Geometric Algebra for Optimal Control in Robotics using *gafro*

Tobias Löw*† and Sylvain Calinon*†
*Idiap Research Institute, Martigny, Switzerland
†EPFL, Lausanne, Switzerland

***Abstract*—Many problems in robotics are fundamentally problems of geometry, which lead to an increased research effort in geometric methods for robotics in recent years. The results were algorithms using the various frameworks of screw theory, Lie algebra and dual quaternions. A unification and generalization of these popular formalisms can be found in geometric algebra. The aim of this paper is to showcase the capabilities of geometric algebra when applied to robot manipulation tasks. In particular the modelling of cost functions for optimal control can be done uniformly across different geometric primitives leading to a low symbolic complexity of the resulting expressions and a geometric intuitiveness. We demonstrate the usefulness, simplicity and computational efficiency of geometric algebra in several experiments using a Franka Emika robot. The presented algorithms were implemented in c++20 and resulted in the publicly available library *gafro*. The benchmark shows faster computation of the kinematics than state-of-the-art robotics libraries.**

I. INTRODUCTION

Robot manipulators are used within an increased diversity of environments and tasks, which leads to a large increase in not only the complexity of the surroundings but also in the systems, that need to be able to adapt to different situations. To ensure safe and efficient interaction the corresponding algorithms need to be fast and be based on accurate models of the environment, which makes it important to think carefully about the representations that are used. Many robotics problems are fundamentally problems of geometry, which is why a lot of recent research is focusing on representing and utilizing these geometric properties for solving a wide variety of problems more efficiently. Screw theory, Riemannian geometry, Lie algebra and dual quaternions are just a few examples of the different methods that have been proposed to be used in robotics. Traditionally the kinematics and dynamics of the robots are expressed in different algebras, including linear algebra, vector calculus and quaternion algebra. While quaternions offer a way to avoid the singularities caused by Euler angles, they do not contain position information and thus conversion operations between algebras are required. To address this limitation, dual quaternions were proposed to extend quaternions by a dual unit, resulting in a translation and rotation. We propose in this paper to use geometric algebra instead, which can be seen as a further unification and generalization of these concepts. In particular conformal geometric algebra is a direct extension of dual quaternions [1].

This work was supported by SEFRI in Switzerland for participation in the EC's Horizon Europe Program through the INTELLIMAN (https://intelliman-project.eu/) and SESTOSENSO (http://sestosenso.eu/) projects.

Geometric Algebra (GA) can be seen as a *high-level mathematical language* for geometry that unifies several known concepts, which makes it a very effective tool when the physics of a system need to be modeled. The roots of geometric algebra can be found in Clifford algebra, which was a unification of quaternions and Grassmann algebra [2]. The result was the geometric product, which is the sum of an inner and an outer product. This unfamiliar concept actually leads to algebraic tools that allow for the simplification of many otherwise complex equations, making them more intuitive to handle. A well-known example for this simplification are the Maxwell equations, which reduce to only a single equation in geometric algebra $\left(\boldsymbol{\nabla} + \frac{1}{c}\frac{\partial}{\partial t}\right) F = J$ [3].

GA is based on a multiplication operation called the geometric product, composed of an inner product and an outer product. The latter describes an oriented plane/volume that extends and generalizes the cross product that is restricted to only 3 dimensions. The resulting elements are called multivectors. This representation can be used to encode geometric primitives in a uniform manner, such as points, lines, planes, spheres, or quadric surfaces such as ellipsoids, as well as the associated transformations u to move from an initial state x_0 to a desired state x_d, which are called motors. In robotics, these operations allow translations and rotations to be treated in the same way, without requiring us to switch between different algebras, as is classically done when handling position data in a Cartesian space and orientation data as quaternions. Practically, GA allows geometric operations to be computed in a very fast way, with compact codes.

The representational advantage of geometric algebra is the geometric significance of its elements, meaning that an object can directly represent geometric primitives, such as lines, spheres and planes, as well as orthogonal transformations, such as rotations, translations, scaling and projections. This allows the direct extraction of geometric information about the problem from the equations. Furthermore, its elements, called multivectors, avoid the parameter redundancy of other representations such as matrices, leading to less memory consumption and optimized computation compared to analytic geometry or vector calculus, which makes it an amenable framework for real-time applications. In engineering the validity of equations is usually determined by a dimensional check of the quantities of the formula. These quantities are of a certain algebraic order when using geometric algebra, which adds a structural check for the validity. These properties were some fundamental criteria in the design of geometric algebra, along with the possibility to formulate basic equations in a coordinate-free manner and to smoothly transfer information between formalisms [4].

Apart from our theoretical contributions we also provide an open-source library that implements all the presented formulations and algorithms. To this end, we have implemented the geometric algebra from scratch using expression templates. There have been various works that published implementations of geometric algebra such as GATL [5], GARAMON [6], Gaigen [7], TbGAL [8], GAL [9], Gaalet [10] and Versor [11]. These libraries all have in common that they are meant to be generic geometric algebra implementations focusing on the computational and mathematical aspects of the algebra itself. In contrast to that, our implementation is targeted specifically at robotics applications and thus not only implements the low-level algebraic computations but also features the computation of the kinematics and dynamics of serial manipulators as well as generic cost functions for optimal control. We therefore have a similar objective as the DQ robotics [12] library, but using the more general conformal geometric algebra as opposed to dual quaternions.

II. METHOD

In this section, we are presenting the mathematical tools that we have previously used in general optimization problems. The presented algorithms are implemented in c++20 in the publicly available library *gafro*
`https://github.com/idiap/gafro/`.

We use the following notation throughout the paper: x to denote scalars, $\boldsymbol{x}$ for vectors, $\boldsymbol{X}$ for matrices, X for multivectors and $\boldsymbol{\mathcal{X}}$ for matrices of multivectors.

A. Geometric Algebra

Geometric algebra is a unified algebra for geometric reasoning, alleviating the need of utilizing multiple algebras to express geometric relations [13]. The core idea of geometric algebra is its multiplication operation called the geometric product

$$\boldsymbol{a}\boldsymbol{b} = \boldsymbol{a} \cdot \boldsymbol{b} + \boldsymbol{a} \wedge \boldsymbol{b}, \tag{1}$$

which is the sum of an inner $\cdot$ and an outer $\wedge$ product [14]. The resulting algebra essentially includes $\mathbb{R}$ and the subspaces of the associated vector space as elements of computations [15]. A general element in geometric algebra is called a multivector and is the linear combination of basis blades. We are using a specific variant known as conformal geometric algebra (CGA) [16]. It embeds the 3-dimensional Euclidean space $\mathbb{R}^3$ into the 5-dimensional space $\mathbb{R}^{4,1}$, with the corresponding geometric algebra $\mathbb{G}_{4,1}$. In CGA every multivector consists of 32 basis blades given in grades from 0 to 5. This high dimension looks to be leading to an increased complexity, in practice, however, these multivectors usually are very sparse, a fact that we exploit in our implementation.

Points are the basic geometric primitives that can be used to construct more complex primitives such as lines, circles and planes by the spanning operation of the outer product. These geometric primitives are in general nullspace representations with respect to either the inner (IPNS) or the outer (OPNS) product, meaning that a geometric primitive is defined by the set of all Euclidean points that result in zero upon multiplication when embedded in CGA. In Figure 1, we show the subspaces that several geometric primitives occupy within the algebra to demonstrate their sparsity. The type of a multivector resulting from a product operation can thus be determined by looking at the expected non-zero elements of the expression. For a list of equations to construct the primitives, we refer the reader to [17].

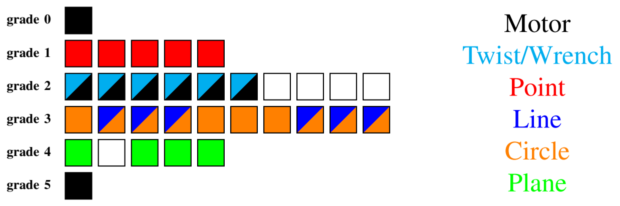

Fig. 1: Non-zero elements of various geometric primitives in their primal representations in conformal geometric algebra. Boxes represent basis blades and colored boxes represent the non-zero blades of the geometric primitive with the matching color. It can be seen that of the 32 basis blades composing multivectors only a sparse number is used for the representations. Note that geometric primitives are single-grade objects, while transformations are mixed-grade.

The multivectors that describe rigid body motions are called rotors, translators and more generally: motors. A general motor is hence composed of a translator and a rotor, i.e.

$$M = TR = \exp(B). \tag{2}$$

The motors in geometric algebra form a Lie group, which is an even sub-algebra $\mathbb{M}$ of $\mathbb{G}^+_{4,1}$. Its associated Lie algebra is the bivector algebra in the linear subspace $\mathbb{B}$.

A motor applied to multivectors results in a sandwiching product, similar to how quaternions rotate vectors

$$Y = MX\widetilde{M}, \tag{3}$$

where $\widetilde{M}$ stands for the reverse of a motor, which can thought of as being similar to a conjugate quaternion.

Motors are isomorphic to dual quaternions [17], which makes them also isomorphic to $SE(3)$. They represent, however, a more general concept of transformations that is valid in any dimension.

B. Geometric Algebra for Serial Manipulators

The forward kinematics of a kinematic chain of N joints can easily be defined using motors [18]. Assuming that we only have revolute joints, the forward motor $M(\boldsymbol{q})$, given the configuration $\boldsymbol{q}$, can be computed with

$$M(\boldsymbol{q}) = \prod_{i=1}^{N} M_i(q_i) = \prod_{i=1}^{N} M_{F,i} R_i(q_i). \tag{4}$$

The constant joint-specific motors $M_{F,i}$ represent the local frames of the joints with the rotation in that frame expressed by the rotor

$$R_i(q_i) = \exp\left(-\frac{1}{2}q_i B_i\right), \tag{5}$$

where the bivectors B_i essentially represent the screw axes of the joints.

The analytic Jacobian is defined as the partial derivatives of the forward kinematic function $\boldsymbol{f}(\boldsymbol{q})$ defined in Equation (4) w.r.t. the joint angles, i.e.

$$\boldsymbol{\mathcal{J}}^A(\boldsymbol{q}) = \frac{\partial M(\boldsymbol{q})}{\partial \boldsymbol{q}} = \left[\frac{\partial M(\boldsymbol{q})}{\partial q_1} \dots \frac{\partial M(\boldsymbol{q})}{\partial q_N}\right]. \tag{6}$$

Note that the size of the multivector matrix is $\boldsymbol{\mathcal{J}}^A(\boldsymbol{q}) \in \mathbb{M}^{1\times N} \subset \mathbb{G}_{4,1}^{1\times N}$ with its elements corresponding to motors.

More information on using conformal geometric algebra for the kinematics/dynamics computation of serial manipulators can be found in [19].

C. Optimal Control using Geometric Algebra

Optimal control is a well-known technique that deals with the problem of finding a control sequence that minimizes an objective function. This objective function encodes the requirements of the task as well as the constraints of the robot and the environment. Modelling these mathematically requires special care, since they will determine the quality of the resulting solution. Furthermore optimal control can be applied as solver to a model predictive control problem, which requires fast convergence in order to achieve acceptable real time control rates. We will show that the use of geometric algebra for geometric primitives improves the clarity of equations and thus reduces computational difficulties. The modelling of the cost functions becomes easier and is done uniformly across all different primitives and is done directly in the error vector as opposed to the precision matrix, which results in a low symbolic complexity of expressions and a geometric intuitiveness, i.e. geometric meaning can directly be inferred.

Discrete-time optimal control aims at finding a control sequence that minimizes the cost function

$$\min_{\boldsymbol{u}} L(\boldsymbol{x}, \boldsymbol{u}) = l_f(\boldsymbol{x}_N) + \sum_{k=1}^{N-1} l_k(\boldsymbol{x}_k) + \|\boldsymbol{u}\|_R^2, \tag{7}$$

where $l_k(\boldsymbol{x}_k)$ and $l_f(k\boldsymbol{x}_N)$ are the state dependent running and final cost, respectively, and $\|\boldsymbol{u}\|_R^2$ is a regularization term representing a control cost.

Formulations in geometric algebra can be seamlessly integrated into optimization problems such as optimal control. To that end target poses can easily be defined using the motor manifold,

$$l(\boldsymbol{q}) = \left\| \log\left(\widetilde{M}_{\text{target}} M(\boldsymbol{q})\right) \right\|_2^2. \tag{8}$$

We also exploit the nullspace representations of the primitives for the formulation of the reaching objectives. By definition of the OPNS, the outer product is zero for any point that is on a geometric primitive. The multivector valued error of a reaching objective can be defined as

$$E(\boldsymbol{q}) = X_d \wedge M(\boldsymbol{q})X\widetilde{M}(\boldsymbol{q}), \tag{9}$$

where $M(\boldsymbol{q})X\widetilde{M}(\boldsymbol{q})$ corresponds to the tip of the end-effector (with $X = \boldsymbol{e}_0$) and the reaching target X_d can be any geometric primitive of the algebra. The Jacobian of the reaching task can be found by applying the chain rule to the multivector expressions

$$\boldsymbol{\mathcal{J}}^E(\boldsymbol{q}) = X_d \wedge \left(\boldsymbol{\mathcal{J}}^A(\boldsymbol{q})X\widetilde{M}(\boldsymbol{q}) + M(\boldsymbol{q})X\widetilde{\boldsymbol{\mathcal{J}}^A}(\boldsymbol{q})\right), \tag{10}$$

where $\boldsymbol{\mathcal{J}}^A(\boldsymbol{q})$ is the analytic Jacobian and the reverse of a multivector matrix is defined as the element-wise multivector reverse. The above expressions are exception-free (e.g. division by zero) and extend to other geometric primitives for X, e.g. when using a line the objective becomes a pointing task. This shows the strength of geometric algebra to uniformly model objective functions involving the geometric primitives.

For example, using the cost function formulation of geometric algebra that was presented in Equation (9) various reaching tasks can be defined. In general, for a reaching task, the end-effector should reach a certain position. This can be modeled by using a point for X. Then the desired multivector X_d can be any other geometric primitive, which in turn means that instead of only reaching a point, we can also reach lines, planes, circles and spheres. Higher order quadrics are possible as well, this however remains the subject of further investigations. We present optimal trajectories that were computed using the iterative linear quadratic regulator to explain how different geometric primitives can be reached using the same structure of the cost function, which is shown in Figure 2.

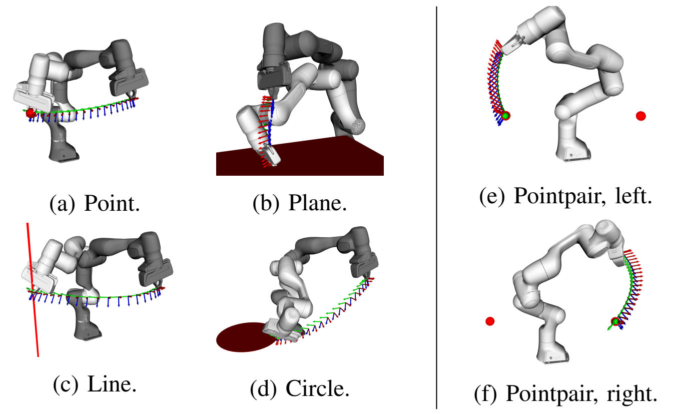

Fig. 2: Examples of optimal trajectories for reaching tasks using different geometric primitives. The initial configuration is always shown in gray and the final one in white. The target geometric primitive is shown in red. And the trajectory is depicted as the frames corresponding to the end-effector.

Using a pointpair as the target presents a special opportunity to model a control problem with options. A pointpair is the result of the outer product of two points. From this outer product nullspace representation, it follows that the outer product of the pointpair and any point $P = \mathcal{C}(\boldsymbol{x})$ with $\boldsymbol{x} \in \mathbb{R}^3$ is zero if and only if P is identical to one of the points that constructed the pointpair. The two possibilities that depend on the initial configuration are shown in Figures 2e and 2f.

The modelling of a pointing task only requires the usage of a line instead of a point for X in Equation (9). A possible scenario where this task would be applied is tracking an object with a robot arm endowed with a camera. The line can in this case be interpreted as the line of sight of the camera. Again, different geometric primitives can be used as the target, since the intersection of a line with any other primitive can be calculated in closed form without exceptions. An example is shown in Figure 3.

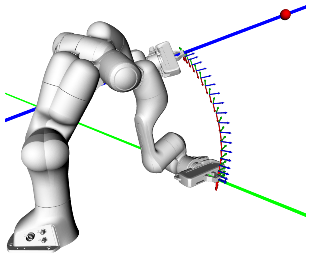
Fig. 3: Optimal trajectory example for a pointing task. The target is shown as the red point. The pointing line is defined to be collinear to the z-axis of the end-effector frame. It is shown in green for the initial configuration and in blue for the final configuration.

For more information on how to use geometric algebra for optimal control in manipulation tasks, we refer the interested reader to [19]. In that work we also present how this optimal control framework can be integrated into model predictive control using an inverse dynamics control approach.

D. Implementation Details

We implemented the presented robotics kinematics and dynamics algorithms along with cost functions for optimal in control in c++20. This resulted in the library *gafro*, which is publicly available. In this section, we are presenting this library on a high-level and highlight some of its features. A more in-depth presentation, exhaustive benchmarks and comparison to other libraries will be part of future work.

At the core of *gafro* is a custom implementation of conformal geometric algebra, that implements the multivectors as templates that take a blade index list as their argument. It exploits the sparsity of the multivector by only storing the data blades that are non-zero by the structure of the objects. Furthermore, it allows the straightforward usage of automatic differentiation libraries such as autdiff [1] for the computation of gradients and hessians of multivector expressions. The geometric, inner and outer products are implemented as expression templates, that are further exploiting this structure by only evaluating the elements of the resulting type that are known to be non-zero. The types are evaluated at compile time and the evaluation tree is constructed, which is then evaluated at runtime in a lazy fashion. One of our design goals for the library was the seamless integration with existing tools for robotics such as libraries for optimization and optimal control. To this end, we used the Eigen library [2], which is de facto the standard tool in robotics, to implement the sparse parameter vector of the multivectors.

Since this library implements robot kinematics and dynamics algorithms, we are comparing and benchmarking *gafro* against several libraries that are commonly used in robotics applications. These libraries include Raisim [20], Pinocchio [21] and KDL [22]. An excerpt of the benchmarking results can be found in Figure 4. As can be seen, our library can compute these important quantities considerably faster than the other libraries that are based on homogeneous transformation matrices.

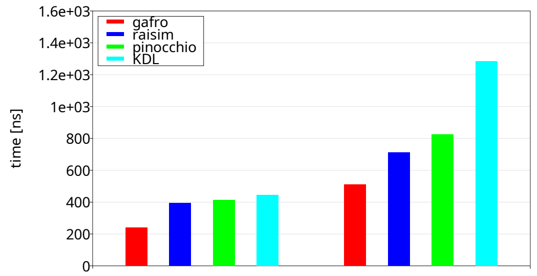

Fig. 4: Benchmark results for *gafro* compared to Raisim, Pinocchio and KDL. The four bars on the left correspond to the forward kinematics and the ones on the right to the Jacobian computation. The benchmarks were all performed on an AMD Ryzen 7 4800U CPU using the compiler flags `-O3 -msse3 -march=native`. The presented results are the average of 10000 executions with 10 repetitions.

III. Conclusion

We presented in this paper the usage of geometric algebra for the modelling of optimal control tasks and how to use the dynamics of serial manipulators computed with geometric algebra for inverse dynamics control.

The provided library, *gafro*, is currently specialized for conformal geometric algebra. The implementation of the multivectors and expressions that define the algebra is generic. Thus it would be possible to use geometric algebras with different signatures, which can be used to explore the usage of different geometric primitives such as quadric surfaces in this optimal control framework.

Higher order quadric surfaces such as cones and paraboloids in $\mathbb{G}_{6,3}$ [23] or ellipsoids and hyperboloids in $\mathbb{G}_{9,6}$ [24] are still a topic of ongoing research. In theory it should be possible to use them seamlessly in combination with the methods that we presented in this paper, since the properties of the different geometric algebras such as the outer product nullspace, which we rely on, remain the same. It is therefore the topic of future work to investigate the integration of these algebras into our formulation. The benefit of this would be a more versatile and generic modeling of surfaces that can be exploited for various manipulation tasks.

[1] https://autodiff.github.io/

[2] https://eigen.tuxfamily.org

References

[1] M. Kamarianakis and G. Papagiannakis, “An All-In-One Geometric Algorithm for Cutting, Tearing, and Drilling Deformable Models,” *Adv. Appl. Clifford Algebras*, no. 31, Jul. 6, 2021.

[2] S. Breuils, K. Tachibana, and E. Hitzer, “New Applications of Clifford’s Geometric Algebra,” *Adv. Appl. Clifford Algebras*, vol. 32, no. 2, p. 17, Apr. 2022.

[3] P. Joot, *Geometric Algebra for Electrical Engineers*. CreateSpace Independent Publishing Platform, 2019.

[4] D. Hestenes, G. Sobczyk, and J. S. Marsh, “*Clifford Algebra to Geometric Calculus. A Unified Language for Mathematics and Physics*,” *American Journal of Physics*, vol. 53, no. 5, pp. 510–511, May 1985.

[5] L. A. F. Fernandes, “Exploring Lazy Evaluation and Compile-Time Simplifications for Efficient Geometric Algebra Computations,” in *Systems, Patterns and Data Engineering with Geometric Calculi*, S. Xambó-Descamps, Ed., vol. 13, Cham: Springer International Publishing, 2021, pp. 111–131.

[6] S. Breuils, V. Nozick, and L. Fuchs, “Garamon: A Geometric Algebra Library Generator,” *Adv. Appl. Clifford Algebras*, vol. 29, no. 4, p. 69, Jul. 22, 2019.

[7] D. Fontijne, “Gaigen 2:: A geometric algebra implementation generator,” in *Proceedings of the 5th International Conference on Generative Programming and Component Engineering - GPCE ’06*, Portland, Oregon, USA: ACM Press, 2006, p. 141.

[8] E. V. Sousa and L. A. F. Fernandes, “TbGAL: A Tensor-Based Library for Geometric Algebra,” *Adv. Appl. Clifford Algebras*, vol. 30, no. 2, p. 27, Apr. 2020.

[9] J. Ong, *GAL*, https://github.com/jeremyong/gal: GitHub, 2019.

[10] F. Seybold and U. Wossner, “Gaalet - a C++ expression template library for implementing geometric algebra,” p. 10,

[11] P. Colapinto, “Versor: Spatial computing with conformal geometric algebra,” University of California at Santa Barbara, 2011.

[12] B. V. Adorno and M. Marques Marinho, “DQ Robotics: A Library for Robot Modeling and Control,” *IEEE Robotics Automation Magazine*, vol. 28, no. 3, pp. 102–116, Sep. 2021.

[13] E. Bayro-Corrochano, *Geometric Algebra Applications Vol. II: Robot Modelling and Control*. Cham: Springer International Publishing, 2020.

[14] D. Hestenes and G. Sobczyk, *Clifford Algebra to Geometric Calculus*. Dordrecht: Springer Netherlands, 1984.

[15] A. Macdonald, “A Survey of Geometric Algebra and Geometric Calculus,” *Adv. Appl. Clifford Algebras*, vol. 27, no. 1, pp. 853–891, Mar. 2017.

[16] E. Bayro-Corrochano, *Geometric Algebra Applications Vol. I*. Cham: Springer International Publishing, 2019.

[17] C. Perwass, *Geometric Algebra with Applications in Engineering* (Geometry and Computing 4). Berlin: Springer, 2009, 385 pp.

[18] E. Bayro-Corrochano and D. Kähler, “Motor algebra approach for computing the kinematics of robot manipulators,” *Journal of Robotic Systems*, vol. 17, no. 9, pp. 495–516, 2000.

[19] T. Löw and S. Calinon. “Geometric Algebra for Optimal Control with Applications in Manipulation Tasks.” (Dec. 14, 2022), [Online]. Available: http://arxiv.org/abs/2212.07237 (visited on 01/25/2023), preprint.

[20] J. Hwangbo, J. Lee, and M. Hutter, “Per-Contact Iteration Method for Solving Contact Dynamics,” *IEEE Robot. Autom. Lett.*, vol. 3, no. 2, pp. 895–902, Apr. 2018.

[21] J. Carpentier, G. Saurel, G. Buondonno, *et al.*, “The Pinocchio C++ library : A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives,” in *2019 IEEE/SICE International Symposium on System Integration (SII)*, Paris, France: IEEE, Jan. 2019, pp. 614–619.

[22] R. Smits, *KDL: Kinematics and Dynamics Library*, http://www.orocos.org/kdl.

[23] J. Zamora-Esquivel, “G 6,3 Geometric Algebra; Description and Implementation,” *Adv. Appl. Clifford Algebras*, vol. 24, no. 2, pp. 493–514, Jun. 2014.

[24] S. Breuils, V. Nozick, and E. Hitzer, “Quadric Conformal Geometric Algebra of R9,6,” p. 15,