. It specifies the data type of the multivector's coefficients. The `Expression` parameter is a type describing the internal structure of the Clifford expression. Depending on the definition of this parameter, the Clifford expression will be classified as *concrete multivector* or (lazy) *multivector expression* (see Sect. 2).

`lazy_context<CliffordExpression`$_1$`, CliffordExpression`$_2$`, ...>`
: A class to define lazy arguments for lazy evaluation of Clifford expressions. It keeps references to the set of instances of `clifford_expression<...>` informed as input argument and produces multivector expressions having the concrete coefficients and basis blades in the input set replaced by symbols.

`metric_space<MetricSpaceType>`
The base metric space class from which all specialized metric space classes derive. The parameter `MetricSpaceType` must be one of those specialized spaces.

That's it. Only three classes! And most of the time, the programmer does not have to worry about parameterizing these classes since `GATL` provides helper functions and auxiliary headers for pre-defined GAs. Also, it is strongly recommended to use the `auto` placeholder type specifier [27] whenever possible.

The last program code presented in Sect. 2 illustrates the use of `GATL`. In this example, `gatl/ga3e.hpp` (line 1) is an auxiliary header defining the namespace `ga3e` (line 3) for a GA for 3-dimensional Euclidean geometry ($C\ell_{3,0}$) with basis vectors $\{e_1, e_2, e_3\}$. The specialized metric space class in this example is `euclidean_metric_space<3>`. It is used inside the header to declare the static constant object `space`. This object is implicitly passed as argument to all metric and non-metric products called in this example, such as the geometric and outer products (lines 17 and 19). `e1` and `e2` (line 17) are also static constant objects defined in the namespace `ga3e`. The constants `e1` and `e2` and the variables `a`, `a_`, `phi_`, `R_`, `b_`, and `b` are instances of the `clifford_expression<...>` class assuming different types in the `Expression` parameter. Notice the use of the `auto` placeholder to deduce the type of a variable from the initializer. In line 14, the `lazy` variable is of type `lazy_context<...>`. Typically, we use the helper function `make_lazy_context(arg1, ...)` to initialize it. We also use the helper functions `vector(arg1, ...)` and `scalar(arg)` to initialize Clifford expressions in lines 12 and 14. In line 17, the helper template `c<arg>` initializes a `clifford_expression<...>` whose `Expression` parameter wraps the constant scalar value 2. The difference between a wrapped constant value and a regular constant value is that the former can be handled at compile time by the symbolic simplification mechanism implemented by `GATL`'s back-end (Sect. 3.2).

In its current version, `GATL` includes the following set of namespaces for specific GAs. These namespaces already use the `ga` namespace. Also, they overload all metric operations in `ga` by setting their respective metric:

`ga1e`, `ga2e`, `ga3e`, `ga4e`, `ga5e`
GAs for Euclidean geometry ($C\ell_{n,0}$), with basis vectors $\{e_1, e_2, \cdots, e_n\}$.

`ga1h`, `ga2h`, `ga3h`, `ga4h`
GAs for homogeneous geometry ($C\ell_{d+1,0}$), with basis vectors $\{e_1, e_2, \cdots, e_d, e_+\}$.

`ga1m`, `ga2m`, `ga3m`
GAs for Minkowski spaces ($C\ell_{d+1,1}$), with basis vectors $\{e_1, e_2, \cdots, e_d, e_+, e_-\}$.

`ga1c`, `ga2c`, `ga3c`
GAs for conformal geometry ($C\ell_{d+1,1}$), with basis vectors $\{e_1, e_2, \cdots, e_d, n_o, n_\infty\}$.

The header file for each namespace is its name followed by the `.hpp` extension, *e.g.*, `gatl/ga3e.hpp`, `gatl/ga3h.hpp`, `gatl/ga3c.hpp`, and so on. Please, refer to examples in the `GATL` repository [11] to see how to declare Clifford Algebras $C\ell_{r,p,q}$ with arbitrary (p, q, r) signatures and assuming arbitrary metric matrices.

The documentation in [11] also includes the complete reference to helper functions and GA operations implemented by GATL. Among them, we highlight:

`+rhs`	Unary plus.
`-rhs`	Unary minus.
`lhs + rhs`	Addition.
`lhs - rhs`	Subtraction.
`gp(lhs, rhs [, mtr])`	Geometric product (same as `lhs * rhs`).
`op(lhs, rhs [, mtr])`	Outer product (same as `lhs ^ rhs`).
`rp(lhs, rhs [, mtr])`	Regressive product.
`lcont(lhs, rhs [, mtr])`	Left contraction (same as `lhs < rhs`).
`rcont(lhs, rhs [, mtr])`	Right contraction (same as `lhs > rhs`).
`dot(lhs, rhs [, mtr])`	Dot product (same as `lhs | rhs`).
`hip(lhs, rhs [, mtr])`	Hestenes' inner product.
`sp(lhs, rhs [, mtr])`	Scalar product.
`cp(lhs, rhs [, mtr])`	Commutator product.
`dp(lhs, rhs [, tol] [, mtr])`	Delta product.
`conjugate(arg)`	Clifford conjugation.
`involute(arg)`	Grade involution.
`reverse(arg)`	Reversion (same as `~arg`).
`rnorm_sqr(arg [, mtr])`	Squared reverse norm.
`rnorm(arg [, mtr])`	Reverse norm.
`inv(arg [, mtr])`	Inverse of the given versor.
`dual(arg [, pseudoscalar [, mtr]])`	Dualization operation.
`undual(arg [, pseudoscalar [, mtr]])`	Undualization operation.

According to GATL conventions, `lhs` and `rhs` are informal shorthand for, respectively, the left-hand side and the right-hand side arguments of binary operations. The `mtr` argument must be an instance of the `metric_space<...>` class, while all other arguments can be either instances of the `clifford_expression<...>` class, native arithmetic types, or third-party classes implementing arbitrary-precision arithmetic or symbolic computation.

3.2 *GATL's Back-End*

All namespaces mentioned in Sect. 3.1 declare a nested `detail` namespace. This is the namespace where the magic happens, *i.e.,* the namespace of GATL's back-end. All data types described in this section are defined in the `detail` namespace.

3.2.1 Expression Structure

The behavior of `GATL` at compile time and runtime is related to the definition of the `Expression` parameter of the instances of `clifford_expression<...>` involved in each operation. Recall that as a template parameter, `Expression` is a type, not an instance. It represents the description of the structure of the respective instance of `clifford_expression<...>`. The possible types for `Expression` are:

`component<Coefficient, BasisBlade>`
: A single multivector component whose coefficient is described by the template parameter `Coefficient` as a real-valued expression, and a basis blade described by the template parameter `BasisBlade`.

`add<Component`$_1$`, Component`$_2$`, ...>`
: The addition of two or more components of type `component<Coefficient, BasisBlade>` having different basis blades.

When the `BasisBlade` parameter is `constant_basis_blade<BasisVectors>`, it means that the basis blade of the component is known at compile time. Here, `BasisVectors` is an unsigned integer value whose bitset represents an unit basis blade. For instance, in Euclidean geometry, $1 = 0001_b$ stands for e_1, $2 = 0010_b$ stands for e_2, $3 = 0011_b$ stands for $e_1 \wedge e_2$, and so on.

Basis blades defined at runtime are represented by setting `BasisBlade` to `dynamic_basis_blade<PossibleGrades, Bitset>`, where `PossibleGrades` is an unsigned integer value known at compile time, indicating the grades that the runtime-defined component may assume (*e.g.,* $1 = 0001_b$ stands for grade 0, $2 = 0010_b$ stands for grade 1, $3 = 0011_b$ stands for grades 0 and 1, and so on). It is important to notice that one may predict the possible grades of the outcome of a GA operation if you know the grades of the arguments, even when the actual basis blades are unknown. For instance, the grade of the outer product of a 2-blade and a 3-blade will be 5 unless $n < 5$. In this case, the resulting grade may be set to any value, and the resulting coefficient will be 0 for sure. `GATL` explores this observation to perform simplifications at compile-time, even on expressions with runtime-defined multivector. In components with dynamic basis blades, the `Bitset` parameter is a type describing a bitwise expression with at least one bitset defined at runtime.

So far, only two templates parameters where not described in detail: `Coefficient` and `Bitset`. In `GATL`, the atomic types for `Coefficient` expressions are:

`constant_value<Value>`
: This type wraps a compile-time defined integer value.

`stored_value`
: This type indicates that the value of the coefficient is stored by the current instance of `clifford_expression<...>`.

`get_value<Tag, Index>` and `get_map_values<Tag, Index>`
: These types can be interpreted as variables within an algebraic expression. The pair of compile-time defined integer values `Tag` and `Index` makes it possible for a `lazy_context<...>` to unequivocally identify the value stored by one of the instances of `clifford_expression<...>` informed to it as initialization argument. `Tag` indexes the argument passed to the lazy context and `Index` indexes the value or values stored by this argument. To different lazy contexts do not confuse their arguments by defining conflicting `Tag` values, each `lazy_context<...>` instance deduces the minimum value it can assign to `Tag` by checking which values have already been used in its arguments.

`function<Name, Argument`$_1$`, Argument`$_2$`, ...>`
: The function represented by this type is defined by the `Name` parameter at compile time. The values that `Name` can assume includes, but is not limited to, `add`, `mul`, `power`, `sine`, `cosine`, and `if_else`. The expected number of arguments depends on the function's name, and they can encode real-valued expressions or logical expressions defined on the same set of atomic types.

The atomic types for `Bitset` are similar to those defined for `Coefficient`: `constant_value<Value>`, `stored_bitset`, `get_bitset<Tag, Index>`, `get_map_bitsets<Tag, Index>`, and `function<Name, Argument`$_1$`, ...>`. The difference is that atomic types for `Bitset` expressions are related to basis blades instead to the values of coefficients. Thus, among the possible values for the `Name` parameter, we highlight `bitwise_and`, `bitwise_or`, `bitwise_xor`, and `if_else`.

When the `Expression` parameter of a `clifford_expression<...>` only includes components with `constant_basis_blade<...>`, `constant_value<...>`, and `function<...>` types, we say that we have a *concrete multivector completely defined at compile time*. These multivectors do not occupy space in the compiled program because they do not store anything. Also, they do not depend on other multivectors. The `e1` object in the sample code is an example of such multivector type:

```
  clifford_expression<           // e1
    long,                        // CoefficientType
    component<                   // Component
      constant_value<1>,         //   Coefficient (same as 1)
      constant_basis_blade<1>    //   BasisBlade (1 = 0001_b, or e_1)
    >
  >
```

The objects `e2` and `c<2>` are also concrete multivectors defined at compile time.

The object `b` is an example of *concrete multivector defined at runtime*. As can be seen in its type definition, `b` stores three double-precision floating-point values:

```
clifford_expression<           // b
  double,                      // CoefficientType
  add<
    component<                 // First Component
      stored_value,            //   Coefficient (runtime defined)
      constant_basis_blade<1>  //   BasisBlade (1 = 0001_b, or e_1)
    >,
    component<                 // Second Component
      stored_value,            //   Coefficient (runtime defined)
      constant_basis_blade<2>  //   BasisBlade (2 = 0010_b, or e_2)
    >,
    component<                 // Third Component
      stored_value,            //   Coefficient (runtime defined)
      constant_basis_blade<4>  //   BasisBlade (4 = 0100_b, or e_3)
    >
  >
>
```

Thus, the size of `b` at runtime is 3×8 bytes $= 24$ bytes. In `GATL`, concrete multivectors store their runtime-defined coefficients and bitsets using sequence containers of type `std::array` or associative containers of type `std::map`. The associative container is used when more than one component of the multivector resulting from an operation have overlapping sets of possible grades, and runtime-defined bitsets. In this case, the description of the `Expression` parameter is simplified in the final `clifford_expression<...>` type and all components having overlapping `PossibleGrades` sets are put in the same `std::map`. Otherwise, the sequential container is chosen to store the data. Notice that it is possible for a `clifford_expression<...>` type to use `std::array` and `std::map` simultaneously to store componets with different configurations.

Lazy multivector expressions depend on concrete multivectors to have the values of their coefficients or the bitsets of their components computed at runtime. In `GATL`, the `Expression` parameter of all lazy `clifford_expression<...>` objects includes at least one getter type. For instance, the type of the `R_` object in the sample code includes `get_value<2, 0>`. Thus, the object `R_` depends on the 1st coefficient (`std::array` is zero-base indexed [27]) of the 2nd argument of the lazy context:

```
clifford_expression<           // R_
  long,                        // CoefficientType
  add<
    component<                 // First Component
      function<                //   Coefficient (same as cos(φ/2))
        cosine,
        function<
          mul,
          function<power, constant_value<2>, constant_value<-1> >,
          get_value<2, 0>
        >
      >,
      constant_basis_blade<0> //   BasisBlade (0 = 0000_b, or 1)
    >,
    component<                 // Second Component
      function<                //   Coefficient (same as -sin(φ/2))
        mul,
        constant_value<-1>,
        function<
          sine,
          function<
            mul,
            function<power, constant_value<2>, constant_value<-1> >,
            get_value<2, 0>
          >
        >
      >,
      constant_basis_blade<3> //   BasisBlade (3 = 0011_b, or e1 ∧ e2)
    >
  >
>
```

Lazy multivector expressions do not store anything. Thus, variables of this kind do not take up memory space at runtime. Of course, if you compile your program in debug mode, then these variables will have 1 byte each to be addressed by the debugger. However, in release mode, they are usually optimized by the compiler.

3.2.2 Expression Simplification and Evaluation

When we state that GATL employs the lazy evaluation strategy, we are referring to the possibility for the programmer to start a lazy context, operate the lazy context arguments by calling GA operations, and evaluate such expressions later. However, strictly speaking, the explicit use of a lazy context by the programmer is optional. This means that the sample program code presented for GATL could have been written without using the lazy context. However, in this case, the evaluation of the sequence of operations would be in an eager fashion, and the lazy multivectors would be concrete multivectors with coefficients defined at runtime. In addition, variable b would have the extra component of grade 3 observed in the sample code of the Versor library, since the suppression of this component would not be foreseen by GATL. The programmer

would choose to use a lazy context if he/she thinks that there are pertinent simplifications to be made in a certain part of his/her program.

Nevertheless, all subroutines implemented by GATL initializes a lazy context to benefit from any simplifications that may arise from the algebraic manipulation of the components of its arguments. It means that the implementation of all GATL subroutines looks like this:

```
template<class CT1, class E1, class CT2, class E2, class MST>
decltype(auto) foo(
  clifford_expression<CT1, E1> const &arg1,
  clifford_expression<CT2, E2> const &arg2,
  metric_space<MST> const &mtr
) {
  auto [lazy, arg1_, arg2_] = make_lazy_context_tuple(arg1, arg2);
  // Compute 'result_' using 'arg1_', 'arg2_', and 'mtr'.
  return lazy.eval(result_);
}
```

Here, CT*i* and E*i* encode the types assumed by, respectively, the template parameters `CoefficientType` and `Expression` of the *i*-th argument of the function `foo(…)`, `MST` is the `MetricSpaceType` parameter, and `make_lazy_context_tuple(…)` is a helper shorthand function for calling `lazy = make_lazy_context(…)` and `[…] = lazy.arguments()`. C++14 introduced `decltype(auto)` to delay the return type deduction after the dust of template instantiation has settled [27].

In high-level operations like dualization, inversion, and versor application, the implementation placed between the creation of the lazy context and the evaluation of the result is similar to what is expected from the user of the library. That is, by calling products and grade-dependent sign-change operations implemented by GATL as subroutines. Core operations, on the other hand, are implemented in a special way. They apply generative template metaprogramming to process the `Expression` parameters of the input arguments and produce the type set to the `Expression` parameter of the non-concrete multivector `result_`. It is at this moment that GATL applies algebraic simplification. For example, geometric product, outer product, regressive product, and inner products implement a set of recursive templates to apply distributivity over addition, and partial template specialization to enable conditional branching to suppress arithmetic operations that equal compile-time defined constant values. More specifically, when the compiler instantiates a template that implements a core operation, GATL's lexicographic expression ordering convention tries to put the subexpressions defined on the same `get_value<…>` (or `get_bitset<…>`) types close to each other. Whenever predefined patterns are identified, they are replaced by simpler equivalent expressions.

Calling the `lazy.eval(result_)` function causes the instantiation of a new `clifford_expression<…>` object. The `Expression` parameter of the new object will be derived from the `Expression` parameter describing `result_`. The coefficients and bitsets stored by the new object (if any) are computed by the lazy context by traversing the `result_`'s `Expression` parameter and solving

subexpressions defined on the occurrences of `get_value<...>`, `get_map_values<...>`, `get_bitset<...>`, and `get_map_bitsets<...>` related to instances of the `clifford_expression<...>` objects given as input. The current lazy context instance does not evaluate subexpressions defined on coefficients and bitsets with `Tag` values coming from other lazy contexts, nor subexpressions completely defined on constant values. It is because it is clear that the evaluation of these subexpressions must be deferred to another part of the program.

4 Experimental Results

We have used the GA Benchmark Project [14] to assess the execution time of `GATL` in comparison to other C++ solutions for GA. The GA Benchmark Project started from informal conversations among developers who attended to AGACSE 2018, in Campinas, Brazil. The idea was to build a suitable environment to compare GA solutions. It is an effort to define standards and methodologies for benchmarking GA code optimizers, libraries, and library generators. The goal of the project is to help physicists, chemists, engineers, and computer scientists to choose the GA solution that best suits their practical needs, as well as to push further the improvement of the compared solutions and to motivate the development of new tools. GA Benchmark is built on the Google Benchmark, a open source library to benchmark code snippets.

The GA Benchmark version 2.0.3 includes the evaluation of twelve binary operations (commutator product, geometric product, inverse geometric product, dot product, Hestenes' inner product, left contraction, right contraction, scalar product, outer product, regressive product, addition, and subtraction) and ten unary operations (dualization, undualization, versor inversion, normalization under reverse norm, squared reverse norm, unary minus, unary plus, Clifford conjugation, grade involution, and reversion) applied to k-blades having grades ranging from $k = 0$ to $k = n$ on eleven models of geometry (Euclidean models with basis vectors $\{e_1, \cdots, e_n\}$ and $n \in \{2, 3, 4, 5\}$, homogeneous models with basis vectors $\{e_1, \cdots, e_{n-1}, e_+\}$ and $n \in \{3, 4, 5\}$, Minkowski spaces with basis vectors $\{e_1, \cdots, e_{n-2}, e_+, e_-\}$ and $n \in \{4, 5\}$, and conformal models assuming $\{e_1, \cdots, e_{n-2}, n_o, n_\infty\}$ and $n \in \{4, 5\}$). GA Benchmark also includes the evaluation of an inverse kinematics algorithm assuming the conformal model of 3-dimensional Euclidean space, where $n = 5$ and the set of basis vectors depends on the solution.

For each possible configuration of input grades in unary and binary operation, GA Benchmark generates random blades (or pairs of random blades) and measures the mean execution times of 30 evaluations of the operation. For the inverse kinematics algorithm, the inputs are random sets having five angular values each. The time required to build input data is not considered when calculating the execution times.

The benchmark is ready to compare seven C++ solutions for GA. Namely, `Gaalet` [25], `Gaalop` [5], `Garamon` [2], `GATL`, `GluCat` [19], `TbGAL` [26], and `Versor` [6]. For `GluCat`, the benchmark includes comparison considering framed-based and matrix-based multivectors. We performed the comparison in a notebook running

Table 2 Ranking of performance of the compared solutions on binary and unary operations according to the gold first method

Solutions	Medals								Compilation errors
	1	2	3	4	5	6	7	8	
`GATL`	247	75	1	0	0	0	0	0	0
`Versor`	246	76	1	0	0	0	0	0	0
`Gaalop`	246	18	0	0	0	0	0	0	59
`Gaalet`	238	26	0	0	0	0	0	0	59
`TbGAL`	34	56	97	107	26	3	0	0	0
`Garamon`	26	255	42	0	0	0	0	0	0
`GluCat` (framed)	6	60	151	72	22	12	0	0	0
`GluCat` (matrix)	0	3	45	108	151	16	0	0	0

Ubuntu 18.04 operating system (Linux kernel version 4.4.0) on bare metal. The computer was equipped with 16 Gb of RAM and one Intel Core i7-8550U processor with 1.99 GHz and 8 cores. The C++ source codes were compiled using GCC 7.4.0 with `O3` optimization in release mode and single thread. The tables and charts that follow only show results considering conformal, Euclidean, and Minkowski geometries, since homogeneous and Euclidean models are equivalent. The complete set of log files produced for the experiments, as well as detailed charts for all operations and models of geometry, are available at the GitHub repository of the GA Benchmark project as the results reported on February 5th, 2020.

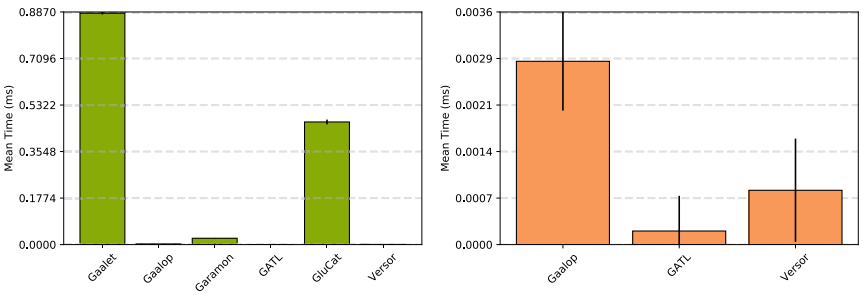
Table 2 classifies the compared solutions using the gold first method, *i.e.,* based first on the number of gold medals, then silver, and so on. A solution receives a gold medal (medal #1) whenever its performance is better than that of other solutions in a particular case of binary or unary operation with input arguments having specific grades. The second best-placed solution receives a silver medal (medal #2), and so on. The medals are distributed among the solutions for testing cases implemented as native subroutines and models of geometry by all of them. Therefore, only Euclidean and Minkowski models were considered here, because `Gaalet` and `GluCat` implement conformal geometry assuming $\{e_1, e_2, \cdots, e_+, e_-\}$ as basis vectors (like the Minkowski model) instead of $\{e_1, e_2, \cdots, n_o, n_\infty\}$. Regarding the set of available operations, unfortunately, the front-end of most of the solutions is incomplete. As can be seen in Table 3, the only operations implemented by all solutions are geometric product, outer product, and reversion. From the results in Table 2, it is possible to conclude that the code optimizer (*i.e.,* `Gaalop`) and libraries that explore template metaprogramming to perform compile-time specialization of subroutines (*i.e.,* `Gaalet`, `GATL`, and `Versor`) present equivalent performance when the mean processing time of common operations are considered. The last column of Table 2 shows the number of cases where the benchmark program could not be compiled due to errors raised by the solution.

Table 3 Binary and unary operations implemented as native subroutines by the compared solutions

	Binary Operations												Unary Operations									
	Main Products			Inner Products					Non-Metric Products		Sum Operations		Dualization Operations		Norm-Based Operations			Plus/Minus Operations		Sign-Change Operations		
Solutions	Commutator Product	Geometric Product	Inverse Geometric Product	Dot Product	Hestenes' Inner Product	Left Contraction	Right Contraction	Scalar Product	Outer Product	Regressive Product	Addition	Subtraction	Dualization	Undualization	Versor Inversion	Normalization	Squared Reverse Norm	Unary Minus	Unary Plus	Clifford Conjugation	Grade Involution	Reversion
`Gaalet`	-	✓	-	-	✓	-	-	✓	✓	-	✓	✓	✓	-	✓	-	-	-	-	-	-	✓
`Gaalop`	-	✓	✓	-	✓	-	-	-	✓	-	✓	✓	✓	-	-	-	-	✓	-	-	-	✓
`Garamon`	-	✓	✓	-	✓	✓	✓	✓	✓	-	✓	✓	✓	-	✓	-	✓	✓	-	-	-	✓
`GATL`	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
`GluCat`	-	✓	✓	-	✓	✓	-	-	✓	-	✓	✓	-	-	✓	-	✓	✓	-	✓	✓	✓
`TbGAL`	-	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
`Versor`	✓	✓	✓	-	-	✓	-	-	✓	-	✓	✓	✓	✓	✓	✓	✓	✓	-	✓	✓	✓

The comparison of the mean execution times of complete algorithms aims to verify the ability of each solution to induce the optimization of sequences of operations and not only of individual subroutines. Figure 1 shows the mean execution times of the inverse kinematics algorithm. The GA Benchmark does not included `TbGAL` in this comparison because the original algorithm performs the addition of general multivectors and `TbGAL` only implements the addition and subtraction operations of scalar values, vectors, pseudovectors, and pseudoscalars. It is because `TbGAL` stores blades and versors as collections of scalar and vector factors instead of as the weighted sums of basis blades. In Fig. 1(a), we only include the result obtained for the frame-based version of `GluCat`, because the mean execution time of the matrix-based version of the library is four times higher for this algorithm. Since both frame and matrix-based versions belong to the same library, we used the one with better performance. In Fig. 1(b) we highlight the Top-3 solutions. The vertical lines on each of the bars on the graph indicate one standard deviation confidence interval.

Surprisingly, according to Fig. 1(a), `Gaalet` proved to be the least efficient solution. We believe that better results can be achieved if the user carefully inspects the outcome of each sequence of operations. In this way, he/she will be able to instruct



(a) Results for six solutions. (b) Detailed view of the Top-3 solutions.

Fig. 1 Mean execution times of the inverse kinematics algorithm implemented using six C++ solutions for GA (a), and the detailed view of the results of the Top-3 solutions (b)

the library to suppress unnecessary components in the intermediate results. But such a task can be difficult and laborious.

When comparing `Garamon` and `GluCat`, we conclude that `Garamon` achieved better results because it stores the multivector components as per grade arrays, while `GluCat` uses dictionaries to store individual components. The high cost of both solutions is related to the time needed to manage the dynamic memory used by the algorithm's intermediate variables.

The detailed view of the results obtained for `Gaalop`, `GATL`, and `Versor` is presented in Fig. 1(b). The proposed library achieved better results because the lazy evaluation and compile-time simplification mechanisms were able to eliminate unnecessary multivector components and arithmetic operations. The same degree of optimization is not achieved by the eager evaluation strategy employed by `Versor`. The optimized codes generated by `Gaalop` show that the lack of performance of the solution in this algorithm is related to the substitution of expressions such as x^2, x^3, etc., by calls to the `std::pow(base, exponent)` function. `GATL`, on the other hand, uses one or two multiplications to raise x to the powers 2, 3, and 4. The `std::pow(...)` function requires many more processing cycles than multiplication.

5 Concluding Remarks

This contribution presents `GATL`, a C++ library that applies the lazy evaluation strategy and template metaprogramming to conduct symbolic optimizations on expressions at compile-time to improve the runtime performance of GA-based programs.

In addition to runtime performance, `GATL` is concerned with being user friendly and intuitive. In other words, we expect the implementation of equations written with GA to be as straightforward as possible without compromising the proper use of available computing resources. To this end, `GATL` offers dozens of GA operations ready for use and completely integrated with the lazy evaluation concept.

Currently, GATL does not include modules for data visualization. However, in our experience, the results produced with GATL can be easily integrated with other visualization solutions, such as `ganja.js` [7].

GATL supports Clifford Algebras assuming metric matrices with arbitrary (p, q, r) signatures and arbitrary sets of basis vectors. The practical use of GATL's current version is limited to spaces with up to $n = p + q + r = 7$ dimensions, except in special situations where the operated multivectors are made up of a small amount of components. This limitation is also observed in other templates-based GA libraries such as `Versor` and `Gaalet`. It is related to the ability of the compiler to analyze and instantiate the templates. Fortunately, each new version of the C ++ language includes features that allows us to replace complex templates by simpler counterparts. Consequently, the expectation is that in the future it will be possible to work with algebras in higher dimensions and produce programs that compile in less time.

The development of computational tools for GA is a living ecosystem. We hope that the ideas presented in this contribution will help developers to improve existing solutions and serve as inspiration for the development of innovative tools.

Acknowledgements This work was sponsored by CNPq-Brazil (grant 311.037/2017-8) and FAPERJ (grant E-26/202.718/2018).

References

1. Arsenovic, A., Hadfield, H., Antonello, J., Kern, R., Boyle, M.: Numerical geometric algebra module for Python (2018). https://github.com/pygae/clifford
2. Breuils, S., Nozick, V., Fuchs, L.: Garamon: a geometric algebra library generator. Adv. Appl. Clifford Algebras **29**(4), 69 (2019)
3. Bromborsky, A.: Symbolic geometric algebra/calculus package for SymPy (2015). https://github.com/brombo/galgebra
4. Castelani, E.V.: Library for geometric algebra (2017). https://github.com/evcastelani/Liga.jl
5. Charrier, P., Klimek, M., Steinmetz, C., Hildenbrand, D.: Geometric algebra enhanced precompiler for C++, OpenCL and Mathematica's OpenCLLink. Adv. Appl. Clifford Algebras **24**(2), 613–630 (2014)
6. Colapinto, P.: Versor: spatial computing with conformal geometric algebra. Master's thesis, University of California at Santa Barbara (2011)
7. De Keninck, S.: Javascript geometric algebra generator for Javascript, C++, C#, Rust, Python
8. Dijkman, D.H.F.: Efficient implementation of geometric algebra. Ph.D. thesis, Universiteit van Amsterdam (2007)
9. Dorst, L., Fontijine, D., Mann, S.: Geometric Algebra for Computer Science: An Object Oriented Approach to Geometry. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann Publishers, Amsterdam (2007)
10. Eid, A.H.: An extended implementation framework for geometric algebra operations on systems of coordinate frames of arbitrary signature. Adv. Appl. Clifford Algebras **28**(1), 16 (2018)
11. Fernandes, L.A.F.: GATL: geometric algebra template library (2020). https://github.com/laffernandes/gatl
12. Fontijne, D.: Implementation of Clifford algebra for blades and versors in $O(n^3)$ time. In: Talk at International Conference on Clifford Algebra (2005)

13. Fontijne, D.: Gaigen 2: a geometric algebra implementation generator. In: Proceedings of the 5th International Conference on Generative Programming and Component Engineering, pp. 141–150 (2006)
14. GA Developers: GA-Benchmark: a benchmark for geometric algebra libraries, library generators, and code optimizers (2020). https://github.com/ga-developers/ga-benchmark
15. Hadfield, H., D., H., A., A.: Gajit: symbolic optimisation and JIT compilation of geometric algebra in Python with GAALOP and Numba. In: Gavrilova, M., Chang, J., Thalmann, N., Hitzer, E., Ishikawa, H. (eds.) Advances in Computer Graphics – Computer Graphics International Conference (CGI). Springer (2019)
16. Hestenes, D.: New Foundations for Classical Mechanics. Reidel Publishing Company (1987)
17. Hildenbrand, D., Pitt, J., Koch, A.: Gaalop - high performance parallel computing based on conformal geometric algebra. In: Bayro-Corrochano, E., Scheuermann, G. (eds.) Geometric Algebra Computing, pp. 477–494. Springer, London (2010)
18. Kanatani, K.: Understanding Geometric Algebra: Hamilton, Grassmann, and Clifford for Computer Vision and Graphics. CRC Press (2015)
19. Leopardi, P.C.: GluCat: generic library of universal Clifford algebra templates (2007). http://glucat.sourceforge.net/
20. Ong, J.: $p(r*_{3,0,1})$ specialized SIMD geometric algebra library (2020). https://github.com/jeremyong/klein
21. Perwass, C.: Geometric Algebra with Applications in Engineering. Springer Publishing Company (2009)
22. Perwass, C., Gebken, C., Grest, D.: CluViz: interactive visualization (2004). http://cluviz.de
23. Pythonic Geometric Algebra Enthusiasts: Symbolic geometric algebra/calculus package for SymPy (2017). https://github.com/pygae/galgebra
24. Reed, M.: <Leibniz-Grassmann-Clifford-Hestenes> differential geometric algebra multivector simplicial-complex (2017). https://github.com/chakravala/Grassmann.jl
25. Seybold, F.: Gaalet: geometric algebra algorithms expression templates (2010). https://sourceforge.net/projects/gaalet/
26. Sousa, E.V., Fernandes, L.A.F.: TbGAL: a tensor-based library for geometric algebra. Adv. Appl. Clifford Algebras **30**(2), 27 (2020)
27. Standard C++ Foundation: ISO International Standard ISO/IEC 14882:2017(E) – Programming Language C++. IOS (2017). https://isocpp.org/std/the-standard

A Quaternion Deterministic Monogenic CNN Layer for Contrast Invariance

Eduardo Ulises Moya-Sánchez, Sebastià Xambó-Descamps, Sebastián Salazar Colores, Abraham Sánchez Pérez, and Ulises Cortés

Abstract Deep learning (DL) is attracting considerable interest as it currently achieves remarkable performance in many branches of science and technology. However, current DL cannot guarantee capabilities of the mammalian visual systems such as lighting changes. This paper proposes a deterministic entry layer capable of classifying images even with low-contrast conditions. We achieve this through an improved version of the quaternion monogenic wavelets. We have simulated the atmospheric degradation of the CIFAR-10 and the Dogs and Cats datasets to generate realistic contrast degradations of the images. The most important result is that the accuracy gained by using our layer is substantially more robust to illumination changes than nets without such a layer.

E. U. Moya-Sánchez (✉)
Gobierno de Jalisco, Guadalajara, Mexico

Universidad Autónoma de Guadalajara, Zapopan, Mexico
e-mail: eduardo.moya@jalisco.gob.mx

S. Xambó-Descamps
UPC-BarcelonaTech, Barcelona, Spain

Barcelona Supercomputing Center, Barcelona, Spain
e-mail: sebastia.xambo@upc.edu

S. Salazar Colores
Centro de investigación en Óptica, León, Mexico
e-mail: sebastiansalazar@cio.mx

A. Sánchez Pérez
Gobierno de Jalisco, Guadalajara, Mexico
e-mail: abraham.sanchez@jalisco.gob.mx

U. Cortés
UPC-BarcelonaTech, Barcelona, Spain

Barcelona Supercomputing Center, Barcelona, Spain
e-mail: ia@lsi.upc.edu

S. Xambó-Descamps (ed.), *Systems, Patterns and Data Engineering with Geometric Calculi*, SEMA SIMAI Springer Series 13, https://doi.org/10.1007/978-3-030-74486-1_7

Acronyms

CIFAR	Canadian Institute for Advanced Research
CNN	Convolutional Neural Network
DL	Deep Learning
FC	Fully Connected Layer
FL	Flatten Layer
FME	Facultat de Matemàtiques i Estadística
GA	Geometric Algebra
GC	Geometric Calculus
HSV	Hue-Saturation-Value (color standard)
MCNN	Monogenic CNN
ML	Machine Learning
MP	Max pooling layer
NN	Neural Network
RESNET	Residucal Network
RGB	Read-Green-Blue (color standard)
UPC	Universitat Politìcnica de Catalunya

1 Introduction

Many authors argue that currently, no suitable theory of CNNs design is available [1, 2]. Although some evidence supports that the depth (number of layers) [3], and the data augmentation during the training process [4], can occasionally provide invariance or equivariance relative to some class of transformations, the reasons for that behaviour do not seem to be well understood. Some investigations indicate that the learning of an invariance response may fail even with very deep CNNs or by large data augmentations in the training [1].

To overcome these shortcomings one idea is to embrace suitable geometric methods, as in [5], where the main techniques are real algebraic varieties and methods of computer algebra, and in [1, 2, 6–10], in which methods of differential geometry are used. In this regard, another strategy to progress in "Geometric Deep Learning" is to use Geometric Calculus (GC) in the sense of [11–14]. The main strong points for this advance are the long history of achievements in a great variety of fields (see [12], §6.4, and the references therein); that it includes the complex numbers and the quaternions as exceptional cases; and the fact that there is a well-developed theory of GC wavelets with the potential to be applied to DL much as scalar wavelets are used in current DL techniques [15–17]. An additional bonus of GC is that the representation of the signals occurs in a higher-dimensional space, and hence they provide a more robust discrimination capacity naturally.

In this work, as the first step in this general strategy, we work with Hamilton's quaternions $\mathbf{H}$, which is the most straightforward geometric calculus beyond the

complex numbers **C** (see Appendix A). The main results are the design and implementation of a CNN layer (M6) based on the *monogenic signal* proposed by Feslberg et al. [18]. These layers substantially enhance the invariance response to illumination-contrast.

Up till now, quaternions have been used with fully connected NNs [19–22], and, more recently, with CNNs [23–26]. In this context, the method proposed in this paper is the first, to the best of our knowledge, that combines a CNN with local phase computations using quaternions.

On the experimental side, to evaluate the predictive performance of M6, we have simulated illumination-contrast changes using the atmospheric degradation model (see Appendix B) over two image-datasets, the CIFAR-10 [27] and the Dogs and Cats [28].

The rest of the paper is organised into four additional sections and two appendices. The core of the paper consists of Sects. 2 and 3, which describe the new monogenic layer, M6, and the experimental setup, respectively. The experiments, results and analyses are presented in Sect. 4. Our conclusions are drawn in Sect. 5. Finally, in appendixes A and B, we summarise what we need about the quaternion field **H** and about the atmospheric scattering model of light, respectively.

2 Monogenic Convolution Neural Network Layer

What we call Monogenic Convolutional Neural Network (MCNN) is the coupling of a deterministic layer based on the monogenic signal, which we call *monogenic layer* (M6), with a conventional convolutional neural network (CNN). What we accomplish in this way, as shown by the experiments in subsequent sections, is a system for classifying images that not only outperforms the usual CNNs in speed but which is also resilient in front of severe changes in contrast.

2.1 Monogenic Signal

The terminology we are going to use is as follows (cf. Felsberg et al. [18]). We define 1D (resp. 2D) *multivectorial signals* as C^1 maps $U \to \mathcal{G}$ from an interval $U \subset \mathbf{R}$ (a region $U \subset \mathbf{R}^2$) into a *geometric algebra* $\mathcal{G}$ (see [12]). For $\mathcal{G} = \mathbf{R}$ ($\mathcal{G} = \mathbf{C}$, $\mathcal{G} = \mathbf{H}$) we say that the signal is *scalar* (*complex, quaternionic*). For technical reasons we also assume that signals are in L^2 (that is, the modulus is square-integrable).

The *Riesz-Felsberg transform* (RF) maps 2D scalar signals to 2D quaternionic signals. Among the signals obtained in this way, our interest lies in the (quaternionic) *monogenic signals* (see [18] for details). Some applications of the monogenic signal are: visual perception measurements [29, 30], local feature detection such as lines (even-signal) and edges (odd-signal) [14, 31], estimation of the disparity of stereo

images and blending of images [32], or computation of fast phase-based video magnification [33].

The monogenic signal $I_M = I_M(x, y) \in \mathbf{H}$ associated to an image[1] $I = I(x, y) \in \mathbf{R}$ (where $x, y \in U$, U a region of $\mathbf{R}^2$). The definition of I_M is as follows (cf. [18]):

$$I_M = I + I_{M'}, \quad I_{M'} = \boldsymbol{i} I_1 + \boldsymbol{j} I_2, \tag{1}$$

where, denoting by $*$ the convolutional product,

$$\begin{aligned} I_1 &= I * h_1, \quad I_2 = I * h_2, \\ h_1(x, y) &= -\tfrac{x}{2\pi(x^2+y^2)^{3/2}}, \\ h_2(x, y) &= -\tfrac{y}{2\pi(x^2+y^2)^{3/2}}. \end{aligned} \tag{2}$$

The signals I_1 and I_2 are the *Riesz transforms* (quadrature filters) of I in the x and y directions [18]. Note that $I_M \in \langle 1, \boldsymbol{i}, \boldsymbol{j} \rangle \subset \mathbf{H}$.

The *local amplitude signal* $|I_M|$ is defined by $|I_M|(x, y) = |I_M(x, y)|$, where the last expression is the modulus of the quaternion $I_M(x, y)$ [18]. Notice that we have

$$|I_M|^2 = |I|^2 + |I_{M'}|^2 = |I|^2 + |I_1|^2 + |I_2|^2, \tag{3}$$

where $|I|(x, y) = |I(x, y)|$ and similarly with $|I_1|$ and $|I_2|$.

2.2 Monogenic Filter Bank

In practice the monogenic signal needs a bandpass filtering, in order to define the local region of the signal [14, 34]. For all the computations of the filtered version of the image I we have used a radial (isotropic) bandpass *Log-Gabor* function in the frequency domain $G(u_1, u_2)$) defined as follows:

$$G(u_1, u_2) = \exp\left(-\frac{\log\left(\frac{\sqrt{u_1^2+u_2^2}}{\omega_0} \right)^2}{2\log(\sigma)^2} \right), \tag{4}$$

where u_1, u_2 are frequency components, ω_0 is the central frequency of the filter, σ is a bandwidth parameter (see [35] for more details). The filtering process is described in the following steps:

[1] Here it is to be noted that I is not the source image we are interested in, but a filtered version of it in the sense explained in Sect. 3.

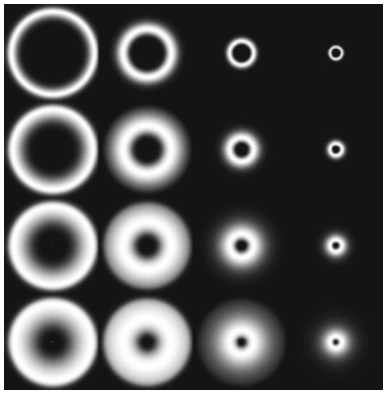


Fig. 1 **a** Log-Gabor filters at different scales and bandwidths in the frequency domain

1. Compute the 2D Fourier transform $\mathcal{F}(\bar{J})$ of the mean value image $\bar{J}$ as in [36], namely

$$\mathcal{F}(\bar{J})[u_1, u_2] = \sum_{m_1} \sum_{m_2} \bar{J}[m_1, m_2] e^{-i2\pi(u_1 m_1 + u_2 m_2)} \tag{5}$$

2. We have modified the P. Kovesi Python implementation [37], in order to compute a monogenic filter bank based on $G(u_1, u_2)$ with different scales. A filter bank can be computed with the following parameters $\omega_0^s = \frac{1}{\min_{wl} * s_f^{s-1}}$, where $\min_{wl}$ is the minimum wavelength, s_f is a scale factor, $s = 1, 2, \ldots, n_s$ is the current scale.

$$\bar{J}^s(u_1, u_2) = \bar{J}(u_1, u_2) \exp\left(- \frac{\log\left(\frac{\sqrt{u_1^2 + u_2^2}}{\omega_0^s} \right)^2}{2 \log(\sigma)^2} \right) \tag{6}$$

Figure 1 presents various views of the Log-Gabor function $G(u_1, u_2)$.

The *local phase* I_ϕ and the *local orientation* I_θ associated to I are defined, following [18], by the relations

$$I_\phi = \text{atan2}\left(\frac{I}{|I_R|} \right), \tag{7}$$

$$I_\theta = \text{atan}\left(\frac{-I_2}{I_1} \right), \tag{8}$$

where the quotients of signals are taken point-wise. For the geometric interpretation of these signals see Fig. 2.

The local phase can distinguish lines and edges [14, 20, 30, 31, 38], whereas the local orientation appears as a pinwheel picture resembling the behaviour of V1 simple cells and orientation columns.

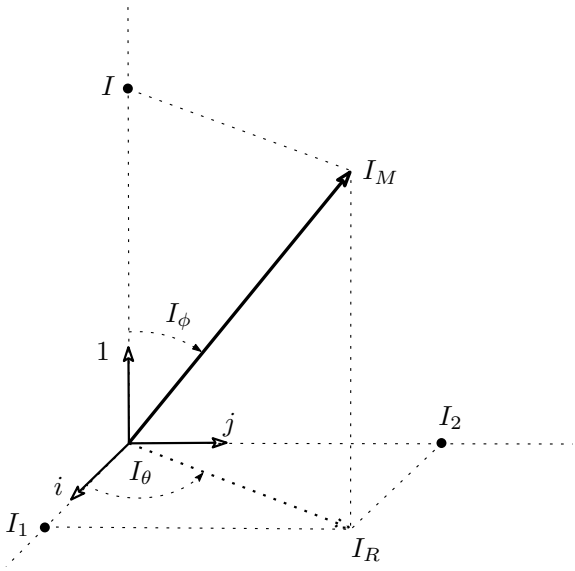


Fig. 2 Geometry of the monogenic signal

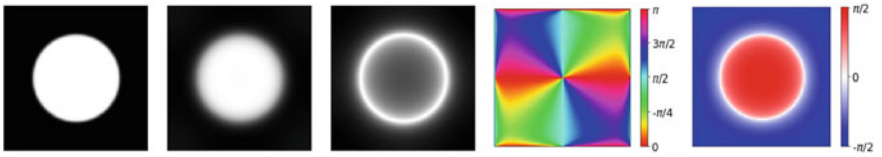


Fig. 3 Original image, filtered image, local energy I_M, local phase I_ϕ, and local orientation I_θ

Figure 3 illustrates the monogenic transform of the image of a white circle. From left to right, we display the original image, the filtered image (in the sense of Sect. 3), and the corresponding local magnitude, phase and orientation signals. The highest local energy values take place at the circle boundary, whereas the dominant values of the local orientation are $-\pi/2$ (blue), 0 (white), $\pi/2$ (red).

2.3 Description of the Monogenic Layer

We report on work about one monogenic layer, which we call M6. As we will see, both share promising features with the V1 layer in which they are inspired.

The purpose of M6 is to perform the following set of operations on the input image J. If J is formed by different channels indexed by $c = 0, 1, \ldots, N-1$, we denote by J_c the image corresponding to channel c. For instance, in an RGB (HSV) image, we would have the images I_R, I_G, and I_B (I_H, I_S, and I_V).

1. Get the average $\bar{J}$ of J over the channels forming J, that is, with the previous notations,

$$\bar{J} = \frac{1}{N} \sum_{c=0}^{N-1} J_c \tag{9}$$

2. Get the image I obtained by filtering $\bar{J}$ in the sense explained in Sect. 3.
3. Calculate the monogenic components $|I_M|$, I_ϕ, I_θ of I as defined by the Eqs. (3), (7), and (8), respectively.
4. Construct two HSV images as follows:

$$HSV_\phi = (H, S, V) = (I_\phi, |I_R|, 1), \tag{10}$$

$$HSV_\theta = (H, S, V) = (I_\theta, |I_R|, 1). \tag{11}$$

5. Transform the HSV images into RGB images RGB_ϕ and RGB_θ according to the standard conventions (see page 304 of [39])

The use of the colour space HSV has been handy for the encoding in different colour hues the values of the monogenic angular components. Moreover, the further transformation to the RGB colour space enhances the visibility of the regions in the original image in which the local amplitude is significant, which translates into a sensitivity to sharp edges.

The six components of $M6$, namely RGB_θ plus RGB_ψ, together with the 3 RGB components of the input signal J, are illustrated in Fig. 4.

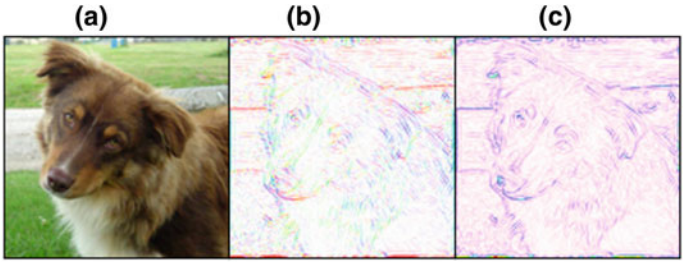


Fig. 4 **a** RGB input image. **b** RGB_θ. **c** RGB_ϕ. $M6$ is defined by **b** and **c**

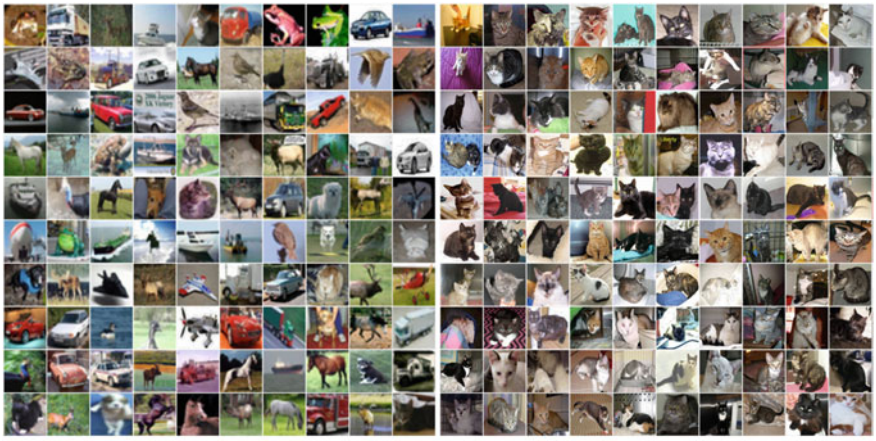
3 Experimental Setup

3.1 Datasets

We have used two datasets, CIFAR-10 [27] (Fig. 5(a)) and Dogs and Cats (Fig. 5(b)). Each data set was split into three sets: training set, validation set and test set. Table 1 shows their main characteristics.

Table 1 Characteristics of the CIFAR-10 and Dogs and Cats datasets

	CIFAR-10	Cats vs Dogs
Training set	36,000	2,400
Validation set	12,000	800
Test set	12,000	800
Total	60,000	4,000
Input shape	[32 × 32 × 3]	[150 × 150 × 3]



(a) (b)

Fig. 5 Examples of: **a** 100 RGB images from the CIFAR-10 dataset; **b** 100 images from the Dogs and Cats dataset

3.2 *Degrading Procedures*

In our experiments, any of the original images is degraded by the addition of fog according to the McCartney atmospheric scattering model [40] (summarized in Appendix B) and further modified by the addition of independent random changes in the illumination colour (in the range 0.8–1.0 for each colour component) of atmospheric light $A(r, g, b)$. In addition, the transmission map $t(x, y)$ was computed with the random parameters summarized in the Table 2.

An account of the details is deferred to Appendix B, while Fig. 6 provides an illustration of a degradation run of the images in Fig. 5. For concreteness, we consider three degraded levels (d_1, d_2, d_3) for each dataset; for an illustration, see Fig. 7

Table 2 Degradation parameters of colour of atmospheric light $A(r, g, b)$ and the transmission map $t(x, y)$

Degradation levels	
$A([0.8, 1], [0.8, 1], [0.8, 1])$	
Level	Parameters
d_0	zero degradation
d_1	$t(x, y) = [0.5, 0.8]$
d_2	$t(x, y) = [0.3, 0.5]$
d_3	$t(x, y) = [0.0, 0.15]$

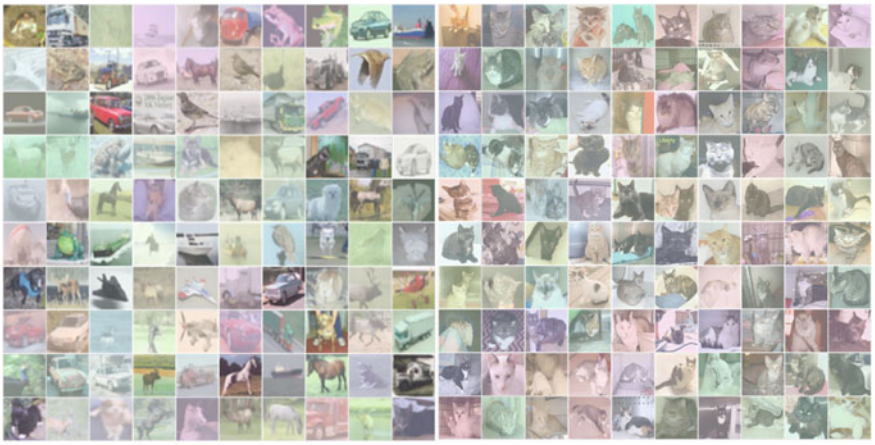


Fig. 6 Degraded versions (d_1) of the images in Fig. 5: **a** From CIFAR-10; **b** From Dogs and Cats

Fig. 7 **a** Original image, level d_0; **b** contrast level d_1; **c** contrast level d_2; **d** contrast level d_3. For the meaning of these labels see Subsect. 3.5

3.3 CNN and MCNN Architectures

Functionally, an NN layer takes an input $\mathbf{x}$ and produces an output $\mathbf{x}'$. The map $f : \mathbf{x} \mapsto \mathbf{x}'$ depends on parameters associated with the layer and whose nature depends on the kind of layer. In general, $\mathbf{x}$, $\mathbf{x}'$, and the layer parameters are multidimensional arrays whose nature is chosen according to the processing that has to be achieved.

Write $[n_1, n_2, \ldots, n_d]$ to denote the type of a d-dimensional (real) array with axis dimensions $n_1, \ldots, n_d$. Thus $[n]$ is the type of n-dimensional vectors and $[n_1, n_2]$, the type of matrices with n_1 rows and n_2 columns. Matrices are useful to represent monochrome images, but for RGB images we need arrays of type $[n_1, n_2, 3]$, or $[n_1, n_2, n_3]$ if it is required that the image be represented by n_3 *channels*, as for example $n_3 = 6$ for a pair of color stereoscopic images.

The parameters associated to convolutional and fully connected layers are represented by a *filter* array of weights, W,[2] and a *bias* array, $\mathbf{b}$. In these cases, the expression of f has the form

$$f^{\pi}(\mathbf{x}) = g(\mathbf{x} \star_{\pi} W + \mathbf{b}), \tag{12}$$

where $\star_{\pi}$ is a *pairing* specific of the layer and g is an activation function, ReLU in this paper, that is applied component-wise to arrays. For convolutional layers, $\star_{\pi} = \star$ is *array cross-correlation*, to be described below, while for fully connected layers, $\star_{\pi}$ is matrix product, which is denoted by juxtoposition of its factors, $\mathbf{x}W$. For a maximum pooling layer, the parameters are represented by a triple of positive integers $(w_1, w_2, s = 1)$, where (w_1, w_2) is the shape of the pooling window and s is the stride (1 by default). In this case $\star_{\pi} = \star_{\mathrm{mp}}$ is given by the rule

$$(\mathbf{x} \star_{\mathrm{mp}} W)[i, j, k] = \max(\mathbf{x}[is : is + w_1 - 1, js : js + w_2 - 1, k]), \tag{13}$$

where we use the standard slicing conventions for arrays. The shape of the array $\mathbf{x} \star_{\mathrm{mp}} W$ is $[n'_1, n'_2, n_3]$, where n'_1 and n'_2 are the greatest integers such that $n'_1 \leq (n_1 - w_1)/s$ and $n'_2 \leq (n_2 - w_2)/s$.

[2] Filters are also called *kernels*.

In the cross-correlation product $\mathbf{y} = \mathbf{x} \star W$, $\mathbf{x}$ is an array of type $[n_1, n_2, n_3]$ and W (the filter) is an array of type $[w_1, w_2, n_3, m_3]$. The pair (n_1, n_2) is the shape of the space dimensions of $\mathbf{x}$ and n_3 the number of channels. The pair (w_1, w_2) denotes the window dimensions of the filter and m_3 the number of channels of the array $\mathbf{y}$. The definition is given by the following formula:

$$\mathbf{y}[i, j, k] = \sum_{m=0}^{w_1-1} \sum_{n=0}^{w_2-1} \sum_{r=0}^{n_3-1} \mathbf{x}[i + m, j + n, r] W[m, n, r, k], \tag{14}$$

which can be expressed more compactly as

$$\mathbf{y}[i, j, k] = \sum_{r=0}^{n_3-1} \mathbf{x}[i : i + w_1 - 1, j : j + w_2 - 1, r] * W[:, :, r, k], \tag{15}$$

where $*$ denotes the ordinary scalar product of matrices. Notice that the shape of $\mathbf{y}$ is $[n_1 - w_1 + 1, n_2 - w_2 + 1, m_3]$.

There is also a *downsampled cross-correlation* $\mathbf{y} = \mathbf{x} \star_s W$ by a stride s:

$$\begin{aligned} \mathbf{y}[i, j, l] &= \sum_{k,m,n} \mathbf{x}[is + m, js + n, k] W[m, n, k, l] \\ &= \sum_{k} \mathbf{x}[is : is + w_1 - 1, js : js + w_2 - 1, k] \\ &\qquad * W[:, :, k, l]. \end{aligned} \tag{16}$$

Table 3 Flow of the monogenic CNN and of CNN-1 for the CIFAR-10 dataset. CNN-1 has 1,250,858 trainable parameters

CIFAR-10		
I	$[32, 32, n_3 = 3]$	
$M6$	$[32, 32, n_3 = 6]$	
$\mathbf{x} \to \mathbf{x}'$	W	$\mathbf{x}'$
C*-1	$[3, 3, n_3, 32]$	[32, 32, 32]
C-2	[3, 3, 32, 32]	[30, 30, 32]
MP-1	$[2, 2, s = 2]$	[15, 15, 32]
C*-3	[3, 3, 32, 64]	[15, 15, 64], Dropout (0.25)
C-4	[3, 3, 32, 64]	[13, 13, 64]
MP-2	$[2, 2, s = 2]$	[6, 6, 64] Dropout (0.25)
FL		[2304]
FC-1	[2304, 512]	[512]
FC-2	[512, 10]	[10] Dropout (0.5)
SMAX		[10]

Table 4 Flow of the monogenic CNN, and of CNN-2, for Cats and Dogs dataset. CNN-2 with 2,797,730 trainable parameters

Cats and Dogs		
I	[224, 224, $n_3 = 3$]	
$M6$	[224, 224, $n_3 = 6$]	
$\mathbf{x} \to \mathbf{x}'$	W	$\mathbf{x}'$
C*-1	[3, 3, n_3, 32]	[224, 224, 32]
MP-1	[2, 2, $s = 2$]	[112, 112, 32]
C-2	[3, 3, 32, 32]	[110, 110, 32]
MP-2	[2, 2, $s = 2$]	[55, 55, 32]
C-2	[3, 3, 32, 64]	[53, 53, 64]
MP-2	[2, 2, $s = 2$]	[26, 26, 64]
FL		[43264]
FC-1	[438264, 64]	[64] Dropout (0.5)
SMAX		[2]

The shape of the array $\mathbf{x} \star_s W$ is $[n'_1, n'_2, n_3]$, where n'_1 and n'_2 are the greatest integers such that $n'_1 \leq (n_1 - w_1)/s$ and $n'_2 \leq (n_2 - w_2)/s$.

In this work, we have used two architectures: CNN-1 (for CIFAR-10) and CNN-2 (for Dogs and Cats). See Fig. 8 for a schematic representation of M6CNN-1, which consists of CNN-1 with the M6 layer (M6CNN-2 is defined similarly). The computation flux of these networks is summarized in Table 3 for CIFAR-10 and in Table 4 for Cats and Dogs. In these two tables, the input I is processed by M6, and the resulting output is processed by a sequence of convolutional (C),[3] max-pooling (MP), flatten (FL), fully connected (FC), and softmax (SMAX) layers. If the monogenic step is omitted, the flow agrees with fairly standard CNNs (here called CNN-1 and CNN-2; see below for further details). The value n_3 is equal to 3 for I and 6 for M6, respectively. The W column specifies the filter of the current step. It is to be understood that the action of the layers C and FC is completed with a ReLU activation function.

Initially, we tested our monogenic layer on top of two CNNs architectures, CNN-1 and CNN-2, both with nine hidden layers. These testings aimed to carry out a relatively fast search of adequate hyper-parameter values that guarantee a classification accuracy close to a baseline mark. Additionally, we have tested our layer on top of a well-known RESNET-20 v2 architecture (with 571,034 trainable parameters) [41], with 18 hidden layers, to ascertain that it also produced gains similar to those observed with the simpler architectures.

[3] C* is a convolution with zero padding.

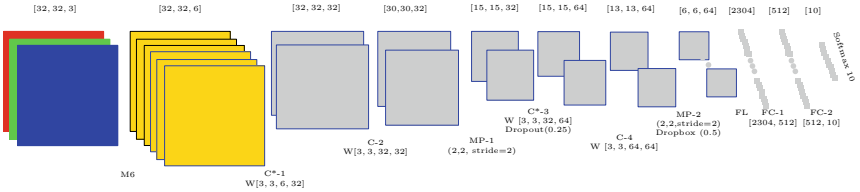


Fig. 8 CNN-1 architecture with 1,250,858 trainable parameters including the monogenic layer, four convolutional layers, 2 dropout, two max polling layers, three fully connected and one softmax function

Table 5 Experimental arrangements for each CNN

CNN	
Trained	Tested
d_0	d_0, d_1, d_2, d_3
d_1	d_0, d_1, d_2, d_3
d_2	d_0, d_1, d_2, d_3
d_3	d_0, d_1, d_2, d_3

3.4 Monogenic Hyper Parameters

To find the best parameters of the Monogenic layer, we evaluated the validation accuracy of CNN-1 on the CIFAR-10 dataset up to 100 epochs, with scales $s = [3, 4, 5]$, minimum wavelength $\min_w = [3, 4, 5]$ (i.e. smallest scale filter), scaling factors $sf = [1.1, 1.2...2.1]$, and standard deviations $\sigma = [0.3, 0.4, 0.48, 0.6]$. Finally we tried the learning rate values $lr = [0.0001, 0.001, 0.005]$. Altogether this amounts to 1584 combinations. The outcome was that the best parameters are $lr = 0.0001$, $s = 1$, $\min_w = 3$, $\sigma = 0.25$, $sf = 1.1$, for a maximum of test accuracy and minimum processing time.

3.5 Experiments

We trained and tested six nets: CNN-1, M6CNN-1, CNN-2, M6CNN-2, RESNET-20 v2, M6-RESNET-20 v2 [41], following the scheme summarized in Table 5, where d_i ($i = 0, 1, 2, 3$) means a degradation degree (see Fig. 7 for an intuitive view of the significance of these values). The computational codes are available at https://github.com/asp1420/monogenic-cnn-illumination-contrast.

In order to test each of all trained models about their generalization capacity, they were run not only on the original test set but also on the three modified versions of it consisting in adding the same three levels of haze.

All the experiments were carried out for 100 epochs, and learning rate of $lr = 0.0001$, no data augmentation, on the CTE-power 9 cluster of the Barcelona Supercomputing Center with one Tesla V-100.

4 Results and Analysis

A synopsis of experimental results for CNN-1 and CNN-2 is reported in Fig. 9 for CIFAR-10 a Fig. 10 for Cats and Dogs. Similarly, Fig. 11 summarizes the findings for RESNET-20 in the case of CIFAR-10.

The general idea of the experimental design has been to train the nets under four different degradation levels. These trained systems are represented on the horizontal axes and labelled by the degradation labels d_j ($j = 0, 1, 2, 3$). Each d_j is run on a set of images not seen before and also presented in four degradation levels d_k ($k = 0, 1, 2, 3$). The observed accuracies are represented by coloured circles in the case of the basic net and by coloured squares in the case of the corresponding monogenic-enhanced net. Thus each of the three graphics quotes 32 accuracies, 16 for the basic net and 16 for the monogenic net.

The main finding is the resilience of each of the monogenic systems d_j with respect to any of the degradations d_k, for the squares above d_j are clustered around 0.70 accuracy for all d_k. This contrasts with the wide dispersion of the circles above d_j, with a (to be expected) maximum when $k = j$ and substantially lower values for $k \neq j$. To note, however, that the basic nets d_j perform slightly better just for the degradation d_j, as shown by the top position of several of the corresponding circles (in Fig. 9, for instance, blue circle for $d_0 - d_0$, green for $d_1 - d_1$, and magenta for $d_2 - d_2$).

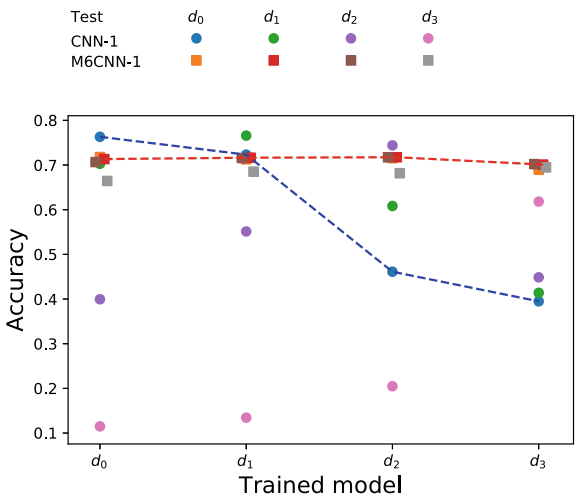


Fig. 9 CIFAR-10 test data with different degradation using CNN1 models. See text for details

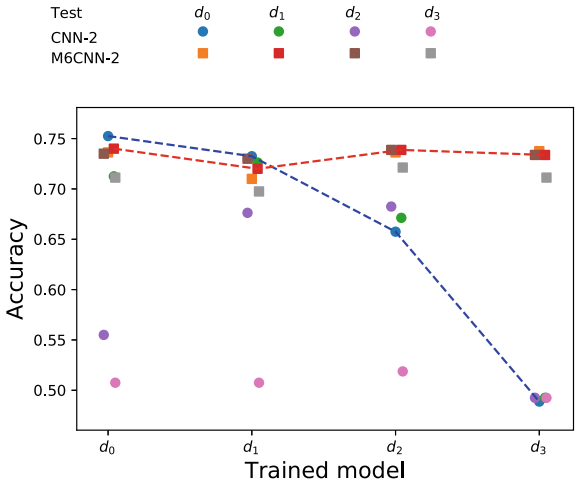


Fig. 10 Dogs and Cats test data with different degradation models using CNN-2. See text for details

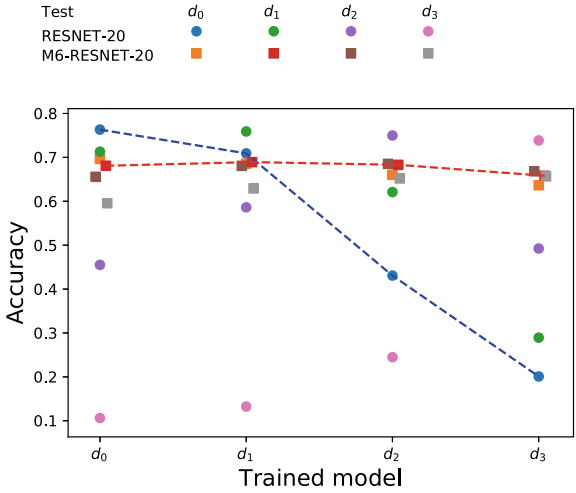


Fig. 11 CIFAR-10 test data with different degradation models using RESNET-20. See text for details

Our experiments results are consistent with the theoretical invariance of the phase-based feature detection to brightness and contrast transformations reported at [38, 42]. The M6 activation maps presented at Fig. 12 strengthened our confidence in that the M6 is invariant to these transforms, due to the activation maps is visually the same even with different levels of degradation.

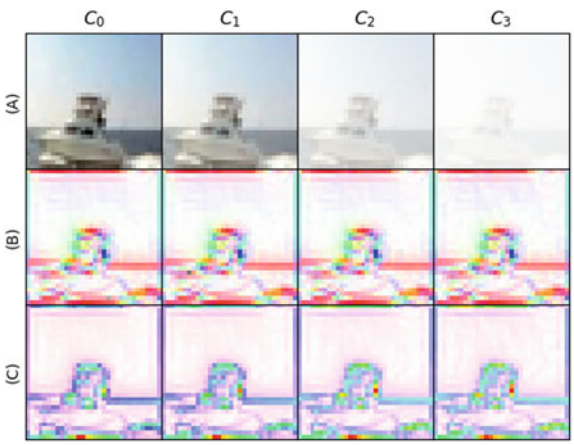


Fig. 12 Activation maps of M6. Row **A** present a one CIFAR-10 image with different degradation levels. Row **B** presents the activation map (local orientation) with different image degradation. Row **C** shows the local phase activation map with different input images

5 Conclusions

The context of this paper is the idea that geometric calculus has the potential to articulate novel and promising researches in deep learning.

In the explorations reported in this paper, we have used the quaternion calculus, which is the most straightforward geometric calculus beyond the complex calculus. More specifically, we have designed a front layer for CNNs that processes a monogenic signal by extracting phase and orientation signals and assembling them in an HSV space.

The experimental results with two different datasets and three CNNs confirm that the accuracy gained by using our layer has a substantially more robust performance when faced with severe illumination changes than the same nets without such a front layer.

We plan to continue the trail walked in this research by developing a front layer for CNNS that is resilient when faced with other transformations of the images, like rotations or even small deformations.

Acknowledgments The authors would like to thank to CONACYT and Barcelona supercomputing Center. Sebastián Salazar-Colores (CVU 477758) would like to thank CONACYT (Consejo Nacional de Ciencia y Tecnología) for the financial support of his PhD studies under Scholarship 285651. Ulises Moya and Ulises Cortés are member of the Sistema Nacional de Investigadores CONACyT.

Appendices

A. Quaternion Algebra

The quaternion algebra $\mathbf{H}$ is a four dimensional real vector space with basis $1, \boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$,

$$\mathbf{H} = \mathbf{R}1 \oplus \mathbf{R}\boldsymbol{i} \oplus \mathbf{R}\boldsymbol{j} \oplus \mathbf{R}\boldsymbol{k} \tag{17}$$

endowed with the bilinear product (multiplication) defined by Hamilton's relations, namely

$$\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{i}\,\boldsymbol{j}\boldsymbol{k} = -1. \tag{18}$$

As it is easily seen, these relations imply that

$$\boldsymbol{i}\,\boldsymbol{j} = -\boldsymbol{j}\boldsymbol{i} = \boldsymbol{k}, \quad \boldsymbol{j}\boldsymbol{k} = -\boldsymbol{k}\boldsymbol{j} = \boldsymbol{i}, \quad \boldsymbol{k}\boldsymbol{i} = -\boldsymbol{i}\boldsymbol{k} = \boldsymbol{j}. \tag{19}$$

The elements of $\mathbf{H}$ are named *quaternions*, and $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$, *quaternionic units*. By definition, a quaternion q can be written in a unique way in the form

$$q = a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}, \quad a, b, c, d \in \mathbf{R}. \tag{20}$$

Its *conjugate*, $\bar{q}$, is defined as

$$\bar{q} = a - (b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}). \tag{21}$$

Note that $(q + \bar{q})/2 = a$, which is called the *real part* or *scalar part* of q, and $(q - \bar{q})/2 = q - a = b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}$, the *vector part* of q.

Since the conjugates of $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ are $-\boldsymbol{i}, -\boldsymbol{j}, -\boldsymbol{k}$, the relations (18) and (19) imply that the conjugation is an *antiautomorphism* of $\mathbf{H}$, which means that it is a linear automorphism such that $\overline{qq'} = \bar{q}'\bar{q}$.

Using Hamilton's relations again, we easily conclude that

$$q\bar{q} = a^2 + b^2 + c^2 + d^2. \tag{22}$$

This allows to define the *modulus* of q, $|q|$, as the unique non-negative real number such that

$$|q|^2 = q\bar{q}. \tag{23}$$

Observe that $|qq'| = |q||q'|$. Indeed, $|qq'|^2 = qq'\overline{qq'} = qq'\bar{q}'\bar{q} = q|q'|^2\bar{q} = |q|^2|q'|^2$.

Finally, for $q \neq 0$, $|q| > 0$ and $q(\bar{q}/|q|^2) = 1$, which shows that any non-zero quaternion has an inverse and therefore that $\mathbf{H}$ is a (skew) field.

B. The Atmospheric Scattering Model

We have used the atmospheric scattering model in order to model the illumination and contrast degradation of the images. The formation of a degraded image is modeling using the atmospheric scattering model proposed by McCartney et al. [40], defined as follows:

$$I(x, y) = J(x, y)t(x, y) + A(1 - t(x, y)), \tag{24}$$

where $I(x, y)$ is the measured image, $J(x, y)$ is the original scene without affectations, $A(r, g, b)$ is the illumination colour of atmospheric light, and $t(x, y)$ is named as transmission map, which can be defined in a homogeneous atmosphere as:

$$t(x, y) = e^{-\beta d(x,y)}, \tag{25}$$

where β is the scattering coefficient of the atmosphere and $d(x, y)$ is the scene depth. An example of the estimated transmission map is presented in Fig. 13.

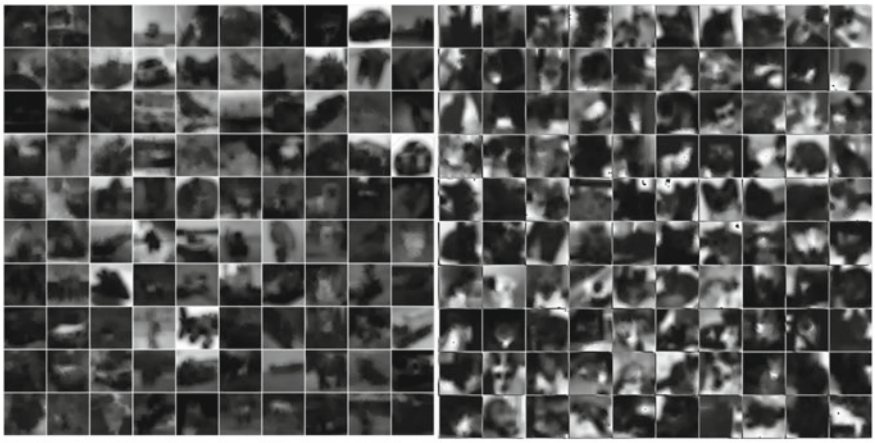


Fig. 13 Transmission map estimation. **a** Transmission map over 100 images from CIFAR-10; **b** Transmission map over 100 images from Dogs and Cats dataset

References

1. Cohen, T., Welling, M.: Group equivariant convolutional networks. In: International Conference on Machine Learning, pp. 2990–2999 (2016)
2. Cohen, T., Geiger, M., Köhler, J., Welling, M.: Convolutional networks for spherical signals. arXiv preprint arXiv:1709.04893 (2017)
3. Anselmi, F., Leibo, J.Z., Rosasco, L., Mutch, J., Tacchetti, A., Poggio, T.: Unsupervised learning of invariant representations. Theor. Comput. Sci. **633**, 112–121 (2016)
4. Simard, P.Y., Steinkraus, D., Platt, J.C.: Best practices for convolutional neural networks applied to visual document analysis. In: International Conference on Document Analysis and Recognition (ICDAR), p. 958. IEEE Computer Society (2003)
5. Weinstein, M., Breiding, P., Sturmfels, B., Kališnik Verovšek, S.: Learning algebraic varieties from samples. Revista Matemática Complutense **31**, 545–593 (2018)
6. Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P.: Geometric deep learning: going beyond Euclidean data. IEEE Sig. Process. Mag. **34**(4), 18–42 (2017)
7. Mallat, S.: Understanding deep convolutional networks. Philos. Trans. Roy. Soc. A **374**(2065), 20150203 (2016)
8. Bruna, J., Mallat, S.: Invariant scattering convolution networks. IEEE Trans. Pattern Anal. Mach. Intell. **35**(8), 1872–1886 (2013)
9. Wang, X., Jin, X., Xu, G., Xu, X.: A multi-scale decomposition based haze removal algorithm. In: International Conference on Remote Sensing, Environment and Transportation Engineering (RSETE), Nanjing, China, vol. 2, pp. 1–4, June 2012
10. Sifre, L., Mallat, S.: Rotation, scaling and deformation invariant scattering for texture discrimination. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1233–1240 (2013)
11. Hestenes, D., Sobczyk, G.: Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics, vol. 5. Springer, Dordrecht (2012)
12. Xambó-Descamps, S.: Real Spinorial Groups–A Short Mathematical Introduction. SBMA/Springerbrief. Springer, Heidelberg (2018)
13. Lavor, C., Xambó-Descamps, S., Zaplana, I.: A Geometric Algebra Invitation to Space-Time Physics Robotics and Molecular Geometry. SBMA/Springerbrief. Springer, Heidelberg (2018)
14. Felsberg, M.: Low-Level Image Processing with the Structure Multivector, vol. 203. Inst. für Informatik und Praktische Mathematik (2002)
15. Mitrea, M.: Clifford Wavelets, Singular Integrals, and Hardy Spaces. Springer, Heidelberg (2006)
16. Chan, W.L., Choi, H., Baraniuk, R.: Quaternion wavelets for image analysis and processing. In: 2004 International Conference on Image Processing, ICIP 2004, vol. 5, pp. 3057–3060, IEEE (2004)
17. Hitzer, E., Sangwine, S.J.: Quaternion and Clifford Fourier Transforms and Wavelets. Springer, Heidelberg (2013)
18. Felsberg, M., Sommer, G.: The monogenic signal. IEEE Trans. Sig. Process. **49**(12), 3136–3144 (2001)
19. Moya-Sánchez, E.U., Bayro-Corrochano, E.: Quaternion atomic function wavelet for applications in image processing. In: Iberoamerican Congress on Pattern Recognition, pp. 346–353. Springer (2010)
20. Bayro-Corrochano, E., Vazquez-Santacruz, E., Moya-Sanchez, E., Castillo-Muñis, E.: Geometric bioinspired networks for recognition of 2-D and 3-D low-level structures and transformations. IEEE Trans. Neural Netw. Learn. Syst. **27**(10), 2020–2034 (2016)
21. Isokawa, T., Matsui, N., Nishimura, H.: Quaternionic neural networks: fundamental properties and applications. In: Complex-Valued Neural Networks: Utilizing High-Dimensional Parameters, pp. 411–439. IGI Global (2009)
22. Buchholz, S., Sommer, G.: Quaternionic spinor MLP. In: ESANN 2000 Proceedings, D-Facto, European Symposium on Artificial Neural Networks, Bruges, Belgium, pp. 377–382, 26–28 April 2000 (2000)

23. Kominami, Y., Ogawa, H., Murase, K.: Convolutional neural networks with multi-valued neurons. In: 2017 International Joint Conference on Neural Networks (IJCNN), pp. 2673–2678. IEEE (2017)
24. Parcollet, T., Zhang, Y., Morchid, M., Trabelsi, C., Linarès, G., De Mori, R., Bengio, Y.: Quaternion convolutional neural networks for end-to-end automatic speech recognition. arXiv preprint arXiv:1806.07789 (2018)
25. Zhu, X., Xu, Y., Xu, H., Chen, C.: Quaternion convolutional neural networks. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 631–647 (2018)
26. Gaudet, C., Maida, A.: Deep quaternion networks. arXiv preprint arXiv:1712.04604 (2017)
27. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Technical report, Citeseer (2009)
28. Elson, J., Douceur, J.R., Howell, J., Saul, J.: Asirra: a CAPTCHA that exploits interest-aligned manual image categorization. In: ACM Conference on Computer and Communications Security, vol. 7, pp. 366–374 (2007)
29. Wang, Z., Simoncelli, E.P.: Local phase coherence and the perception of blur. In: Advances in Neural Information Processing Systems, pp. 1435–1442 (2004)
30. Sierra-Vázquez, V., Serrano-Pedraza, I.: Application of Riesz transforms to the isotropic AM-PM decomposition of geometrical-optical illusion images. J. Opt. Soc. Am. A **27**, 781–796 (2010)
31. Moya-Sánchez, E.U., Vázquez-Santacruz, E.: A geometric bio-inspired model for recognition of low-level structures. In: International Conference on Artificial Neural Networks, pp. 429–436. Springer (2011)
32. Tewari, A.: Image blending using local phase. Master of Science in Informatics at Grenoble, Université Joseph Fourier (2015)
33. Wadhwa, N., Rubinstein, M., Durand, F., Freeman, W.T.: Riesz pyramids for fast phase-based video magnification. In: 2014 IEEE International Conference on Computational Photography (ICCP), pp. 1–10. IEEE (2014)
34. Unser, M., Sage, D., Van De Ville, D.: Multiresolution monogenic signal analysis using the Riesz-Laplace wavelet transform. IEEE Trans. Image Process. **18**(11), 2402–2418 (2009)
35. Felsberg, M., Sommer, G.: A new extension of linear signal processing for estimating local properties and detecting features. In: Mustererkennung 2000, pp. 195–202. Springer (2000)
36. González, R.C., Woods, R.E., Eddins, S.L.: Digital Image Processing Using MATLAB, 2nd edn. Tata McGraw-Hill (2010). MATLAB examples
37. Kovesi, P.D.: MATLAB and Octave functions for computer vision and image processing (2018). https://www.peterkovesi.com/
38. Granlund, G.H., Knutsson, H.: Signal Processing for Computer Vision. Springer, Dordrecht (2013)
39. Agoston, M.K.: Computer Graphics and Geometric Modeling, vol. 1. Springer, London (2005)
40. McCartney, E.J., Hall, F.F.: Optics of the atmosphere: scattering by molecules and particles. Phys. Today **30**(5), 76–77 (1977)
41. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: European Conference on Computer Vision, pp. 630–645. Springer (2016)
42. Boukerroui, D., Noble, J.A., Brady, M.: On the choice of band-pass quadrature filters. J. Math. Imaging Vis. **21**(1–2), 53–80 (2004)

Geometric Calculi and Automatic Learning *An Outline*

Sebastià Xambó-Descamps and Eduardo Ulises Moya

Abstract Signal representation and processing are the backbone of mathematically-aided engineering. Among the myriad of ideas and results in that realm, many sorts of algorithms and techniques capable of learning from experience have taken the stage in the last decades, with a crescendo of great successes in a variety of fronts in recent years. This paper provides a sketchy outline of those developments that seem more relevant or promising in view of their bearing on the geometric calculus (multivector) representations of signals and the concomitant automatic learning algorithms. The corresponding artificial neurons, and their organization in networks, may be seen as a way to transcend the biologically inspired neuron networks much as the wheel or aviation transcended legs or bird flight. Recent developments suggest that there are exciting research opportunities ahead.

Prelude

The evolution of learning and reasoning, of understanding how they are achieved by minds, and the quest toward making them easier, and even mechanizing them, is centuries old, and for some aspects even millennia, [102]. Other valuable historical sources are [115] and, more recent, [135], which has the benefit of plentiful historical notes aptly inserted in an otherwise excellent treatise on many aspects of artificial intelligence as presently understood.

The wave is not frozen in those sources. It keeps its mighty rolling. Two noteworthy recent advances should suffice here to illustrate this relentless progression. One is about natural language processing (see the paper [24], and the chronicle [104]: "The

S. Xambó-Descamps (✉)
UPC, Departament de Matemàtiques, and BSC/CNS, Jordi Girona 1-3, Omega-428, Barcelona, Spain
e-mail: sebastia.xambo@upc.edu

E. U. Moya
Gobierno del Estado de Jalisco and Universidad Autónoma de Guadalajara, Guadalajara, Jalisco, Mexico
e-mail: eduardo.moya@jalisco.gob.mx

S. Xambó-Descamps (ed.), *Systems, Patterns and Data Engineering with Geometric Calculi*, SEMA SIMAI Springer Series 13, https://doi.org/10.1007/978-3-030-74486-1_8

latest natural-language system generates tweets, pens poetry, summarizes emails, answers trivia questions, translates languages and even writes its own computer programs"). The other is about an outstanding leap in the difficult problem of protein folding (see [36]: "AI makes gigantic leap in solving protein structures: DeepMind's program for determining the 3D shapes of proteins stands to transform biology, say scientists").

There are also books that try to look far ahead into the future, like [21, 49, 61, 130, 155, 163], but here, for our purposes, and no matter how alluring these and other works are, we feel bound to take a suitable measure of Turing's appraisal that "We can only see a short distance ahead, but we can see plenty there that needs to be done."

Thus we will first look into conventional automatic learning (Sect. 1) and neural networks (Sect. 2), illustrated by several examples, with the purpose of reviewing some fundamental concepts of automatic learning, and then we will consider (Sect. 3) the neural networks that process multivectors of a geometric algebra, with special emphasis on the complex and quaternionic cases. The last section (Sect. 4) advances a few prospective suggestions and the paper ends with the list of cited references.

1 Overview of Conventional Automatic Learning

The area in which we are primarily interested in is *automatic learning* (AL for short). It is a quest that has much in common with *machine learning* (ML), and with *deep learning* (DL) in particular, which in turn are subsidiary of *artificial intelligence* (AI). To a first approximation, AL seeks efficient algorithms that return, given a data set (experience), predictors of relevant information associated to new data.

In one form or another, the matters covered in this section can be found in books such as [47, 107, 168] (this includes a carefully chosen list of motivating examples, §1.3). See also [164] for a quite neat overview. When we speak of data, we refer to digitized forms of information, like audio signals or pictures. From a theoretical point of view, we may assume that a datum is a vector in some vector space.

Let us assume that data x are drawn from a space $\mathcal{X}$ (usually called *input space*) according to a probability distribution P, which we express symbolically by $x \sim P$. A *dataset* of length m is a sequence $\mathcal{D} = \{x^1, \dots, x^m\}$ such that $x^j \sim P$ independently, $j \in 1..m$, where $1..m = \{1, \dots, m\}$.

In *unsupervised learning* one of the major goals is to subdivide $\mathcal{D}$ into *clusters*, so that data in the same cluster are similar, according to some given criterion, while elements in different clusters are dissimilar. It is also required that any $x \in \mathcal{X}$ be assigned to one of the clusters.

k-Means. This is a typical algorithm, with many variants, for unsupervised learning from a dataset $\mathcal{D}$. For simplicity, we will assume that $\mathcal{X}$ is a convex set in some Euclidean space. In particular we have a distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ (so for all $x, x'x'' \in \mathcal{X}$ we have that $d(x, x') > 0$ if $x \neq x'$; $d(x, x) = 0$; $d(x, x') = d(x', x)$; and $d(x, x'') \leq d(x, x') + d(x', x'')$ (*triangular inequality*)). The algorithm

partitions $\mathcal{D}$ into k clusters, where k is predefined ($1 < k < m$), according to the following steps:

1) Choose k distinct elements $z^1, \ldots, z^k$ from $\mathcal{D}$ at random;
2) Assign each $x^j \in \mathcal{D}$ to the first z^i such that $d(x^j, z^i) = \min_{l \in 1..k} d(x^j, z^l)$, thus getting an initial grouping of $\mathcal{D}$ into k groups $Z_1, \ldots, Z_k$;
3) Replace each z^i by the centroid (barycenter) of Z_i;
4) Iterate until the z^i are stable according to some predefined tolerance.

At the end we have a partition of $\mathcal{D}$ into k clusters Z_i and the corresponding centroids z^i ($i \in 1..k$). Now each $x \in \mathcal{X}$ is assigned to the first Z_i such that $d(x, z^i) = \min_{l \in 1..k} d(x, z^l)$. This produces a partition of $\mathcal{X}$ into k clusters $X_1, \ldots, X_k$ such that $Z_i \subseteq X_i$.

A good reference for unsupervised learning is [78, Ch. 10]. That chapter also includes a presentation of Principal Component Analysis (PCA), which is a cornerstone of dimensionality reduction techniques. The PCA is closely related to the Singular Vector Decomposition (SVD), which is clearly presented in [151]. To mention also the treatise [79], with chapters on related topics, as a presentation of autoencoders and a few interesting applications. In any case, we agree with the appraisal in [133, end of §4.2.2] that "unsupervised learning is an extremely active area of research and one that has yet to be solved".

In *supervised learning* the problem is to come up with algorithms that produce a function $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y}$ is called the *output space*, with an acceptable capacity to predict the values of an unknown function $f^* : \mathcal{X} \to \mathcal{Y}$ (called the *expert* or *supervisor*) from a *labeled dataset*, or set of *examples*, $\mathcal{D}^* = \{(x^1, y^1), \ldots, (x^m, y^m)\}$, $y^j = f^*(x^j)$. When the output space is a finite set (of *classes*), the problem is said to be a *classification* problem. *Regression* or *interpolation* problems are those for which $\mathcal{Y} = \mathbb{R}$, or some other suitable continuous space. In this context, the learning algorithm is usually called the *learner*.

k-NN (nearest neighbors). This is an illustration of a classification algorithm. In this case the dataset has the form $\mathcal{D}^* = \{(x^j, y^j)\}$ ($j \in 1..m$), where $x^j \sim P$ and $\mathcal{Y}$ is a finite set. This algorithm partitions $\mathcal{D} = \{x^1, \ldots, x^m\}$, and $\mathcal{X}$, into $n = |\mathcal{Y}|$ clusters that are labeled by the $y \in \mathcal{Y}$. A much studied example is the case where $\mathcal{Y}$ is the set of decimal digits and the x^j are random images of handwritten digits (see [92, 114, 133] for detailed computational presentations).

With the same assumptions on $\mathcal{X}$ as for the k-Means, fix a positive integer k. Given $x \in \mathcal{X}$, the algorithm k-NN selects a label in $\mathcal{Y}$ according to the following rules:

1) Find $x^{j_1}, \ldots, x^{j_k}$ in $\{x^1, \ldots, x^m\}$ so that the k distances $d(x, x^{j_1}), \ldots, d(x, x^{j_k})$ are the smallest among the distances $d(x, x^j)$, $j \in 1..m$. To get $\{x^{j_1}, \ldots, x^{j_k}\}$ it is enough to make a list of the pairs $(j, d^j = d(x, x^j))$, sort them in non-decreasing order of the d^j, retain the first k pairs $(j_1, d^{j_1}), \ldots, (j_k, d^{j_k})$, and extract the indices $\{j_1, \ldots, j_k\}$.
2) Assign to x the mode of the set $\{y^{j_1}, \ldots, y^{j_k}\}$. If there is more than one element representing the mode, chose the first in the list.

For $k = 1$, for example, x is assigned to the first y^j such that $d(x, x^j) = \min_{k \in 1..m} d(x, x^k)$.

The k-NN algorithm can be modified so that k increases with m but with $k/m \to 0$, say $k = \log n$. For the analysis of this asymptotic version of the algorithm, particularly in the case $\mathcal{Y} = \{0, 1\}$ (binary classifiers) and its relation to the optimal classifier (called *Bayes' classifier*), we refer to [145, 150, 164]. It is also important to mention the contribution [152], which outlines a wondrous improvement of the k-NN algorithm capable of classifying n objects using a dataset of size $m < n$. This seemingly impossible task is achieved by replacing the standard 'hard' labels y^j by 'soft' labels, that is, by distributions of probability on n objects. This paper also provides references to the key foregoing works that paved the way for this discovery.

In supervised learning in general, we can deal with both cases (classification and regression) by setting $f(x) \simeq f^*(x)$ to indicate $f(x) = f^*(x)$ for classification and $f(x) \approx f^*(x)$ for regression, where $\approx$ indicates approximate equality according to some predefined criterion. With this convention, we can measure the goodness of f with the *learning rate*, that is, the proportion ℓ of cases for which $f(x^j) \simeq y^j$ ($j \in 1..m$), and its *accuracy*, also called *generalization rate* or *predicting capacity*, as the proportion a of cases for which $f(x) \simeq f^*(x)$ ($x \in \mathcal{X}$). Thus $1 - \ell$ is the *learning error rate* and $1 - a$ is the *generalization error rate*. Notice that ℓ depends only of f and $\mathcal{D}^*$, whereas a involves the whole input space, which is beyond the learner's reach. In practice a is replaced by the proportion of accurate predictions for a *testing* dataset independent of the *training dataset* $\mathcal{D}^*$. The study of the validity conditions of this approach belongs to the subject of *statistical learning*, for which here we can only provide standard references, like [20, 60, 78, 84, 161, 164], or general treatises on ML, like [5, 47, 62, 112, 145, 168].

Let us remark that achieving a learning rate $\ell = 1$ amounts to a perfect memorization of the examples $\mathcal{D}^*$, which usually entails a poor performance when faced with new examples, which means a low generalization rate. Therefore it is to be expected that the best use of the training examples $\mathcal{D}^*$ is to find a trade-off learning rate beyond which the generalization rate decreases. To be more precise, a useful setup is to assume that the functions available to the learner are a set $\mathcal{F} = \{f_w \mid w \in W\}$ of functions $f_w : \mathcal{X} \to \mathcal{Y}$ parameterized by the elements w is some parameter set W (set of *weights*), and that it has a measure $L(w) = L(f_w)$ (*loss functional*) of how close f_w is of f^*. Then the training can be construed as a search of $w \in W$ such that $L(w)$ is as small as possible. Let us consider a few examples.

Linear Regression. Assume that the data are vectors $x \in \mathbb{R}^n$ and that the values y are real numbers. For any vector $x \in \mathbb{R}^n$, set

$$f_w(x) = w \cdot x = w_1 x_1 + \cdots + x_n x_n$$

(a *weighted sum* of the entries $x_1, \ldots, x_n$ of x). If we take $L(w) = \sum_{j=1}^n (w \cdot x^j - y^j)$ (*square loss*) as a measure of how well does f_w fit the dataset (notice that $L(w) = 0$ is equivalent to the perfect fitting $f_w(x^j) = y^j$ for all j), then the goal is to find $\operatorname{argmin}_w L(w)$ (the w that minimizes $L(w)$). In particular the solution must satisfy the

condition $\nabla_w L(w) = 0$ (or $\partial L/\partial w_k = 0, k \in 1..n$), which turns out to be equivalent to the relation $wX^TX = yX$, where X denotes the $m \times n$ matrix whose rows are the vectors x^j and $y = (y^1, \ldots, y^m)$. This relation is most useful when X^TX is invertible, a condition that requires $n \leq m$, as in this case we get a unique solution w. If X^TX is singular, but $wX^TX = yX$ still has a solution, then it is possible to find w of minimum norm. Otherwise one can apply well known numerical methods to find a suitable w, as in [159, Lecture 11]. Notice that a loss of the form $L(w) = w_0 + w_1x_1 + \cdots + w_nx_n$ can be treated with the same methods by augmenting each vector x with a 0-th component $x_0 = 1$. The method of *least squares* (as is ancestrally called) can be applied to determine objects in some space that approximate a cloud of (noisy) points in the same space provided that those objects can be described as vectors in a higher dimensional space. For example, conics in a plane can be described by the vector of their coefficients, and this allows to apply least squares to find the conic that best fits a cloud of points (cf. [80], on optimization techniques for geometric estimation). The method can also be extended to find curves that best fit a cloud of points. In the case of spline curves, the control points are regarded as the weights to be determined (see [52, 59, 96, 97, 99, 140, 167], and the forthcoming [137], where curves other than splines are also explored).

A special case of linear regression is the *logistic regression*, which is suitable when the log of the *odds* $P(x)/(1 - P(x))$ of drawing $x \in \mathcal{X}$ can be predicted by a linear regression, say

$$\log \frac{P(x)}{1 - P(x)} = w_1x_1 + \ldots + w_dx_d = w \cdot x.$$

Once the weights $w_1, \ldots, w_n$ are ascertained by means of a dataset, then we get the *logistic predictor*

$$P(x) = \sigma(w \cdot x) = \frac{1}{1 + e^{-w \cdot x}},$$

where $\sigma(t) = 1/(1 + e^{-t})$ is the *logistic function*. It has a sigmoid form and ranges between 0 for $t = -\infty$ and 1 for $t = \infty$. Occasionally it may be handy to have a sigmoid ranging from $-k$ to k, k a positive constant, like for instance $2k\sigma(t) - k = k(1 - e^{-t})/(1 + e^{-t})$.

Support Vector Techniques for Classification. These techniques, usually known as *support vector machines* (SVM), were introduced by Vapnik and his school in 1992 (see [43], no doubt a classic of AL, where they were actually called "Support-Vector Networks"). An early application was the recognition of hand-written digits with an accuracy not less than the best systems available at the time. Excellent treatments of this topic can be found in [107, Ch. 5] and [5, Ch. 14].

Assume that $\mathcal{X} = \mathbb{R}^n$. Of a dataset of the form $\mathcal{D} = \{(x^1, y^1), \ldots, (x^m, y^m)\}$, with $x^j \in \mathbb{R}^n$ and $y^j \in \{-1, 1\}$, we say that it is *linearly separable* if there is a hyperplane $h(x) = w \cdot x + b$ ($w \in \mathbb{R}^n$, $b \in \mathbb{R}$)) such that $h(x^j) > 0$ or $h(x^j) < 0$ according to whether $y^j = +1$ or $y^j = -1$, that is, if $y^j h(x^j) > 0$ for all $j \in 1..m$. Geometrically, the points with $y^j = 1$ lie on the positive half-space defined by h

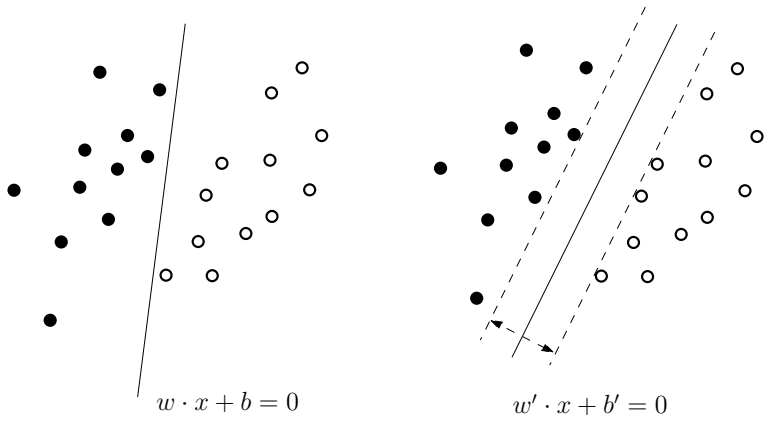


Fig. 1 Separation by hyperplanes and support vectors. On the left, the white points (+1) and the black points (−1) are linearly separable. On the right we see the same set of points and the greatest margin separator, which is computed by the SVM algorithm described in the text

and the points with $y^j = -1$ lie on the negative half-space (see Fig. 1). In general, there are (if any) infinitely many separating hyperplanes, so that we can envisage to impose additional constraints, like the condition that the points on either side are as far as possible from the hyperplane. This idea leads to the notion of *margin* and the appearance of *support vectors* as described next.

Given a separating hyperplane $h(x) = w \cdot x + b$, let $s = \min_{j\in 1..m} |w \cdot x^j + b| > 0$. On dividing h by s, an artifice that does not change the hyperplane, we may assume the *normalization condition* $\min_{j\in 1..m} |w \cdot x^j + b| = 1$, and in this case $1/|w|$ is the distance of any x^j at minimum distance from the hyperplane, for this distance is $|w \cdot x^j + b|/|w|$. The quantity $1/|w|$ is the *margin* of the hyperplane and the x^j at a margin distance are its *support vectors*. It is therefore clear that to maximize the margin of h is equivalent to minimize $|w|$, or, more conveniently, $\frac{1}{2}|w|^2$, whose gradient and hessian are w and the identity, respectively. This shows that the problem is equivalent to minimizing $|w|$ under the constraints $y^j(w \cdot x^j + b) \geq 1$, which in turn can be solved by the technique of Lagrange multipliers. Among the properties of the solution, obtained by means quadratic optimization techniques (see [107, Ch. 5]), let us highlight that the support vectors lie on the hyperplanes $w \cdot x + b = \pm 1$ (called *marginal planes*) and that w turns out to be an explicit linear combination of the support vectors. Moreover, the solution only depends on the inner products to the data vectors (which form their *Gram matrix*).

The technique can be adapted to more general classification problems. One is the classification of data that are not linearly separable. Not being able to assume that $y^j(w \cdot x^j + b) \geq 1$ for all j, introduce non-negative *deviation variables* $t_1, \ldots, t_n$ and consider the relaxed constraints $y^j(w \cdot x^j + b) \geq 1 - t_j$. In this situation, a convenient modification of the function to be minimized is $\frac{1}{2}|w|^2 + \lambda \sum t_j$. The hyperplane produced with this minimization separates correctly the x^j with margin

$1/|w|$ except the *outliers*, which means that fall either on the incorrect half-space or within the ribbon $-1 < w \cdot x^j + b < +1$ (see, for example, [78, Ch. 9]). Another extension is to multi-class classifications (*ibidem*).

Finally, let us mention the very useful device consisting of applying linear separation after mapping the input space to a higher dimension by means of a non-linear map. Roughly, it works as follows. A characteristic (or *feature map*) of the space $\mathcal{X}$ is a map $\psi : \mathcal{X} \to \mathbb{R}^{n'}$, where n' can be arbitrary. Usually n' and ψ are chosen to facilitate that the data $\psi(x)$ appear to be linearly separable in $\mathbb{R}^{n'}$ when this condition is not satisfied in $\mathcal{X}$. Actually, if we manage to obtain a linear separator h' of the $\psi(x^i) \in \mathbb{R}^{n'}$, then $h(x) = h'(\psi(x))$ is a non-linear separator of the x^i in $\mathbb{R}^n$ and the hypersurface $\{h(x) = 0\}$ is the *decision boundary*. A relevant point is that these techniques lead naturally to the notion of *kernels* (see, for example, [5, Ch. 14]), which rely on the pairing $K(x, x') = \psi(x) \cdot \psi(x')$ or, more specifically, on the *kernel matrix* $K(x^i, x^j)$, which is sufficient, as remarked before, to run the support vector algorithms (in $\mathbb{R}^{n'}$), and the *kernel trick* amounts to the realization that often the values $K(x^i, x^j)$ can be judiciously specified with no reference to ψ (see [5, §14.6] for examples, and in particular for the *polynomial* and *radial* kernels). One more point is that there are also interesting cases in which $n' < n$, and then we speak of *dimension reduction*. As noted, the PCA and SVD mentioned earlier fall under this notion. A quite interesting achievement is the t-SNE separation algorithm developed in [100] and [160] mapping images of hand-written digits (dimension $n = 28^2$) to $\mathbb{R}^2$. Further references: [35, 139, 146], [45, Chapters 9 and 10], [2, 84, 145].

2 Conventional Neural Networks (NN)

In AL, a useful model of a *neuron* (see Fig. 2) is a function of the form

$$x \mapsto f_w(x) = \sigma(x \cdot w), \tag{1}$$

where $w \in \mathbb{R}^n$ (*weights* or *parameters*) and σ is a *sigmoid* function (called *activation function*), like for instance the *logistic function* $\sigma(t) = (1 + e^{-t})^{-1}$, in which case the neuron computes a logistic regression. Augmenting x with $x_0 = 1$ and providing an extra weight w_0 (called the *bias*), the neuron computes $\sigma(w_0 + w_1 x_1 + \cdots + x_n w_n)$. If we want to display separately the bias and the other weights, we will write f_{w,w_0} or some similar notation.

A *neural network* (NN) can be construed as a *composition of neurons* according to a graph of connections called the *architecture* of the net. Here we will consider the case of directed graphs and thus leaving aside nets based on undirected graphs such as those of Hopfield networks and Boltzmann machines. Nor will we discuss networks with feedback (those having closed paths).

The standard architecture of a NN is a directed graph structured in *layers* L_j, as illustrated in Fig. 3, and its functional signature can be condensed as a chain:

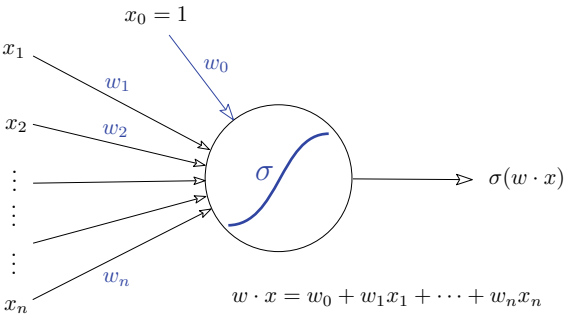


Fig. 2 Scheme of a neuron. The neuron's output depends on the weights w and on σ (*activation function*), and this functionality is represented by the decorated circle

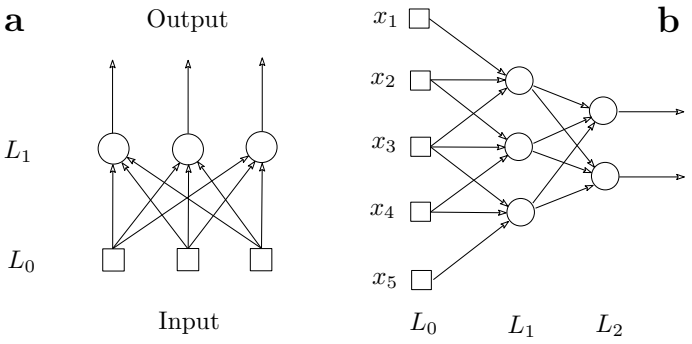


Fig. 3 **a** Neural network with no hidden neurons and fully connected. **b** Network with a hidden layer L_1 of three neurons fully connected to the two output neurons of L_2. The input layer, L_0, is only partially connected to L_1. Since each hidden neuron receives the outputs of three input neurons, the weights of the three hidden neurons could share the same weights. This notion prefigures the convolutional neural networks (CNN, or ConvNets) described later. Notice that if the three shared weights are $w = (w_1, w_2, w_3)$, the inputs of the hidden neurons are $y_j = \sum_{i=1}^{3} w_i x_{i+j-1}$, $j = 1, 2, 3$, which we recognize as the cross correlation $x \star w$ of the input vector x with w

$$\mathcal{N}: \quad \text{Input} \to L_0 \xrightarrow{f_0} L_1 \xrightarrow{f_1} \cdots \to L_m \xrightarrow{f_m} L_{m+1} \to \text{Output} \tag{2}$$

Conventionally, the net is *shallow* if $m = 1$ and *deep* if $m > 1$. The layers $L_1, \ldots, L_m$ are considered to be *hidden*, while the input and output layers (L_0 and L_{m+1}) are *visible*.

Functionally, the layer L_j takes an input x and yields an output x'. The map $f_j : x \mapsto x'$ depends on the parameters associated to the layer neurons and its expression defines the kind of layer (the main kinds are introduced later). The input x^0 to L_0 is the signal to be processed (a sound or an image, for example). The output of L_{m+1} (*output layer*) is the transformation produced by the net on x^0. The role of L_0 is akin to the sensory organs of living beings. The output is the result of applying progressively the maps $f_0, f_1, \ldots, f_m$ (that is, the composition $f_m \circ f_{m-1} \circ \cdots \circ f_1 \circ f_0$) to the input. In terms of the biological analogy, the hidden layers are alike the brain structure,

and the output, to the signals sent by the brain to the various organs involved in the behavior of the being (like locomotion and phonation, for example).

The map $f_j : x \mapsto x'$ is parametrized by the set W_j of weights of the neural connections arriving at (neurons of) L_{j+1}, so that we can write $f_j = f_{W_j}$. Consequently, the map computed by the NN is parametrized by the set $W = \cup_j W_j$: $f_W = f_{W_m} \circ \cdots \circ f_{W_0}$. Since the activation functions of the neurons are non-linear, f_W is a highly non-linear map. The number of parameters is generally large or very large, the more so the deeper the net. At present the number of parameters of the largest NNs are approaching 10^{12}. In biological terms, these weights play the role of the synaptic potentials of the neocortex, but these still outnumber by more than two orders of magnitude the artificial connections.

NNs are universal approximators, in the sense that any continuous function can be approximated by a NN, and even by a fully connected shallow one with a single output neuron. The function computed by such a NN has the form

$$f(x) = w' \cdot x', \quad x'_i = \sigma(w^{(i)} \cdot x), \tag{3}$$

where $w' \in \mathbb{R}^{n_1+1}$ ($n_1 = |L_1|$) is the vector of output weights and $w^{(i)} \in \mathbb{R}^{n_0+1}$ ($n_0 = |L_0|$) is the vector of weights arriving at the i-th neuron of L_1. Here we assume that $x_0 = x'_0 = 1$, so that $w_0^{(i)}$ and w'_0 play the role of biases. The class of functions f given by (3) is sufficient to approximate an arbitrary continuous function to any degree of precision uniformly on compact sets (see [7, 10, 46, 74, 126]).

The Array Model. In general, x and x' in $x' = f_j(x)$, and the *layer parameters* W_j, b_j (weights and biases), are multidimensional arrays whose nature is chosen according to the processing that has to be achieved.

Write $[n_1, n_2, \ldots, n_d]$ to denote the type of a d-dimensional (real) array with axis dimensions $n_1, \ldots, n_d$. Thus $[n]$ is the type of n-dimensional vectors and $[n_1, n_2]$ the type of matrices with n_1 rows and n_2 columns. Matrices are useful to represent monochrome images, but for RGB images we need arrays of type $[n_1, n_2, 3]$, or $[n_1, n_2, n_3]$ if it is required that the image be represented by n_3 channels.

The parameters associated to *convolutional* and *fully connected* layers are represented by an *array of weights*, W, and a bias array, b. In these cases, the expression of f has the form

$$f^\pi(x) = g\,(x \star_\pi W + b) \tag{4}$$

where $\star_\pi$ is a pairing specific of the layer and g an activation function that is applied component-wise to arrays. Here it is to be remarked that instead of a sigmoid activation it has become practical to use a *rectified linear unit* (ReLU), defined as $\max(0, x)$ (cf. [5, §12.2.1]). Its advantages are that it is continuous and piecewise linear, that it is not bounded above, and that it works fine when its derivative is needed (the jump function $x \mapsto 0$ if $x \leq 0$, 1 if $x > 0$). For convolutional layers, $\star_\pi = \star$ is *array cross-correlation*, while for fully connected layers, $\star_\pi$ is *matrix product*, which is denoted by juxtoposition of its factors, xW.

In the cross-correlation product $y = x \star W$, x is an array of type $[n_1, n_2, n_3]$ and W (the *filter*) is an array of type $[w_1, w_2, n_3, m_3]$. The pair (n_1, n_2) is the shape of the geometric dimensions of x and n_3 the number of channels. The pair (w_1, w_2) denotes the window dimensions of the filter and m_3 the number of channels of the output array y. The definition is given by the following formula:

$$y[i, j, k] = \sum_{m=0}^{w_1-1} \sum_{n=0}^{w_2-1} \sum_{r=0}^{n_3-1} x[i+m, j+n, r] W[m, n, r, k] \tag{5}$$

which can be expressed more compactly as

$$y[i, j, k] = \sum_{r=0}^{n_3-1} x\,[i : i + w_1 - 1, j : j + w_2 - 1, r] * W[:; r, k] \tag{6}$$

where we use the standard slicing conventions for arrays and $*$ denotes the ordinary scalar product of matrices. Notice that the shape of y is $[n_1 - w_1 + 1, n_2 - w_2 + 1, m_3]$.

There is also a *downsampled cross-correlation* $y = x \star_s W$ by a *stride* s :

$$\begin{aligned} y[i, j, l] &= \sum_{k,m,n} x[is + m, js + n, k] W[m, n, k, l] \\ &= \sum_{k} x\,[is : is + w_1 - 1, js : js + w_2 - 1, k] * W[:; k, l] \end{aligned} \tag{7}$$

The shape of the array $x \star_s W$ is $\left[n_1', n_2', n_3\right]$, where n_1' and n_2' are the greatest integers such that $n_1' \leq (n_1 - w_1)/s$ and $n_2' \leq (n_2 - w_2)/s$.

A point about terminology: When a NN has at least one convolutional layer, we qualify it as a *convolutional* NN (CNN for short).

For a maximum pooling (*maxpool*) layer, the parameters are represented by a triple of positive integers $(w_1, w_2, s = 1)$, where (w_1, w_2) is the shape of the pooling window and s is the stride $\big($1 by default). In this case $\star_\pi = \star_{\text{mp}}$ is given by the rule

$$\left(x \star_{\text{mp}} W\right)[i, j, k] = \max\left(x\,[is : is + w_1 - 1, js : js + w_2 - 1, k]\right). \tag{8}$$

The shape of the array $x \star_{\text{mp}} W$ is $\left[n_1', n_2', n_3\right]$, where n_1' and n_2' are the greatest integers such that $n_1' \leq (n_1 - w_1)/s$ and $n_2' \leq (n_2 - w_2)/s$.

Training. A training algorithm for the network (2) using a labeled dataset $\mathcal{D}^*$ is any procedure to adjust the weights W_j and biases b_j so that the function $f_{W_m, b_m} \circ \cdots \circ f_{W_0, b_0}$ computed by the net has a good balance of the learning and generalization rates. This is usually done by iterating two steps: a forward pass ending with a measure (*loss*) of how close the result is to what it should be, and a backward pass to modify the parameters in order to decrease the loss incurred in the forward step. When the number of parameters is less than the number of data samples, we are faced with an unavoidable trade-off: the learning and the generalization can both

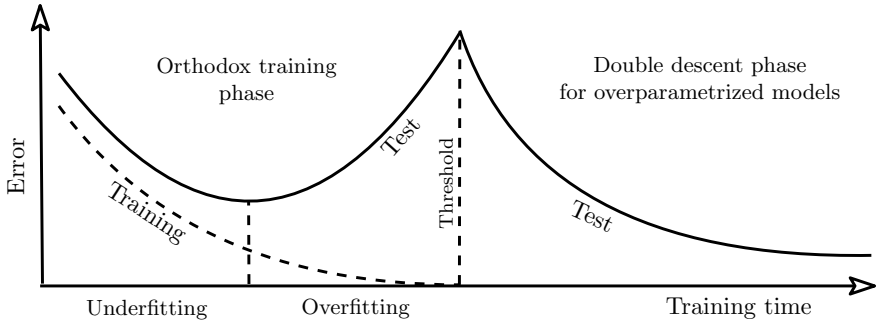


Fig. 4 Adaptation of Fig. 1 in [17]. The Threshold marks the vanishing of the training error. Below the threshold, we have the orthodox phase, with a mark for the trade-off boundary between underfitting and overfitting. The part on the right represents the extraordinary behavior, quite paradoxical at a first glance, that occurs when the number of parameters exceeds the number of training data: the generalization error decreases again from the peak that separates the two phases, thus making possible a perfect learning of the training data together with a high generalization capacity that increases with the number of parameters

increase for a while, but there is a turning point beyond which the learning keeps improving but the generalization enters a steady degrading. Before the turning point we have *underfitting*, in the sense that the learning and the generalization rates can still improve. After the turning point we have *overfitting*, in the sense that we are forcing the net to have a higher learning rate at the expense of a poorer generalization rate (under these circumstances, rote learning of the data does not favor the generalization capacity). This is the underparemeterized scenario (capacity, as measured by the number of weights, below the number of data) and it was the accepted wisdom until not long ago.

The question of what happens with overparameterized networks, a scenario favored by the increasing computing power, has been addressed in the last couple of years and the answers so far are surprising breakthroughs. Prominent among such discoveries is the 'double descent' phenomenon described in [17], which shows that for overparameterized NNs the training follows the pattern explained above until reaching zero training error, corresponding to a threshold of maximal testing error, and then the test error starts decreasing steadily and becomes smaller than the relative minimum achieved before the threshold (see Fig. 4). To learn more about this fascinating behavior, see [19] (on the role of kernel learning in deep learning), [18] (two models of double descent), and also [103, 113].

Further references for NNs: [138] (overview of DL in NNs), [154] (DL with SVM), [101] (mathematical underpinnings of CNN), [105] (deep versus shallow performance of NNs), [162] (mathematics of DL), [177] (universality of deep CNNs).

3 Geometric Neural Networks (GNN)

The first part of this section, devoted to briefly explain some basic notions of geometric algebra, is meant to ease the understanding of the related concepts introduced in the second part.

A Sketch of Geometric Algebra. The reason for using geometric algebras is that their formalism is optimally adapted to express the geometric facts of any *linear geometric space*, that is, of a real vector space $E = E_{r,s}$ endowed with a metric (a bilinear symmetric real-valued product $x \cdot x', x, x' \in E$) of signature (r, s). The most direct way to introduce the geometric algebra $\mathcal{G}_{r,s}$ of this space, one that is arguably the closest to the ideas on which W. K. Clifford (1845–1879) based his creation, is that Grassmann's exterior algebra of E, ΛE, has a unique bilinear **associative** product with unit 1 (called *geometric product* by Clifford himself) such that

$$xa = x \cdot a + x \wedge a \quad (x \in E, a \in \Lambda E), \tag{9}$$

where $x \cdot a = i_x(a)$ (the contraction of x with a).

Since i_x is the unique skew derivation of ΛE such that $i_x(x') = x \cdot x'$ for any $x' \in E$, the formula (9) shows how to multiply any multivector a by any vector x on the left. In fact, the formula suffices for the calculation of any product of multivectors because of the following reasoning. By bilinearity, it is enough to know how to multiply a non-zero exterior product $b = x_1 \wedge \cdots \wedge x_r$ $(r \geq 2)$ of vectors $x_1, \ldots, x_r$ (such products are called *r-blades*) by an arbitrary multivector a. We can further assume that $x_1, \ldots, x_r$ are pair-wise orthogonal, for the space $\langle x_1, \ldots, x_r \rangle$ has orthogonal bases and the exterior product of any such basis is equal, up to a multiplicative constant, to b. Finally we have that $x_1 \cdots x_r = b$ (by induction on r we may assume that $x_2 \cdots x_r = x_2 \wedge \cdots \wedge x_r$, and then $x_1 x_2 \cdots x_r = x_1(x_2 \wedge \cdots \wedge x_r) = x_1 \wedge x_2 \wedge \cdots \wedge x_r$ because $x_1 \cdot (x_2 \wedge \cdots \wedge x_r) = 0$). So $ba = x_1 \cdots x_r a$, a product that can be determined by r applications of (9). For an extensive study of geometric algebras in a similar spirit, see [171, chapter 3], or the forthcoming [172], which also includes the treatment of nonlinear geometric spaces.

Now the *geometric algebra* $\mathcal{G}_{r,s}$ is the exterior algebra $\Lambda E_{r,s}$ enriched with the geometric product (this structure is also known as *Clifford's algebra*). It is clear then that it has dimension 2^n, where $n = r + s = \dim E$. Note that the Eq. (9) shows that the linear grading of $\mathcal{G}_{r,s}$, which is in fact a grading with respect to the exterior product, is not a grading with respect to the geometric product. But the decomposition $\mathcal{G} = \mathcal{G}^+ \oplus \mathcal{G}^-$ into *even* $(\mathcal{G}^+)$ and *odd* $(\mathcal{G}^-)$ degree components is a grading mod 2 *also with respect to the geometric product* (ultimately this is derived from the Eq. (9), by which the product of two vectors is resolved as the sum of a scalar, which has degree 0, and a bivector, which has degree 2). In particular, $\mathcal{G}^+$ is a subalgebra.

The isomorphisms $\mathcal{G}_{1,0} \simeq \mathbb{R} \oplus \mathbb{R}$, $\mathcal{G}_{0,1} \simeq \mathcal{G}^+_{2,0} \simeq \mathbb{C}$, $\mathcal{G}_{2,0} \simeq \mathbb{R}(2)$, or $\mathcal{G}_{0,2} \simeq \mathcal{G}^+_{3,0} \simeq \mathbb{H}$, easy to derive directly, are in fact examples of a general trend (cf. [171]): $\mathcal{G}_{r,s}$ is isomorphic to a matrix algebra $F_\nu(m)$, where $\nu = s - r \mod 8$, $F_\nu = \mathbb{R}, \mathbb{C}, \mathbb{H}, 2\mathbb{H}, \mathbb{H}, \mathbb{C}, \mathbb{R}, 2\mathbb{R}$ for $\nu = 0, 1, 2, 3, 4, 5, 6, 7$, and $\dim(F_\nu)m^2 = 2^n$.

For example, $\mathcal{G}_{1,3} = F_2(m) = \mathbb{H}(2)$. Of these isomorphisms, those that most closely connect algebra with geometry are $\mathbf{C} = \mathcal{G}^+_{2,0} \simeq \mathbb{C}$ and $\mathbf{H} = \mathcal{G}^+_{3,0} \simeq \mathbb{H}$ in the case of the Euclidean plane and space, respectively (of $\mathbf{C}$ and $\mathbf{H}$ we say that they are the *geometric* complex numbers and quaternions, respectively, since they emerge directly from the geometry and not from ad hoc definitions as the usual ones for $\mathbb{C}$ and $\mathbb{H}$). For samples of various applications of geometric algebra, see [89, 171] and their bibliographies.

For a discussion of a broader perspective of geometric algebra and its applications, see [94] (especially Ch. 1), and the references cited there.

$\mathcal{A}$-neurons and $\mathcal{A}$-networks. The quantities x_j and w_j used in the neuron model introduced in Eq. (1) are real numbers. But we can imagine that they are entities of an algebraic structure $\mathcal{A}$ sufficient to guarantee that the expression $x \cdot w = x_1 w_1 + \cdots + x_n w_n$, and an activation function $\sigma : \mathcal{A} \to \mathcal{A}$, make sense. For example, $\mathcal{A}$ can be a real algebra of finite dimension and σ the function of an ordinary sigmoid applied component-wise (with respect to a fixed basis of the algebra). We thus arrive at the concept of *$\mathcal{A}$-neuron* and, connecting neurons as we have done before, to the notion of *$\mathcal{A}$-neural network*, or $\mathcal{A}$-NN. Another generalization is to replace x and w with more general data structures, such as $\mathcal{A}$-arrays (or tensors), and the product $x \cdot w$ by a suitable operation $x \star w$. Among these operations, the most commonly used are certain bilinear products, such as *cross-correlation*, as well as nonlinear ones, such as max-pooling.

Thus the usual neurons and neural networks are $\mathbb{R}$-neurons and $\mathbb{R}$-neuronal networks. Beyond real numbers, among the most immediate concrete cases of algebras $\mathcal{A}$ we can mention $\mathbb{C}$ (complex numbers), $\mathbb{H}$ (quaternions), $\mathbb{O}$ (octonions), an algebra of matrices $\mathbb{R}(n)$, or a geometric algebra $\mathcal{G} = \mathcal{G}_{r,s}$ of signature (r, s). To simplify the terminology, we will talk about real, complex, quaternionic (QNN), octonionic (ONN), matrix (MNN), and geometric (GNN) networks, respectively.

One advantage of $\mathcal{A}$-neurons is that the number of (real) weights they require decreases in inverse proportion to $d = \dim \mathcal{A}$. The argument is based on the simple observation that $w \in \mathcal{A}$ counts for d real weights, whereas both x and $x' = x \star w$ count for d real parameters each and hence we need d^2 real weights to connect them. Another advantage is that the algebraic structure of $\mathcal{A}$ can be regarded as a resource for describing and implementing AL algorithms, a point that is particularly relevant when $\mathcal{A}$ is a geometric algebra on account of its intimate connection with the geometry of its geometric space. This is not unlike the use of finite fields as alphabets for coding information, for being able to sum, multiply and divide alphabet symbols turns out to represent a great bonus with respect to a set with no structure.

Besides the further references provided henceforth, we find that the text [8] is a remarkably inspiring early reference for most of the topics discussed in this section. In particular, it studies complex NNs in Chap. 2 and QNNs in Chap. 5. It also features interesting applications of these algebras to predict chaotic time series (Chap. 6) and to robotics (Chap. 7).

Complex NNs. Perhaps the most important idea of these networks is that they can exploit the phase properties of complex numbers. At the beginning of the study of

these networks, the contributions of Hirose and his school stand out. They focus on signal processing, with collections such as [70] (2003) and treated as [116] (2009), or [68] (2012; a second edition of a book of the same title and author published in 2006), and the collection of ten articles collected in [69] (2013), of which the first stands out, by Hirose himself (the editor of the volume), with the title *Application fields and fundamental merits of complex-valued neural networks*. The text [3] belongs to the same circle, which illustrates with very convincing graphic experiments the value of considering the phase.

More recently we have [58] (2016), on complex convolutional networks; [127] (2017), focused on image classification; [158] (2017), where the emphasis is on deep networks; and [108] (2018), which provides an assessment of complex networks in real signal classification tasks. Finally we mention [26], which reveals the significance of complex networks from other points of view, particularly that of deep AL.

QNNs. The interest of these networks comes from the relation that the quaternions keep with the group of rotations of the ordinary Euclidean space, a relation especially transparent in terms of $\mathbf{H}$, for the expression $\underline{h}(x) = hx\bar{h}, h \in \mathbf{H}$ non-zero, is a vector and $\underline{h}$ is a similarity of ratio $|h|^2$ (a rotation if h is unitary). Another reason is that quaternions have three phases and that these phases can be used to extract valuable information from the signals to be processed.

Research in QNNs also comes from long ago, even before that of complex networks. We refer to [22] and [72] for relevant historical information regarding what is called *Clifford's analysis*, especially in relation to Fourier and wavelet transforms in a quaternionic context and their generalization to the geometric context. In the origin of the more specific topic we are considering, we find Gerald Sommer and his collaborators: [33] (generalization of Gabor filters) and [30] (generalization of the real multilayer perceptron, cf. [62]). The report [38] presents a quaternion wavelet theory "for image analysis and processing" and [75] an overview of the properties and applications of quaternion networks up to that point.

In the last decade, research on QNNs has continued both on the applied and theoretical fronts. The article [76] deals with the quaternionic multilayer perceptron. Hopfield QNNs and their rotation invariance are investigated in [82]. In the works [120] and [121], the QNNs are applied to the comprehension of the spoken language. Deep QNNs are studied in [56] and convolutional ones in [179]. Finally [176] presents a quaternionic version of capsule networks aimed at processing point clouds in Euclidean space and in [110] a new QNN deterministic layer is introduced that provides contrast invariance and sensitivity to rotation angles using quaternionic Gabor functions and Hilbert transforms, while in [109] the authors use the Riesz transform in the quatermion monogenic representation to propose a novel deterministic convolution layer in the Fourier domain robust to contrast and haze changes in image classification.

GNNs. Interestingly, the study of GNNs began even before that of QNNs, as in [123], and G. Sommer was a strong proponent of this inquiry at the beginning of

the millennium with works such as [149], in which he developed the theoretical foundations that served him well for problems such as artificial vision and robotics; [31], dedicated to a $\mathcal{G}$-version of the multilayer perceptron; [50] and [34], which develop the notion of *monogenic signal*. A culmination of these efforts was Sven Buchholz's thesis, [28], which should be considered, as its title indicates, a theory of neuronal computation with geometric algebras. As a sample of applications, we cite [132] (image segmentation), [16] (support vectors in the geometric context), the volumes [13] (geometric computing for wavelet transforms, artificial vision, learning, control and action) and [15] (geometric computing in engineering and computer science), [53] (use of geometric algebra for edge detection in color images), [125] (clustering methods based on the conformal geometric algebra $\mathcal{G}_{4,1}$), and [166] (treatment of multispectral images with geometric algebra). We end with [14], the first volume of what should be a systematic treatment on these developments, but see also [42] and [12].

Other $\mathcal{A}$-NNs. In the recent article [169], convolutional octonion networks are constructed and applied to CIFAR-10 and CIFAR-100 image classification. According to the authors, they have better convergence and accuracy than other networks applied to the same tasks. Octonions have also been successfully applied to dictionary learning, as for instance in [90], an approach that can in fact be formulated for more general algebras, including geometric ones, as in [91].

Another recent example is the case when $\mathcal{A}$ is the algebra of *commutative quaternions*, $\mathbb{H}^c = \langle 1, \boldsymbol{i}_1, \boldsymbol{i}_2, \boldsymbol{i}_2 \rangle$. They were introduced by C. Segre in 1892 (see [141]) and can be defined by the relations $\boldsymbol{i}_1^2 = \boldsymbol{i}_3^2 = -1, \boldsymbol{i}_2^2 = 1, \boldsymbol{i}_1\boldsymbol{i}_2\boldsymbol{i}_3 = -1$. These imply that $\boldsymbol{i}_1\boldsymbol{i}_2 = \boldsymbol{i}_2\boldsymbol{i}_1 = \boldsymbol{i}_3$, $\boldsymbol{i}_2\boldsymbol{i}_3 = \boldsymbol{i}_3\boldsymbol{i}_2 = \boldsymbol{i}_1$, $\boldsymbol{i}_3\boldsymbol{i}_1 = \boldsymbol{i}_1\boldsymbol{i}_3 = -\boldsymbol{i}_2$, and hence $\mathbb{H}^c$ is commutative. This algebra has been revived in [117] at the level of what, in our notations, would be called $\mathbb{H}^c$-neurons.

Finally let us have a look to the recent paper [73]. In its Abstract we read:

> Our work considers a richer set of objects for activations and weights, and undertakes a comprehensive study of alternative algebras as number representations by studying their performance on two challenging problems: large-scale image classification using the ImageNet dataset and language modeling using the enwiki8 and WikiText-103 datasets. We denote this broader class of models as AlgebraNets. Our findings indicate that the conclusions of prior work, which explored neural networks constructed from $\mathbb{C}$ (complex numbers) and $\mathbb{H}$ (quaternions) on smaller datasets, do not always transfer to these challenging settings. However, our results demonstrate that there are alternative algebras which deliver better parameter and computational efficiency compared with $\mathbb{R}$. We consider $\mathbb{C}$, $\mathbb{H}$, $M_2(\mathbb{R})$ (the set of 2×2 real-valued matrices), $M_2(\mathbb{C})$, $M_3(\mathbb{R})$, $M_4(\mathbb{R})$, dual numbers and the $\mathbb{R}^3$ cross product. Additionally, we note that multiplication in these algebras has higher compute density than real multiplication, a useful property in situations with inherently limited parameter reuse such as auto-regressive inference and sparse neural networks.

These are all $\mathcal{A}$-NNs. Are they GNNs? By our comments on the isomorphism class of $\mathcal{G}_{r,s}$, this is certainly the case for $2\mathbb{R} \simeq \mathcal{G}_{1,0}$,[1] $\mathbb{C} \simeq \mathcal{G}_{0,1}$, $\mathbb{R}(2) \simeq \mathcal{G}_{2,0}$, $\mathbb{H} \simeq \mathcal{G}_{0,2} = \mathcal{G}^+_{3,0}$, $\mathbb{C}(2) \simeq \mathcal{G}_{1,2}$, and $\mathbb{R}(4) \simeq \mathcal{G}_{2,2}$. The exceptions are $\mathbb{R}(3)$ and $(\mathbb{R}^3, \times)$, as their

[1] $2\mathbb{R} = \mathbb{R} \oplus \mathbb{R}$ is the algebra of dual numbers, and in general $2\mathcal{A} = \mathcal{A} \oplus \mathcal{A}$. By $\mathcal{A}(n)$, or $M_n(\mathcal{A})$, we denote that algebra of $n \times n$ matrices with entries in $\mathcal{A}$.

dimensions are not powers of 2. Note however that the nature of the latter is also geometric, as the cross product is the Hodge dual of their wedge product, which lives in $\mathcal{G}_{3,0}$. See also [8, Ch. 3] (on Vectorial NNs). To remark also that although the octonions are not a geometric algebra, they are nevertheless a subalgebra of $\mathcal{G}_{0,7}$ (see [98, §7.4]).

4 Outlook

In this section we try to establish some connections between what has been said or hinted before and possible lines of inquiry in the area of AL by means of what can be described, in a broad sense, as geometric calculi. Our comments will refer to the following topics: AL of mathematical structures; Other faces of geometric AL; Robotics; Computational resources and techniques; Recent advances on k-NN; and Other liaisons.

AL of Mathematical Structures. A compelling illustration of this theme is reported in [87]. In our view, it represents a line of research that may be promising for AL of geometric calculi as well: "Neural networks have a reputation for being better at solving statistical or approximate problems than at performing calculations or working with symbolic data. In this paper, we show that they can be surprisingly good at more elaborated tasks in mathematics, such as symbolic integration and solving differential equations. We propose a syntax for representing mathematical problems, and methods for generating large datasets that can be used to train sequence-to-sequence models. We achieve results that outperform commercial Computer Algebra Systems such as Matlab or Mathematica" (from the paper's Abstract). For other works of a similar potential, see [63] (on learning algebraic structures), [4] (the bearing of AL on current research in number theory), [37] (a kindred report in the realm of physical sciences, with many useful insights in various aspects of AL), [64] ("a foray into discrete analogues of Riemannian manifolds, providing a rich interplay between combinatorics, geometry and theoretical physics"), [44] (on finding symbolic equations that match a given dataset, with the surprising illustration of an "overdensity equation for dark matter"), [39] (showing that "neural networks can learn advanced theorems and complex computations without built-in mathematical knowledge"), [95] (a version of AL that learns mappings between function spaces, with impressive applications to partial differential equations). Altogether, these works point out to novel avenues for inquiries in AL that are transforming the understanding of science in general and of mathematics in particular in ways never seen hitherto.

Other Faces of Geometric AL. For people working in geometric algebra/calculus, it is natural to term AL as "geometric" if based on those formalisms. But AL researchers came up with a different use for this qualification, as in [65]: "[...] we consider the general question of how to construct deep architectures with small learning complexity on general non-Euclidean domains, which are typically unknown and need to be estimated from the data". Even more explicit in these appraisals is [23]: "Geometric

deep learning is an umbrella term for emerging techniques attempting to generalize (structured) deep neural models to non-Euclidean domains such as graphs and manifolds. The purpose of this paper is to overview different examples of geometric deep learning problems and present available solutions, key difficulties, applications, and future research directions in this nascent field". Further evidence for the great potential of this paradigm can be gleaned in the survey [86], whose main thrust lies in linking graph neural networks and (neural) symbolic computing: "The need for improved explainability, interpretability and trust of AI systems in general demands principled methodologies, as suggested by neural-symbolic computing. In this paper, we review the state-of-the-art on the use of GNNs as a model of neural-symbolic computing". We do not regard the two views of "geometric" that we are considering as antagonistic in any way, as in fact we sense that each can benefit from the other.

Robotics. We have already mentioned the application of quaternions to robotics presented in [8, Ch. 7]. Among later texts, let us refer to the pioneer book [42], particularly Chaps. 2 and 7; Selig's treatise *Geometric fundamentals of robotics*, [142]; the collection [12], especially the papers in Part VIII (Geometry and Robotics), and the extensive compilation [13], especially Part IV (Geometric computing of robot kinematics and dynamics) and Part VI (Applications II: Robotics and medical robotics). For recent summaries of robotics analyzed with CGA, see [89, Ch. 4] and [175].

Concerning AL in robotics, it has proceeded largely in parallel to the geometric developments, as witnessed by [118] (how machine learning has been applied to robotic path-planning and path-planning related concepts), the survey [83] (reinforcement learning in robotics; see also [9]), Lenz' PhD thesis [93], and the surveys [143] (DL techniques for mobile robot applications), [134] (DL methods for robot vision), [153] (learning control in robotics). It appears ever more clearly that advanced AL is playing a major role in robotics aimed at providing all sorts of assisting services to humans, as epitomized by the memoir [41]. In all these cases, the opportunities for applying geometric methods to gain theoretical and applied advantages seem clearly plentiful, if only because of the many engineering aspects that concur in any such system.

Computational Resources and Techniques. Currently, there is a wealth of software (frameworks) for deep learning (see Comparison_of_deep-learning_software in Wikipedia). For example, Tensorflow (see [1]) provides "an interface for expressing machine learning algorithms, and an implementation for executing such algorithms". Most of them offer a Python interface and increasingly also a Julia interface, as for instance Tensorflow. An interesting case is Flux (2017), which is pure Julia (framework and interface). But as far as we know, none of these frameworks can deal with GNNs beyond complex NNs.

On the other hand, there is a rich variety of systems that perform computations with geometric algebras (see, for example, the Software section in the Wikipedia Geometric_algebra article). But again, and as far as we know, none offers a deep learning framework. By its design, the Julia system described in [131] has perhaps the highest potentiality to serve as a basis for developing such a framework. A first step

in this direction would be a framework supporting QNNs. Another useful resource is provided by template libraries, as for instance [51].

Recent Advances in Unsupervised Learning. The authors of [164] also express the view that "the theory [...] for many branches of unsupervised learning is still in its infancy" (end of §2.1). For our inquiry, there are two main directions to look at. One concerns recent advances in conventional (non-geometric) unsupervised learning, as for example [170], which orchestrates a powerful scenario for an automatic physicist with no supervision. In our appraisal, there is much that can conceivably be transferred to other domains, like the strategies that it advocates and the algorithmic ways by which they are marshaled. For other instances of a similar kind, see [85] (reconstruction of the periodic table), [124] (proposing "a family of biologically plausible artificial neural networks (NNs) for unsupervised learning") and [77] (steps "towards the long-term goal of machine-assisted scientific discovery from experimental data without making prior assumptions about the system").

The other direction is linking unsupervised learning with GNNs. Aside from contributions such as the innovative paper [125], which develops a clustering method based on CGA, it appears to be a largely uncharted terrain. Many of the ideas in the preceding paragraph may be relevant for these explorations. In this, it may bear further fruits the unsupervised learning of Lie group transformations studied in [148] on account of its generality and the geometric character of Lie groups (cf. [171, §6.5]).

Other Liaisons. In Sect. 3 we have met layered $\mathcal{A}$-NNs, but now it is convenient for us to allow more flexible architectures. By adapting the conventional notions about graph NNs (cf. [48, 178]), we find that a suitable class, among many other possible generalizations, is formed by directed acyclic graphs (N, E) with no isolated nodes and endowed with (trainable) *weights* $w_e \in \mathcal{A}\,(e \in E)$ and, for each non-initial node n, (trainable) *biases* $b_n \in \mathcal{A}$ and activation functions $\sigma_n : \mathcal{A} \to \mathcal{A}$. The states of a node are in one-to-one correspondence with elements of $a \in \mathcal{A}$. The initial nodes are input nodes. For a non-initial node n, its state a_n is determined by the formula $a_n = \sigma_n(b_n + \sum_{e:e_0=n} w_e a_{e_1})$, where e_0 and e_1 are the nodes connected by the edge e. The output of the net is given by the states of the terminal nodes produced by these rules. In the layered $\mathcal{A}$-NN, the initial (final) nodes are those of L_0 (L_{m+1}). Let us also suggest that it may be productive, particularly in the case of GNNs, to allow that weights w be operators acting on states a according to suitable law $w \star a$ (let us dub $\star$NNs these structures). These notions draw some inspiration from [89, Ch. 5] and [88] (on oriented CGA and its application to molecular distance geometry), and actually it looks puzzling to see whether they could help in porting AL to bear on the problems tackled by molecular distance geometry (see the more specific comments on AL in Chemistry at the end of this section). In doing so, it is important to bear in mind early trailblazers on Clifford neurons such as [29, 32, 71].

Other areas where the scheme may provide analytic and geometric advantages is in the treatment of 3D point clouds (see the survey [59], and papers like [54, 66, 165]), as well as in devising more powerful capsule nets: see [67, 136, 173, 174]. Of these, only the latter operates with complex numbers. Since CapsNets process elementary patterns, they should benefit from drawing ideas about pattern theories,

say in the sense of the monograph [111], and also to enhance explainability along the lines of [144].

A few hints on the aptness of ⋆NNs to properly deal with invariance and covariance properties are in order. These concepts always refer to the action of some group. If a group $G = \{g\}$ acts on a set $\mathcal{X}$, a function $f(x)$ is *invariant* under this action if $f(g \cdot x) = f(x)$ for all $x \in \mathcal{X}$ and $g \in G$. Similarly, if G also acts on a set $\mathcal{Y}$, a map $f : \mathcal{X} \to \mathcal{Y}$ is *covariant* (or *equivariant*) with respect to the actions of G in $\mathcal{X}$ and $\mathcal{Y}$ if $f(g \cdot x) = g \cdot f(x)$ for all $x \in \mathcal{X}$ and $g \in G$. Note that an invariant function f is covariant if we let the action of G act trivially on the range of f, so that $g \cdot f(x) = f(x)$ for all $x \in \mathcal{X}$. The main reason in the context of AL to care about G-covariance is that no data augmentation is required to recognize features in arbitrary G-poses, as in [40] for discrete groups of rigid motions.

Let us go back to AL for Chemistry. We note a perceptible ⋆ character of the networks studied in works such as [57, 81, 122, 147, 156], which motivates a careful study of their contributions from the ⋆NN point of view. See also the collection [128] and especially the paper [11], in which the relevant group is $SE(3)$, the group of distance-preserving transformations of the ordinary Euclidean space. The main claim is that the authors "directly verify that the performance gains are connected with the unique SE(3)-equivariant convolution architecture of the new model". Even closer to the spirit of our disquisition is [157], as for us geometric algebras are optimally suited for the treatment of tensors in the sense of this paper, and many other geometric entities and formalisms as well. By the way, we note that the AlgebraNets that we have seen before are special cases of ⋆NNs, and that they have been mainly applied to classification problems, but more research could uncover properties and applications based on their geometric character.

We end with a few remarks on the scattering transforms (a special kind of CNN) introduced in [27] and further studied in [55] (for graph networks), [25] and [119]. The computational side of this transform has produced the system [6]. Altogether, it would be worthwhile to define and study a geometric scattering transform based on the geometric algebra wavelet theory first introduced in [106] and further exploited in [38] (for quaternions), the collection [129] (particularly the paper by P. Cerejeiras, M. Ferreira, and U. Kähler), and [72]. It would also be gainful to devise a scattering transform network that could be trained, both in the conventional sense and in the geometric realm just mentioned, and a computational platform that could deal with both.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: TensorFlow: large-scale machine learning on heterogeneous distributed systems (2016). http://export.arxiv.org/pdf/1603.04467. www.tensorflow.org
2. Abe, S.: Support Vector Machines for Pattern Classification, 2nd edn., vol. 2. Springer, London (2010). First edition published in 2005

3. Aizenberg, I.: Complex-Valued Neural Networks with Multi-valued Neurons, vol. 353. Springer, Heidelberg (2011)
4. Alessandretti, L., Baronchelli, A., He, Y.-H.: Machine learning meets number theory: the data science of Birch-Swinnerton-Dyer (2019). https://arxiv.org/pdf/1911.02008.pdf
5. Alpaydin, E.: Introduction to Machine Learning, 4th edn. Adaptive Computation and Machine Learning. MIT Press, Cambridge (2020). 1st edn. 2004; 2nd, 2010; 3rd, 2014
6. Andreux, M., Angles, T., Exarchakis, G., Leonarduzzi, R., Rochette, G., Thiry, L., Zarka, J., Mallat, S., Andén, J., Belilovsky, E., et al.: Kymatio: scattering transforms in Python. J. Mach. Learn. Res. **21**(60), 1–6 (2020)
7. Anthony, M., Bartlett, P.L.: Neural Network Learning: Theoretical Foundations. Cambridge University Press, Cambridge (1999)
8. Arena, P., Fortuna, L., Muscato, G., Xibilia, M.G.: Neural networks in multidimensional domains: fundamentals and new trends in modelling and control, vol. 234 of LNCIS. Springer (1998)
9. Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: Deep reinforcement learning: a brief survey. IEEE Signal Process. Mag. **34**(6), 26–38 (2017). https://discovery.ucl.ac.uk/id/eprint/10083557/1/1708.05866v2.pdf
10. Barron, A.R.: Approximation and estimation bounds for artificial neural networks. Mach. Learn. **14**(1), 115–133 (1994)
11. Batzner, S., Smidt, T.E., Sun, L., Mailoa, J.P., Kornbluth, M., Molinari, N., Kozinsky, B.: SE(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials (2021). https://arxiv.org/pdf/2101.03164.pdf
12. Bayro-Corrochano E.: Handbook of Geometric Computing. Springer, Heidelberg (2005). Paperback edition 2010
13. Bayro-Corrochano, E.: Geometric Computing: For Wavelet Transforms, Robot Vision, Learning. Control and Action. Springer, Heidelberg (2010)
14. Bayro-Corrochano, E.: Geometric Algebra Applications Vol. I: Computer Vision, Graphics and Neurocomputing. Springer, Cham (2018)
15. Bayro-Corrochano, E., Scheuermann, G. (eds.): Geometric Algebra Computing. In Engineering and Computer Science. Springer, London (2010)
16. Bayro-Corrochano, E.J., Arana-Daniel, N.: Clifford support vector machines for classification, regression, and recurrence. IEEE Trans. Neural Netw. **21**(11), 1731–1746 (2010)
17. Belkin, M., Hsu, D., Ma, S., Mandal, S.: Reconciling modern machine-learning practice and the classical bias–variance trade-off. Proc. Natl. Acad. Sci. **116**(32), 15849–15854 (2019). https://arxiv.org/pdf/1812.11118.pdf
18. Belkin, M., Hsu, D., Xu, J.: Two models of double descent for weak features (2019). https://arxiv.org/pdf/1903.07571.pdf
19. Belkin, M., Ma, S., Mandal, S.: To understand deep learning we need to understand kernel learning (2018). https://arxiv.org/pdf/1802.01396.pdf
20. Blum, A., Hopcroft, J., Kannan, R.: Foundations of Data Science. Cambridge University Press, Cambridge (2020)
21. Bostrom, N.: Superintelligence: Paths, Dangers. Strategies. Oxford University Press, New York (2014)
22. Brackx, F., Hitzer, E., Sangwine, S.J.: History of quaternion and Clifford-Fourier transforms and wavelets. Quaternion Clifford Fourier Transf. Wavelets **27**, XI–XXVII (2013)
23. Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P.: Geometric deep learning: going beyond Euclidean data. IEEE Signal Process. Mag. **34**(4), 18–42 (2017)
24. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners (2020). https://arxiv.org/pdf/2005.14165.pdf
25. Bruna, J.: The scattering representation. In: Mathematics of Deep Learning. Cambridge Univ. Press (2019, forthcoming)
26. Bruna, J., Chintala, S., LeCun, Y., Piantino, S., Szlam, A., Tygert, M.: A mathematical motivation for complex-valued convolutional networks. arXiv:1503.03438 (2015)

27. Bruna, J., Mallat, S.: Invariant scattering convolution networks. IEEE Trans. Pattern Anal. Mach. Intell. **35**(8), 1872–1886 (2013)
28. Buchholz, S.: A theory of neural computation with Clifford algebras. Ph.D. thesis, Christian-Albrechts Universität Kiel (2005)
29. Buchholz, S., Hitzer, E., Tachibana, K.: Coordinate independent update formulas for versor Clifford neurons. In: SCIS & ISIS 2008, Japan Society for Fuzzy Theory and Intelligent Informatics, pp. 814–819 (2008)
30. Buchholz, S., Sommer, G.: Quaternionic spinor MLP. In: ESANN 2000 Proceedings, pp. 377–382. D-Facto. European Symposium on Artificial Neural Networks, Bruges (Belgium), 26–28 April 2000 (2000)
31. Buchholz, S., Sommer, G.: Clifford algebra multilayer perceptrons. In: Geometric Computing with Clifford Algebras, pp. 315–334. Springer (2001)
32. Buchholz, S., Tachibana, K., Hitzer, E.: Optimal learning rates for Clifford neurons. In: International Conference on Artificial Neural Networks, pp. 864–873. Springer (2007)
33. Bulow, T., Sommer, G.: Quaternionic Gabor filters for local structure classification. In: Proceedings. Fourteenth International Conference on Pattern Recognition (Cat. No. 98EX170), vol. 1, pp. 808–810. IEEE (1998)
34. Bülow, T., Sommer, G.: Hypercomplex signals - a novel extension of the analytic signal to the multidimensional case. IEEE Trans. Signal Process. **49**(11), 2844–2852 (2001)
35. Burges, C.J.C.: A tutorial on support vector machines for pattern recognition. Data Min. Knowl. Disc. **2**(2), 121–167 (1998)
36. Callaway, E.: 'It will change everything': DeepMind's AI makes gigantic leap in solving protein structures. Nature (2020)
37. Carleo, G., Cirac, I., Cranmer, K., Daudet, L., Schuld, M., Tishby, N., Vogt-Maranto, L., Zdeborová, L.: Machine learning and the physical sciences. Rev. Modern Phys. **91**(4), 045002 (2019). https://arxiv.org/pdf/1903.10563.pdf
38. Chan, W.L., Choi, H., Baraniuk, R.: Quaternion wavelets for image analysis and processing. In: IEEE International Conference on Image Processing, vol. 5, pp. 3057–3060 (2004)
39. Charton, F., Hayat, A., Lample, G.: Deep differential system stability–learning advanced computations from examples (2020). https://arxiv.org/pdf/2006.06462.pdf
40. Cohen, T., Welling, M.: Group equivariant convolutional networks. In: International Conference on Machine Learning, pp. 2990–2999 (2016). http://proceedings.mlr.press/v48/cohenc16.pdf
41. Colomé, A., Torras, C.: Reinforcement Learning of Bimanual Robot Skills, volume 134 of Springer Tracts in Advanced Robotics. Springer, Cham (2020)
42. Corrochano, E.B.: Geometric Computing for Perception Action Systems: Concepts, Algorithms, and Scientific Applications. Springer, New York (2001)
43. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
44. Cranmer, M., Sanchez-Gonzalez, A., Battaglia, P., Xu, R., Cranmer, K., Spergel, D., Ho, S.: Discovering symbolic models from deep learning with inductive biases (2020). https://arxiv.org/pdf/2006.11287v1.pdf
45. Cucker, F., Zhou, D.X.: Learning Theory: An Approximation Theory Viewpoint, vol. 24. Cambridge University Press, Cambridge (2007)
46. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Math. Control Signals Syst. **2**(4), 303–314 (1989)
47. Deisenroth, M.P., Aldo Faisal, A., Ong, C.S.: Mathematics for Machine Learning. Cambridge University Press, Cambridge (2020)
48. DeVore, R., Hanin, B., Petrova, G.: Neural network approximation (2020). https://arxiv.org/pdf/2012.14501.pdf
49. Domingos, P.: The Master Algorithm. Basic Books (2015). How the quest for the ultimate learning machine will remake our world
50. Felsberg, M., Sommer, G.: The monogenic signal. IEEE Trans. Signal Process. **49**(12), 3136–3144 (2001)
51. Fernandes, L.A.F.: Exploring Lazy Evaluation and Compile-Time Simplifications for Efficient Geometric Algebra Computations (2021). In this volume

52. Flöry, S.: Fitting B-spline curves to point clouds in the presence of obstacles. Institüt für Discrete Mathematik und Geometrie der Technischen Universität Wien (2005). https://www.rechenraum.com/de/assets/publications/simon_floery_da.pdf
53. Franchini, S., Gentile, A., Sorbello, F., Vassallo, G., Vitabile, S.: Clifford algebra based edge detector for color images. In: 2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems, pp. 84–91. IEEE (2012)
54. Franchini, S., Vitabile, S.: Geometric Calculus Applications to Medical Imaging: Status and Perspectives (2021). In this volume
55. Gama, F., Ribeiro, A., Bruna, J.: Diffusion scattering transforms on graphs (2018). https://arxiv.org/pdf/1806.08829.pdf
56. Gaudet, C.J., Maida, A.S.: Deep quaternion networks. In: 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2018)
57. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry (2017). https://arxiv.org/pdf/1704.01212.pdf
58. Guberman, N.: On complex valued convolutional neural networks (2016). https://arxiv.org/pdf/1602.09046.pdf
59. Guo, Y., Wang, H., Hu, Q., Liu, H., Liu, L., Bennamoun, M.: Deep Learning for 3D Point Clouds: A Survey (2019). https://arxiv.org/pdf/1912.12033.pdf
60. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference and Prediction. Springer Series in Statistics, 2nd edn. Springer, New York (2009)
61. Hawkins, J., Blakeslee, S.: On Intelligence: How a New Understanding of the Brain Will Lead to the Creation of Truly Intelligent Machines. Macmillan, New York (2007)
62. Haykin, S.: Neural Networks and Learning Machines, 3rd edn. Pearson (2009)
63. He, Y.-H., Kim, M.: Learning algebraic structures: Preliminary investigations (2019). https://arxiv.org/pdf/1905.02263.pdf
64. He, Y.-H., Yau, S.-T.: Graph Laplacians, Riemannian manifolds and their machine-learning (2020). https://arxiv.org/pdf/2006.16619.pdf
65. Henaff, M., Bruna, J., LeCun, Y.: Deep convolutional networks on graph-structured data (2015). arXiv:1506.05163
66. Hildenbrand, D., Hitzer, E.: Analysis of point clouds using conformal geometric algebra (2008). http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.151.7539
67. Hinton, G.E., Sabour, S., Frosst, N.: Matrix capsules with EM routing. In: ICLR 2018, 15 pages. Springer (2018)
68. Hirose, A.: Complex-Valued Neural Networks, 2nd edn. Springer (2012). Japanese edition 2004, first English edition 2006
69. Hirose, A. (ed.): Complex-Valued Neural Networks: Advances and Applications. Wiley, IEEE Computational Intelligence (2013)
70. Hirose, A. (ed.): Complex-Valued Neural Networks: Theories and Applications, volume 5 of Series on Innovative Intelligence. World Scientific (2003)
71. Hitzer, E.: Geometric operations implemented by conformal geometric algebra neural nodes (2013). https://arxiv.org/pdf/1306.1358.pdf
72. Hitzer, E., Sangwine, S.J.: Quaternion and Clifford Fourier Transforms and Wavelets. Springer, Basel (2013)
73. Hoffmann, J., Schmitt, S., Osindero, S., Simonyan, K., Elsen, E.: Algebranets (2020). https://arxiv.org/pdf/2006.07360.pdf
74. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Netw. **2**(5), 359–366 (1989)
75. Isokawa, T., Matsui, N., Nishimura, H.: Quaternionic neural networks: fundamental properties and applications. In: Complex-Valued Neural Networks: Utilizing High-dimensional Parameters, pp. 411–439. IGI Global (2009)
76. Isokawa, T., Nishimura, H., Matsui, N.: Quaternionic multilayer perceptron with local analyticity. Information **3**(4), 756–770 (2012). https://arxiv.org/pdf/1901.09342
77. Iten, R., Metger, T., Wilming, H., Del Rio, L., Renner, R.: Discovering physical concepts with neural networks. Phys. Rev. Lett. **124**(1), 010508 (2020)

78. James, G., Witten, D., Hastie, T., Tibshirani, R.: An Introduction to Statistical Learning (corrected at 4th printing), volume 112 of Springer Texts in Statistics. Springer (2014)
79. Jones, A., Kruger, C., Johnson, B.: The Unsupervised Learning Workshop. Data Science & Artificial Intelligence. Packt⟩ Publishing (2020)
80. Kanatani, K.: Overviews of optimization techniques for geometric estimation. Memoirs Facul. Eng. Okayama Univ. **47**, 1–18 (2013)
81. Klicpera, J., Groß, J., Günnemann, S.: Directional message passing for molecular graphs (2020). https://arxiv.org/pdf/2003.03123.pdf
82. Kobayashi, M.: Rotational invariance of quaternionic Hopfield neural networks. IEEJ Trans. Electr. Electron. Eng. **11**(4), 516–520 (2016)
83. Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: a survey. Int. J. Robot. Res. **32**(11), 1238–1274 (2013)
84. Kulkarni, S., Harman, G.: An Elementary Introduction to Statistical Learning Theory, vol. 853. Wiley, Hoboken (2011)
85. Kusaba, M., Liu, C., Koyama, Y., Terakura, K., Yoshida, R.: Recreation of the periodic table with an unsupervised machine learning algorithm (2019). https://arxiv.org/ftp/arxiv/papers/1912/1912.10708.pdf
86. Lamb, L., Garcez, A., Gori, M., Prates, M., Avelar, P., Vardi, M.: Graph neural networks meet neural-symbolic computing: a survey and perspective (2020). https://arxiv.org/pdf/2003.00330.pdf
87. Lample, G., Charton, F.: Deep learning for symbolic mathematics (2019). https://arxiv.org/pdf/1912.01412.pdf
88. Lavor, C., Alves, R.: Recent advances on oriented conformal geometric algebra applied to molecular distance geometry (2021). In this volume
89. Lavor, C., Xambó-Descamps, S., Zaplana, I.: A Geometric Algebra Invitation to Space-Time Physics. Robotics and Molecular Geometry. SBMA/Springerbrief. Springer, Cham (2018)
90. Lazendić, S., De Bie, H., Pižurica, A.: Octonion sparse representation for color and multispectral image processing. In: 2018 26th European Signal Processing Conference (EUSIPCO), pp. 608–612. IEEE (2018)
91. Lazendic, S., Pizurica, A., De Bie, H.: Hypercomplex algebras for dictionary learning. In: Early Proceedings of the AGACSE 2018 Conference, pp. 57–64. Unicamp/IMECC (2018). https://biblio.ugent.be/publication/8570237/file/8570249.pdf
92. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural Comput. **1**(4), 541–551 (1989)
93. Lenz, I.: Deep learning for robotics. Ph.D. thesis, Cornell University (2016)
94. Li, H.: Invariant algebras and geometric reasoning. World Scientific (2008)
95. Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., Anandkumar, A.: Fourier neural operator for parametric partial differential equations (2020). https://arxiv.org/pdf/2010.08895.pdf
96. Liu, Y., Yang, H., Wang, W.: Reconstructing B-spline curves from point clouds–a tangential flow approach using least squares minimization. In: International Conference on Shape Modeling and Applications 2005 (SMI 2005), pp. 4–12. IEEE (2005)
97. Liu, Z., Cohen, F., Zhang, Z.: Fitting B-splines to scattered data—new and old parameterization. In: 2014 International Conference on Multimedia Computing and Systems (ICMCS), pp. 75–80. IEEE (2014)
98. Lounesto, P.: Clifford Algebras and Spinors, 2nd edn., volume 286 of LMS Lecture Notes Series. Cambridge University Press, Cambridge (2001)
99. Lyche, T., Mørken, K.: Spline methods. Department of Informatics, Center of Mathematics for Applications, University of Oslo, Oslo (2008). http://pzs.dstu.dp.ua/DataMining/spline/bibl/splinDraft.pdf
100. van der Maaten, L., Hinton, G.: Visualizing data using t-SNE. J. Mach. Learn. Res. **9**, 2579–2605 (2008)

101. Mallat, S.: Understanding deep convolutional networks. Phil. Trans. R. Soc. A **374**(2065), 20150203 (2016)
102. McCorduck, P.: Machines Who Think: A personal Inquiry into the History and Prospects of Artificial Intelligence. A K Peters Ltd., London (2004)
103. Mei, S., Montanari, A.: The generalization error of random features regression: precise asymptotics and double descent curve (2019). https://arxiv.org/pdf/1908.05355.pdf
104. Metz, C.: Meet GPT-3. It has learned to code (and blog and argue) (2020). https://www.nytimes.com/2020/11/24/science/artificial-intelligence-ai-gpt3.html
105. Mhaskar, H., Liao, Q., Poggio, T.: Learning functions: when is deep better than shallow. arXiv:1603.00988 (2016)
106. Mitrea, M.: Clifford Wavelets, Singular Integrals, and Hardy Spaces. Springer, Heidelberg (1994)
107. Mohri, M., Rostamizadeh, A., Talwalkar, A.: Foundations of Machine Learning, 2nd edn. MIT Press, Cambridge (2018)
108. Mönning, N., Manandhar, S.: Evaluation of complex-valued neural networks on real-valued classification tasks. arXiv:1811.12351 (2018)
109. Moya-Sánchez, E.U., Xambó-Descamps S., Salazar Colores, S., Sánchez Pérez A., Cortés, U.: A Quaternion Deterministic Monogenic CNN Layer for Contrast Invariance (2021). In this volume
110. Ulises Moya-Sánchez, E., Xambó-Descamps, S., Pérez, A.S., Salazar-Colores, S., Martínez-Ortega, J., Cortés, U.: A bio-inspired quaternion local phase CNN layer with contrast invariance and linear sensitivity to rotation angles. Pattern Recogn. Lett. **131**, 56–62 (2020)
111. Mumford, D., Desolneux, A.: Pattern Theory: The Stochastic Analysis of Real-world Signals. A. K. Peters, London (2010)
112. Murphy, K.P.: Machine Learning: A Probabilistic Perspective. MIT Press, Cambridge (2012)
113. Nagarajan, V., Zico Kolter, J.: Deterministic PAC-Bayesian generalization bounds for deep networks via generalizing noise-resilience (2019). https://arxiv.org/pdf/1905.13344.pdf
114. Nielsen, M.A.: Neural Networks and Deep Learning, vol. 25. Determination Press, San Francisco (2015)
115. Nilsson, N.J.: The Quest For Artificial Intelligence–A History of Ideas and Achievements. Cambridge University Press, Cambridge (2009)
116. Nitta, T.: Complex-Valued Neural Networks: Utilizing High-Dimensional Parameters. IGI Global, Hershey (2009)
117. Nitta, T., Gan, H.H.: Fundamental structure of orthogonal variable commutative quaternion neurons. In: Proceedings of the Joint 11th International Conference on Soft Computing and Intelligent Systems and 21st International Symposium on Advanced Intelligent Systems, SCIS & ISIS2020 (Online conference, December 5–7, 2020), pp. 434–436 (2020)
118. Otte, M.: A survey of machine learning approaches to robotic path-planning. Int. J. Robot. Res. **5**(1), 90–98 (2008)
119. Pan, C., Chen, S., Ortega, A.: Spatio-temporal graph scattering transform (2020). https://arxiv.org/pdf/2012.03363.pdf
120. Parcollet, T., Morchid, M., Bousquet, P.-M., Dufour, R., Linarès, G., De Mori, R.: Quaternion neural networks for spoken language understanding. In: 2016 IEEE Spoken Language Technology Workshop (SLT), pp. 362–368. IEEE (2016)
121. Parcollet, T., Zhang, Y., Morchid, M., Trabelsi, C., Linarés, G., De Mori, R., Bengio, Y.: Quaternion convolutional neural networks for end-to-end automatic speech recognition. arXiv:1806.07789 (2018)
122. Park, C.W., Kornbluth, M., Vandermause, J., Wolverton, C., Kozinsky, B., Mailoa, J.P.: Accurate and scalable multi-element graph neural network force field and molecular dynamics with direct force architecture (2020). https://arxiv.org/ftp/arxiv/papers/2007/2007.14444.pdf
123. Pearson, J.K., Bisset, D.L.: Neural networks in the Clifford domain. In: Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN 1994), vol. 3, pp. 1465–1469. IEEE (1994)

124. Pehlevan, C., Chklovskii, D.B.: Neuroscience-inspired online unsupervised learning algorithms: artificial neural networks. IEEE Signal Process. Mag. **36**(6), 88–96 (2019). https://arxiv.org/pdf/1908.01867.pdf
125. Pham, M.T., Tachibana, K.: A conformal geometric algebra based clustering method and its applications. Adv. Appl. Clifford Algebras **26**(3), 1013–1032 (2016)
126. Pinkus, A.: Density in approximation theory (2005). https://arxiv.org/pdf/math/0501328.pdf
127. Popa, C.-A.: Complex-valued convolutional neural networks for real-valued image classification. In: 2017 International Joint Conference on Neural Networks (IJCNN), pp. 816–822. IEEE (2017)
128. Pyzer-Knapp, E.O., Laino, T.: Machine Learning in Chemistry: Data-Driven Algorithms, Learning Systems, and Predictions. ACS Publications, Washington, DC (2019)
129. Qian, T., Vai, M.I., Xu, Y. (eds.): Wavelet Analysis and Applications. Applied and Numerical Harmonic Analysis. Birkhäuser (2007). Includes: *Clifford analysis and the continuous spherical wavelet transform* (P. Cerejeiras, M. Ferreira, U. Kähler)
130. Ramge, T.: Who's afraid of AI? Fear and promise in the age of thinking machines. The Experiment (2019). Originally published in Germany as *Mensch und Maschine* by Philipp Reclam jun. Verlag GmbH & Co., 2018
131. Reed, M.: Differential geometric algebra with Leibniz and Grassmann. In: Proceedings JuliaCon, vol. 1, no. 1 (2019). https://github.com/chakravala/Grassmann.jl
132. Rivera-Rovelo, J., Bayro-Corrochano, E.: Medical image segmentation using a self-organizing neural network and Clifford geometric algebra. In: The 2006 IEEE International Joint Conference on Neural Network Proceedings, pp. 3538–3545. IEEE (2006)
133. Rosebrock, A.: Deep Learning for Computer Vision with Python: Starter Bundle. PyImageSearch (2017)
134. Ruiz-del Solar, J., Loncomilla, P., Soto, N.: A survey on deep learning methods for robot vision. arXiv:1803.10862 (2018)
135. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Pearson, Upper Saddle River (2016). 4th edition 2020
136. Sabour, S., Frosst, N., Hinton, G.E.: Dynamic routing between capsules. In: Advances in Neural Information Processing Systems, pp. 3856–3866. Springer (2017)
137. Sayols, N., Xambó-Descamps, S.: Learning curves from a sample of its points (2021, in preparation)
138. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
139. Schölkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge (2002)
140. Sederberg, T.W.: Computer Aided Geometric Design. BYU Scholars Archive (2012). https://scholarsarchive.byu.edu/facpub/1
141. Segre, C.: The real representations of complex elements and extension to bicomplex systems. Mathematishcen Annalen **40**, 413–467 (1892)
142. Selig, J.M.: Geometric Fundamentals of Robotics. Springer, New York (2004)
143. Shabbir, J., Anwer, T.: A survey of deep learning techniques for mobile robot applications. arXiv:1803.07608 (2018)
144. Shahroudnejad, A., Afshar, P., Plataniotis, K.N., Mohammadi, A.: Improved explainability of capsule networks: relevance path by agreement. In: 2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP), pp. 549–553 (2018). https://arxiv.org/pdf/1802.10204.pdf
145. Shalev-Shwartz, S., Ben-David, S.: Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, Cambridge (2014)
146. Smale, S., Yao, Y.: Online learning algorithms. Found. Comput. Math. **6**(2), 145–170 (2006)
147. Smidt, T.E., Geiger, M., Kurt Miller, B.: Finding symmetry breaking order parameters with Euclidean neural networks. Phys. Rev. Res. **3**(1), L012002 (2021)
148. Sohl-Dickstein, J., Wang, C.M., Olshausen, B.A.: An unsupervised algorithm for learning Lie group transformations. *arXiv preprint* arXiv:1001.1027 (2010). https://arxiv.org/pdf/1001.1027.pdf

149. Sommer, G. (ed.): Geometric Computing with Clifford Algebras: Theoretical Foundations and Applications in Computer Vision and Robotics. Springer, Heidelberg (2001)
150. Stone, C.J.: Consistent nonparametric regression. Ann. Stat., 595–620 (1977)
151. Strang, G.: Linear Algebra and Learning From Data. Wellesley-Cambridge Press, Cambridge (2019)
152. Sucholutsky I., Schonlau, M.: 'Less Than One'-Shot Learning: Learning N Classes From $M<N$ Samples (2020). https://arxiv.org/pdf/2009.08449.pdf
153. Tai, L., Zhang, J., Liu, M., Boedecker, J., Burgard, W.: A survey of deep network solutions for learning control in robotics: from reinforcement to imitation. arXiv:1612.07139, v4 (2018)
154. Tang, Y.: Deep learning using linear support vector machines. arXiv:1306.0239, v4 (2015)
155. Tegmark, M.: Life 3.0: Being human in the age of artificial intelligence. Knopf (2017)
156. Thomas, N., Smidt, T., Kearnes, S., Yang, L., Li, L., Kohlhoff, K., Riley, P.: Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds (2018). https://arxiv.org/pdf/1802.08219.pdf
157. Townshend, R.J.L., Townshend, B., Eismann, S., Dror, R.O.: Geometric prediction: moving beyond scalars (2020). https://arxiv.org/pdf/2006.14163.pdf
158. Trabelsi, C., Bilaniuk, O., Zhang, Y., Serdyuk, D., Subramanian, S., Santos, J.F., Mehri, S., Rostamzadeh, N., Bengio, Y., Pal, C.J.: Deep complex networks. arXiv:1705.09792 (2017)
159. Trefethen, L.N., Bau III, D.: Numerical Linear Algebra. Siam (1997)
160. Van Der Maaten, L.: Accelerating t-SNE using tree-based algorithms. J. Machine Learn. Res. **15**(1), 3221–3245 (2014)
161. Vapnik, V.: The Nature of Statistical Learning Theory, 2nd edn. Springer, New York (2000)
162. Vidal, R., Bruna, J., Giryes, R., Soatto, S.: Mathematics of deep learning. arXiv:1712.04741 (2017)
163. Vivancos, D.: Automate or be automated. Amazon Fulfillment, Poland (2020)
164. Von Luxburg, U., Schölkopf, B.: Statistical learning theory: models, concepts, and results. In: Handbook of the History of Logic, vol. 10, Inductive logic, pp. 651–706. Elsevier (2011)
165. Wang, H., Liu, Q., Yue, X., Lasenby, J., Kusner, M.J.: Pre-training by completing point clouds (2020). https://arxiv.org/pdf/2010.01089.pdf
166. Wang, R., Shi, Y., Cao, W.: GA-SURF: a new speeded-up robust feature extraction algorithm for multispectral images based on geometric algebra. Pattern Recogn. Lett. **127**, 11–17 (2019)
167. Wang, W., Pottmann, H., Liu, Y.: Fitting B-spline curves to point clouds by curvature-based squared distance minimization. ACM Trans. Graph. (ToG) **25**(2), 214–238 (2006)
168. Watt, J., Borhani, R., Katsaggelos, A.: Machine Learning Refined: Foundations, Algorithms, and Applications. Cambridge University Press, Cambridge (2020)
169. Jiasong, W., Ling, X., Fuzhi, W., Kong, Y., Senhadji, L., Shu, H.: Deep octonion networks. Neurocomputing **397**, 179–181 (2020)
170. Wu, T., Tegmark, M.: Toward an AI Physicist for Unsupervised Learning (2019). https://arxiv.org/pdf/1810.10525.pdf, v4
171. Xambó-Descamps, S.: Real spinorial groups–a short mathematical introduction. Springer, SBMA/Springerbrief (2018)
172. Xambó-Descamps, S.: Calculi on linear and non-linear geometric spaces (2021, to appear)
173. Xi, E., Bing, S., Jin, Y.: Capsule Network Performance on Complex Data (2017). http://arxiv.org/abs/1712.03480
174. Xiang, C., Zhang, L., Tang, Y., Zou, W., Xu, C.: MS-CapsNet: a novel multi-scale capsule network. IEEE Signal Process. Lett. (2018)
175. Zaplana, I.: New Perspectives on Robotics with Geometric Calculus (2021). In this volume
176. Zhao, Y., Birdal, T., Lenssen, J.E., Menegatti, E., Guibas, L., Tombari, F.: Quaternion Equivariant Capsule Networks for 3D Point Clouds (2019). http://arxiv.org/abs/1912.12098, v2, 2020
177. Zhou, D.-X.: Universality of deep convolutional neural networks. Appl. Comput. Harmonic Anal. (2019)
178. Zhou, Y., Zheng, H., Huang, X.: Graph neural networks: Taxonomy, advances and trends (2020). https://arxiv.org/pdf/2012.08752.pdf
179. Zhu, X., Xu, Y., Xu, H., Chen, C.: Quaternion convolutional neural networks. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 631–647 (2018)

Author Index

S. Xambó-Descamps (ed.), *Systems, Patterns and Data Engineering with Geometric Calculi*, SEMA SIMAI Springer Series 13, https://doi.org/10.1007/978-3-030-74486-1