Geometric Algebra for Optimal Control With Applications in Manipulation Tasks

Tobias Löw, *Student Member, IEEE*, and Sylvain Calinon, *Member, IEEE*

***Abstract*—Many problems in robotics are fundamentally problems of geometry, which have led 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 article is to showcase the capabilities of geometric algebra when applied to robot manipulation tasks. In particular, the modeling 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.**

***Index Terms*—Geometric algebra (GA), model-based optimization, optimal control.**

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 robotic 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 translation and rotation. We propose in this article to use geometric algebra (GA) instead, which can be seen as a further unification and generalization of these concepts. In particular, conformal geometric algebra (CGA) is a direct extension of dual quaternions [1].

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 GA 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 is the Maxwell equations, which reduce to only a single equation in GA $(\boldsymbol{\nabla} + \frac{1}{c}\frac{\partial}{\partial t})F = J$ [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 three dimensions. The resulting elements are called multivectors. This representation can be used to encode geometric primitives in a uniform manner (depicted in red in Fig. 1), 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 (depicted in green in Fig. 1). 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. In Fig. 1, it means that $u = f(x_0, x_d)$ can be described uniquely for the different geometric objects represented in the figure.

The representational advantage of GA is the geometric significance of its elements, meaning that an object can directly

Manuscript received 14 December 2022; revised 20 March 2023; accepted 8 May 2023. Date of publication 5 June 2023; date of current version 4 October 2023. This work was supported by the State Secretariat for Education, Research and Innovation in Switzerland under the European Commission's Horizon Europe Program through the INTELLIMAN project (https://intelliman-project.eu/; HORIZON-CL4-Digital-Emerging Grant 101070136) and the SESTOSENSO project (http://sestosenso.eu/; HORIZON-CL4-Digital-Emerging Grant 101070310). This paper was recommended for publication by Associate Editor J. Kelly and Editor E. Yoshida upon evaluation of the reviewers' comments. *(Corresponding author: Tobias Löw.)*

The authors are with the Idiap Research Institute, 1920 Martigny, Switzerland, and also with the Swiss Federal Institute of Technology Lausanne, 1015 Lausanne, Switzerland (e-mail: tobias.loew@idiap.ch; sylvain.calinon@idiap.ch).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TRO.2023.3277282.

Digital Object Identifier 10.1109/TRO.2023.3277282

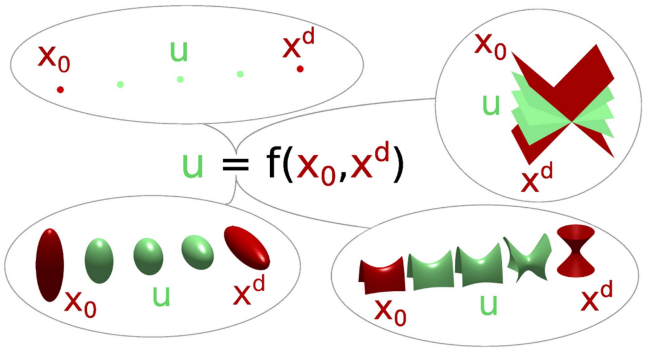


Fig. 1. Generic cost function in geometric algebra can consider different geometric primitives without changing its structure.

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 GA, which adds a structural check for the validity. These properties were some fundamental criteria in the design of GA, along with the possibility to formulate basic equations in a coordinate-free manner and to smoothly transfer information between formalisms [4].

We want to show the versatility of GA and that it can be used as a single tool to solve a variety of problems due to its unification of concepts. Simultaneously, we want to make GA more accessible by enabling the usage of standard tools and solvers for minimizing cost functions that are expressed in GA. Our contributions are as follows.

1) We extend the Lagrangian dynamics of serial manipulator in CGA to include a nontrivial inertia tensor and subsequently use the derived dynamics in an inverse dynamics control scheme.
2) We propose the usage of GA to define objective functions for optimizations as they appear in inverse kinematics and optimal control problems for manipulation tasks and show the modeling of different geometric relations in an optimization problem, while keeping the structure of the cost function uniform.
3) We demonstrate how GA formulations can be seamlessly used within existing frameworks based on linear matrix algebra by exploiting the sparsity of GA in order to facilitate its adoption.
4) We provide an open-source library called ***g**eometric **a**lgebra **f**or **ro**botics (gafro)* that implements all the presented formulations for robotics. The library is based on a fast and efficient custom implementation of CGA using expression templates. The benchmark shows faster computation of the kinematics than state-of-the-art robotics libraries.

The rest of this article is organized as follows. Section II presents the related work. Section III introduces GA in a formal manner. Section IV explains how GA can be used to compute the kinematics and dynamics of serial manipulators. Section V introduces GA for optimal control. Section VI then shows the experiments. The source code of the library as well as more information and accompanying videos can be found at https://tloew.gitlab.io/geometric_algebra/.

II. Related Work

The resurgence of geometric methods in robotics has spawned a variety of different approaches to formalize control, learning, and optimization problems in robotics. These methods include screw theory, Riemannian geometry, Lie algebra, and dual quaternions. A common motivation between these frameworks is the modeling of robot kinematics and dynamics, which is closely tied to representing rigid body transformations. GA essentially presents a generalization and unification of these concepts, and thus, it is naturally connected to a large variety of recent work in robotics research.

Conceptually, GA can be seen as an extension and generalization of dual quaternions [5], since dual quaternions can be identified with a certain Clifford algebra, which forms the foundation of GA. The literature on dual quaternions is, hence, the most closely related to our work. Starting with formulating rigid body transformations, dual quaternion algebra offers efficient ways for blending them, which is useful in computer graphics [6]. In robotics, there have been various works describing the kinematics and dynamics of robots in dual quaternion algebra such as [7] and [8] that combine the geometric understanding of screw theory, the thoroughness of Lie Algebra, and the simplicity of spatial algebra. This is also true for the motors of GA due to the close connection to dual quaternions. Efficient control is an important aspect for using real robots, and dual quaternions have been used to design an admittance controller [9] and a linear–quadratic regulator controller for trajectory tracking [10]. Collision avoidance is an important aspect of control, and in [11], vector field inequalities based on dual quaternions were proposed to handle them during surgical tasks.

Along with the regained interest of using geometric methods in robotics came various works proposing the use of differential and Riemannian geometry for learning and optimization problems. The topic of learning from demonstration often requires data to be represented as distributions; thus, Calinon [12] presented how to use Gaussians on Riemannian manifolds. The manifold of semi-positive-definite matrices has been used to study manipulability ellipsoids for learning robot skills [13]. In [14], a Riemannian metric was proposed that helps manipulators avoid singularities. Riemannian optimization is also used to solve the inverse kinematics problem of kinematic chains using distance geometry [15].

GA has been applied successfully in a variety of different applications and fields. For example, in the field of computer

graphics, which started to repopularize it, it found applications in mesh deformation [16] as well as ray casting and surface representation [17]. In the domain of image processing, techniques for adaptive filtering have been devised [18], [19].

A popular example in robotics to show the strengths of GA is solving the inverse kinematics problem. There have been various methods that proposed to utilize the geometric primitives and their intersection, such as FABRIK [20], which finds an iterative solution and has also been extended to include model constraints [21]. Recently, another extension, called FABRIKx [22], has been proposed to address the inverse kinematics problem of continuum robots. A similar approach that finds a closed-form solution using the geometric primitive intersection has been described in [23], while Bayro-Corrochano and Zamora-Esquivel [24] presented the differential and inverse kinematics of robots using CGA, and an iterative inverse kinematics solution was derived in [25]. Recent work has approached the topic of formulating constrained dynamics in CGA [26]. Newton–Euler modeling has been proposed for multicopters using motor algebra in [27] and for robot control using CGA in [28]. The interpolation of motors, i.e., rigid body transformations, has been shown to have useful applications in surgical robotics to model and plan surgical paths using virtual reality [29]. CGA was presented for robust pose control of manipulators in [30], and Zamora-Esquivel and Bayro-Corrochano [31] used it for robot object manipulation. Building on this work, we develop an optimal control framework that allows the formulation of various geometric primitives as task objectives via a generic cost function. Our extension of the dynamics then enables this optimal control framework to be used in model-predictive control (MPC) fashion for a torque-controlled robot.

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 GA from scratch using expression templates. There have been various works that published implementations of GA, such as GATL [32], GARAMON [33], Gaigen [34], TbGAL [35], GAL [36], and Versor [37]. These libraries all have in common that they are meant to be generic GA 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 [38] library, but using the more general CGA as opposed to dual quaternions.

III. Geometric Algebra

In this section, we will give a brief introduction to GA with a focus on the specific variant known as CGA. We will use the following notation throughout this article: x denotes scalars, $\boldsymbol{x}$ denotes vectors, $\boldsymbol{X}$ denotes matrices, X denotes multivectors, and $\boldsymbol{\mathcal{X}}$ denotes matrices of multivectors.

GA is a single algebra for geometric reasoning, alleviating the need of utilizing multiple algebras to express geometric relations [39]. The core idea of GA 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 [40].

The resulting algebra essentially includes $\mathbb{R}$ and the subspaces of the associated vector space as elements of computations [41]. Hence, let $\mathbb{R}^{p,q,r}$ be a vector space, where p, q, and r are the number of basis vectors that square to $1, -1$, and 0, respectively, i.e., the dimension of this vector space is $n = p + q + r$. The associated GA $\mathbb{G}_{p,q,r}$ then has $2^n = 2^{p+q+r}$ basis elements, which are called blades. A general element in GA is called a multivector and is the linear combination of basis blades. 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. Common variants of geometric include motor algebra $\mathbb{G}^+_{3,0,1}$, projective GA $\mathbb{G}_{3,0,1}$, and CGA $\mathbb{G}_{4,1,0}$ [42]. Dual quaternions can be identified with the Clifford algebra $Cl^+_{0,3,1}$ [43], which means they are based on an algebra with a degenerate metric. Due to this, many of the operations that we are presenting in this article are not possible in dual quaternion algebra.

In this article we are using CGA [44]. Conformal refers to angle-preserving transformations. It embeds the 3-D Euclidean space $\mathbb{R}^3$ into the 5-D one $\mathbb{R}^{4,1}$. The corresponding GA $\mathbb{G}_{4,1}$ introduces two null vectors ($\boldsymbol{e}_0$ and $\boldsymbol{e}_\infty$) that essentially represent a point at the origin and a point at infinity. The 5-D space means that CGA has 32 basis blades. An explanation of that structure can be found in Appendix A.

A point $\boldsymbol{x}$ in Euclidean space $\mathbb{R}^3$ is embedded into CGA by using the conformal embedding, which is bijective, meaning that any point $\boldsymbol{x} \in \mathbb{R}^3$ can be uniquely identified with a point $X \in \mathbb{G}_{4,1}$

$$X = \mathcal{C}(\boldsymbol{x}) = \boldsymbol{e}_o + \boldsymbol{x} + \frac{1}{2}\boldsymbol{x}^2\boldsymbol{e}_\infty. \tag{2}$$

Points are the basic geometric primitives that can be used to construct others by the spanning operation of the outer product. These geometric primitives are, in general, null-space 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, i.e.,

$$\text{IPNS: } \mathbb{NI}_G(\boldsymbol{A}) = \left\{\boldsymbol{x} \in \mathbb{R}^3 : \mathcal{C}(\boldsymbol{x}) \cdot \boldsymbol{A} = 0\right\} \tag{3}$$

$$\text{OPNS: } \mathbb{NO}_G(\boldsymbol{A}) = \left\{\boldsymbol{x} \in \mathbb{R}^3 : \mathcal{C}(\boldsymbol{x}) \wedge \boldsymbol{A} = 0\right\}. \tag{4}$$

The IPNS and OPNS representations are connected by a duality relationship. Duality in this case means multiplication with the pseudoscalar I, the highest grade element of the algebra (i.e., $I = e_{0123\infty}$ for CGA)

$$X^* = IX. \tag{5}$$

We refer to the OPNS as the primal space for its more convenient usage, which consequently makes the IPNS the dual representation, although both representations can be used to represent all geometric primitives. In Fig. 2, we show the subspaces

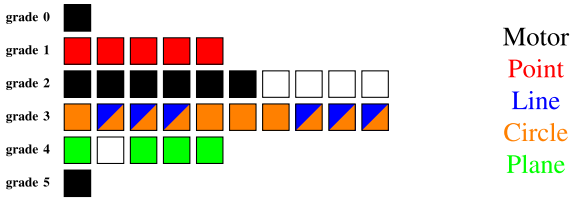


Fig. 2. Nonzero elements of various geometric primitives in their primal representations in CGA. Boxes represent basis blades and colored boxes represent the nonzero blades of the geometric primitive with the matching color. It can be seen that of the 32 basis blades, only a sparse number is used for the representations. Note that geometric primitives are single-grade objects, while transformations are mixed-grade objects. The boxes correspond to the basis blades that are shown in Table I in Appendix A.

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 nonzero elements of the expression, which is exploited in our implementation of GA. The primitives of CGA with their corresponding equations for construction can be found in Appendix B. These primitives can be extended when going to higher dimensional GAs; for example, $\mathbb{G}_{6,3}$ additionally introduces quadric surfaces like ellipsoids and hyperboloids [45], and the quadric CGA $\mathbb{G}_{9,3}$ allows using arbitrary quadric surfaces [46].

CGA provides an exception-free way of computing incidence relations between geometric objects [47]. This is achieved via the meet operator

$$Y = X_1 \vee X_2 = (X_1^* \wedge X_2^*)^*. \tag{6}$$

The resulting multivector retains a geometric meaning, e.g., when a line meets a sphere, there are three possibilities.

1) The line intersects the sphere, in which case Y is a point pair.
2) The line is tangential to the sphere, which results in a single point.
3) The line and the sphere are completely separate, resulting in an imaginary point that is related to the distance between the objects.

This geometric result is directly encoded in the result, and no special cases need to be considered.

There are several geometric operations available in GA, such as translations and rotations, but also dilations, reflections, projections, and rejections. For brevity, we only present rigid body transformations, i.e., translations and rotations, in this article. Thorough introductions can be found in [3] and [48] and a survey of relevant research in [5] and [49].

A. Rigid Body Motions in Geometric Algebra

The elements 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 (we omit other conformal operations such as scaling in this introduction to GA in order to keep it short), i.e.,

$$M = TR. \tag{7}$$

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

$$Y = MX\widetilde{M} \tag{8}$$

where $\widetilde{M}$ stands for the reverse of a motor, which can thought of as being similar to a conjugate quaternion.[1]

Both translators and rotors can be found with an exponential mapping of bivectors, i.e.,

$$T = \exp\left(\frac{1}{2}(\boldsymbol{t} \wedge \boldsymbol{e}_\infty)\right) = 1 - \frac{1}{2}\boldsymbol{t} \wedge \boldsymbol{e}_\infty \tag{9}$$

and

$$R = \exp\left(\frac{\theta}{2}B\right) = \cos\left(\frac{\theta}{2}\right) - \sin\left(\frac{\theta}{2}\right)B. \tag{10}$$

The bivectors are $\boldsymbol{t} \wedge \boldsymbol{e}_\infty \in \text{span}\{\boldsymbol{e}_{1\infty}, \boldsymbol{e}_{2\infty}, \boldsymbol{e}_{3\infty}\}$ and $B \in \text{span}\{\boldsymbol{e}_{23}, \boldsymbol{e}_{13}, \boldsymbol{e}_{13}\}$, respectively. Rotors can be seen as isomorphic to quaternions; they are, however, more general and do not require the introduction of complex numbers.

The motors in GA form a group, which is an even subalgebra $\mathbb{M}$ of $\mathbb{G}_{4,1}^+$

$$\mathbb{M} = \text{span}\left\{1, \boldsymbol{e}_{23}, \boldsymbol{e}_{13}, \boldsymbol{e}_{12}, \boldsymbol{e}_{1\infty}, \boldsymbol{e}_{2\infty}, \boldsymbol{e}_{3\infty}, \boldsymbol{I}_3\boldsymbol{e}_\infty\right\} \subset \mathbb{G}_{4,1}^+. \tag{11}$$

It forms a Lie group with an associated Lie algebra, which is the bivector algebra in the linear subspace $\mathbb{B}$ that is defined as

$$\mathbb{B} = \text{span}\left\{\boldsymbol{e}_{23}, \boldsymbol{e}_{13}, \boldsymbol{e}_{12}, \boldsymbol{e}_{1\infty}, \boldsymbol{e}_{2\infty}, \boldsymbol{e}_{3\infty}\right\} \subset \mathbb{M}. \tag{12}$$

Since the motor group is a Lie group, it is also a smooth manifold. The motor manifold $\mathcal{M}$ can be found with the group constraint

$$\mathcal{M} = \left\{M \in \mathbb{M} : M\widetilde{M} = 1\right\}. \tag{13}$$

Motors are isomorphic to dual quaternions [48], which makes them also isomorphic to $SE(3)$. They represent, however, a more general concept of transformations that is valid in any dimension. Furthermore, due to their similarities with dual quaternions, the same advantages over transformation matrices apply to motors as well, i.e., they require less memory and operations for multiplication compared to transformation matrices [50]. The motor manifold being a Lie group consequently makes the bivector algebra its Lie algebra. The operation that connects the motor manifold with the bivector algebra is the exponential map, and its inverse the logarithmic map

$$M = \exp(B) \quad \Longleftrightarrow \quad B = \log(M). \tag{14}$$

The exponential and logarithmic maps are the Lie group operators that move an element from the bivector Lie algebra $\mathbb{B}$ (equivalent to $\mathfrak{se}(3)$) to the motor manifold Lie group $\mathcal{M}$ (equivalent to $SE(3)$) and *vice versa*. The interpretation of the bivector is the representation of a dual line $B = L^*$. This essentially defines the screw axis of the motor. Note that L^* is the IPNS of a line and as such a 2-blade, i.e., a bivector [51]. The motor manifold, therefore, elegantly combines position and orientation. Having both position and orientation represented by one entity removes

[1]Note the similarities with the operations using conjugate quaternion in quaternion algebra.

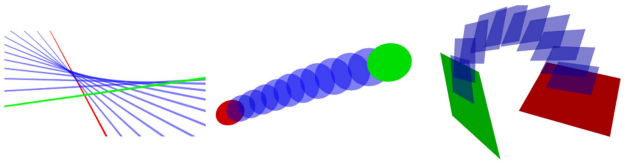


Fig. 3. Rigid body transformations of various geometric primitives. Red marks the initial primitive and green the final one; the primitives resulting from the trajectory of interpolated motors are shown in blue.

the need to switch between algebras for computation, e.g., linear and quaternion algebra. Using the motor, manifold optimization problems can be solved, as presented in [52].

Since the motors and the geometric primitives are part of the same algebra, the motors can be used to apply rigid body transformations to these primitives. This means that the primitives can be translated, rotated, reflected, and scaled using angle-preserving transformations. Some visual examples of how motors are transforming geometric primitives are shown in Fig. 3.

It has been shown in [29] that efficient interpolation between motors can be achieved via the bivector space, which is similar to spherical linear interpolation (SLERP). In this case, the parameterized motor curve $M(t)$ can be found as an interpolation in the bivector space of viapoint motors using the exponential and logarithmic map

$$M(t) = \exp\left(\sum_{j=1}^{n} w_j(t)\log(M_j)\right). \tag{15}$$

The weights need to fulfill $\sum_{j=1}^{n} w_j = 1$ for each time step, but can otherwise be chosen arbitrarily. In [16], this is exploited for mesh deformation.

IV. Geometric Algebra for Serial Manipulators

In this section, we present how GA can be used to express the kinematics and dynamics of serial manipulators. We do this while also explicitly drawing connections to the expressions and terminology of classical linear algebra.

A. Forward Kinematics of Serial Manipulators

The forward kinematics of a kinematic chain of N joints can easily be defined using motors [53]. Assuming that we only have revolute joints, the forward motor $M(\boldsymbol{q})$, given the configuration $\boldsymbol{q}$, can be computed with

$$M(\boldsymbol{q}) = \prod_{i=1}^{N} M_i(q_i) = \prod_{i=1}^{N} M_{F,i}R_i(q_i). \tag{16}$$

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

$$R_i(q_i) = \exp\left(-\frac{1}{2}q_iB_i\right) \tag{17}$$

where the bivectors B_i essentially represent the rotation planes of the joints. These quantities can easily be found using, e.g., Denavit–Hartenberg parameters [54].

We denote by $M^k(\boldsymbol{q})$ the forward kinematics up the kth joint, i.e.,

$$M^k(\boldsymbol{q}) = \prod_{i=1}^{k} M_i(q_i) = \prod_{i=1}^{k} M_{F,i}R_i(q_i). \tag{18}$$

B. Jacobians of Serial Manipulators

In the literature about serial kinematic chains, one can generally find the distinction between two Jacobians: the geometric and the analytic Jacobian. In this section, we will explain how these quantities translate to GA.

Using an arbitrary representation of the end-effector forward kinematics

$$\boldsymbol{\xi} = \boldsymbol{f}(\boldsymbol{q}) \tag{19}$$

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

$$\boldsymbol{J}^A(\boldsymbol{q}) = \frac{\partial \boldsymbol{f}(\boldsymbol{q})}{\partial \boldsymbol{q}} \tag{20}$$

and it relates the joint angle velocity to time derivatives of the end-effector configuration using the given representation

$$\dot{\boldsymbol{\xi}} = \boldsymbol{J}^A(\boldsymbol{q})\dot{\boldsymbol{q}}. \tag{21}$$

The geometric Jacobian, on the other hand, defines the relationship of the joint angle velocity to the linear and angular velocity of the end-effector in a certain coordinate frame

$$\begin{bmatrix}\boldsymbol{v}\\ \boldsymbol{\omega}\end{bmatrix} = \boldsymbol{J}^G(\boldsymbol{q})\dot{\boldsymbol{q}}. \tag{22}$$

The relationship between the analytic and the geometric Jacobians can be found by a representation specific mapping

$$\boldsymbol{J}^G(\boldsymbol{q}) = \boldsymbol{J}^M(\boldsymbol{\xi})\boldsymbol{J}^A(\boldsymbol{q}). \tag{23}$$

In GA, the analytic Jacobian can be found as the derivative of the forward kinematics motor defined in (16), i.e.,

$$\boldsymbol{\mathcal{J}}^A(\boldsymbol{q}) = \frac{\partial M(\boldsymbol{q})}{\partial \boldsymbol{q}} = \left[\frac{\partial M(\boldsymbol{q})}{\partial q_1} \dots \frac{\partial M(\boldsymbol{q})}{\partial q_N}\right]. \tag{24}$$

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. The partial derivative of the forward motor w.r.t the ith joint angle is

$$\frac{\partial M(\boldsymbol{q})}{\partial q_i} = M_1(q_1)\dots M_{F,i}\left(-\frac{1}{2}B_i\right)R_i(q_i)\dots M_N(q_N). \tag{25}$$

Similarly, the geometric Jacobian of a serial kinematic chain in GA can be found by transforming the rotation bivectors of each joint using the respective motor, i.e.,

$$\boldsymbol{\mathcal{J}}_j^G(\boldsymbol{q}) = \begin{bmatrix}B_1' & \dots & B_j' & \mathbf{0}\end{bmatrix} \tag{26}$$

with the rotation bivectors

$$B_i' = M_1\dots M_{i-1}M_{F,i}B_i\widetilde{M}_{F,i}\widetilde{M}_{i-1}\dots\widetilde{M}_1. \tag{27}$$

In this case, the size of the multivector matrix is $\boldsymbol{\mathcal{J}}^G(\boldsymbol{q}) \in \mathbb{B}^{1\times N} \subset \mathbb{G}_{4,1}^{1\times N}$. Its elements correspond to bivectors.

From (25) and (27), the relationship between the analytic and geometric Jacobians in GA can easily be derived as

$$\mathcal{J}^G_{kj}(\boldsymbol{q}) = -2\mathcal{J}^A_{kj}(\boldsymbol{q})\widetilde{M}(\boldsymbol{q}). \tag{28}$$

For the computation of the dynamics, the time derivative of the geometric Jacobian is also required, and it can be found as

$$\dot{\boldsymbol{\mathcal{J}}}^G(\boldsymbol{q},\dot{\boldsymbol{q}}) = \begin{bmatrix} \dot{B}'_1 & 0 & 0 & \cdots & 0 \\ \vdots & \dot{B}'_2 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \dot{B}'_1 & \dot{B}'_2 & \dot{B}'_3 & \cdots & \dot{B}'_N \end{bmatrix}. \tag{29}$$

The required time derivatives of the rotation bivectors $\dot{B}'_j$ can be found using

$$\dot{\boldsymbol{\mathcal{B}}}'(\boldsymbol{q},\dot{\boldsymbol{q}}) = \begin{bmatrix} \dot{B}'_1 \\ \vdots \\ \dot{B}'_N \end{bmatrix} = \boldsymbol{\mathcal{J}}^{\times}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{30}$$

with

$$\mathcal{J}^{\times}_{ij}(\boldsymbol{q}) = B'_i \times B'_j = \frac{1}{2}(B'_iB'_j - B'_jB'_i) \tag{31}$$

where the operator $\times$ is called the commutator product.

C. Inverse Kinematics of Serial Manipulators

Using the expressions derived in Section IV-B, the inverse kinematics problem for a serial kinematic chain can be formulated as an optimization problem on the motor manifold. The goal is to find the joint angles $\boldsymbol{q}$ that minimize the following equation:

$$\boldsymbol{q}^* = \arg\min_{\boldsymbol{q}} \left\| \log\left(\widetilde{M}_{\text{target}} M(\boldsymbol{q})\right) \right\|_2^2. \tag{32}$$

The forward kinematics motor $M(\boldsymbol{q})$ can be found using (16). The expression $\widetilde{M}_2 M_1$ can be understood as the shortest screw motion between two points on the motor manifold. The $\log(\cdot)$ operation moves the problem to Bivector space, i.e., the Lie algebra of the motor manifold.

The Jacobian can be found as

$$\boldsymbol{J}_{\mathbb{B}}(\boldsymbol{q}) = \mathcal{E}^{\mathbb{B}\to\mathbb{R}^6}\left[\frac{\partial}{\partial q_i}\log\left(\widetilde{M}_{\text{target}} M(\boldsymbol{q})\right)\right]. \tag{33}$$

Here, $\boldsymbol{J}_{\mathbb{B}}(\boldsymbol{q}) \in \mathbb{R}^{6\times N}$ is a linear algebra matrix. The interpretation of $\boldsymbol{J}_{\mathbb{B}}(\boldsymbol{q})$ is an embedding of the multivectors into a matrix algebra and exploiting their sparsity. The expression can be further untangled into

$$\boldsymbol{J}_{\mathbb{B}}(\boldsymbol{q}) = \boldsymbol{J}_{\mathcal{M}\to\mathbb{B}}(\boldsymbol{q})\boldsymbol{J}_{\mathcal{M}}(\boldsymbol{q}). \tag{34}$$

$\boldsymbol{J}_{\mathcal{M}}(\boldsymbol{q}) \in \mathbb{R}^{8\times N}$ is the embedding of the analytic Jacobian of (24) multiplied by the target motor, i.e.,

$$\boldsymbol{J}_{\mathcal{M}}(\boldsymbol{q}) = \mathcal{E}^{\mathcal{M}\to\mathbb{R}^8}\left[\widetilde{M}_{\text{target}}\boldsymbol{\mathcal{J}}^A(\boldsymbol{q})\right]. \tag{35}$$

$\boldsymbol{J}_{\mathcal{M}\to\mathbb{B}}(\boldsymbol{q}) \in \mathbb{R}^{6\times 8}$ can be understood as the Jacobian of the local parameterization from the motor manifold to the bivector space and, hence, is the Jacobian of the $\log(\cdot)$ operation. The derivation of $\boldsymbol{J}_{\mathcal{M}\to\mathbb{B}}$ can be found in Appendix D.

With (33), the Gauss–Newton step becomes

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k - \alpha\left(\boldsymbol{J}_{\mathbb{B}}(\boldsymbol{q})^{\top}\boldsymbol{J}_{\mathbb{B}}(\boldsymbol{q})\right)^{-1}\boldsymbol{J}_{\mathbb{B}}(\boldsymbol{q})^{\top}\boldsymbol{f}(\boldsymbol{q}_k) \tag{36}$$

where α is the line-search parameter. This shows how GA functions on the motor manifold can be optimized using classical methods by embedding the multivectors into a matrix algebra, which will be exploited later when defining the cost functions for optimal control problems in GA.

D. Dynamics of Serial Manipulators

In classical linear algebra, the manipulator equation is found to be

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau} - \boldsymbol{\tau}_{\text{ext}} \tag{37}$$

where $\boldsymbol{M}(\boldsymbol{q})$ is known as the inertia or generalized mass matrix, $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ is representing Coriolis/centrifugal forces, $\boldsymbol{g}(\boldsymbol{q})$ stands for the gravitational forces, $\boldsymbol{\tau}$ is the vector of joint torques, and $\boldsymbol{\tau}_{\text{ext}}$ are the external torques.

In GA, (37) can be transformed to a simplified version, which was shown in [39]. The influence of the link inertia in that work, however, was assumed to be a scalar constant, which, of course, is not accurate for real systems. Therefore, we extend this equation by a joint position dependent inertia tensor and subsequently derive the necessary influence on the Coriolis/centrifugal forces. We first present the GA reformulation of (37) that was derived in [39] and then present our extension with the joint position-dependent inertia tensor. The elements of (37) can be expressed as multivector matrices, where the generalized mass matrix becomes

$$\boldsymbol{\mathcal{M}}(\boldsymbol{q}) = \boldsymbol{\mathcal{I}}(\boldsymbol{q}) + \boldsymbol{\mathcal{V}}^{\top}(\boldsymbol{q})\boldsymbol{m}\boldsymbol{\mathcal{V}}(\boldsymbol{q}). \tag{38}$$

The scalar valued $\boldsymbol{m}$ is an $N \times N$ matrix that contains all link masses along its diagonal. The Coriolis/centrifugal forces become

$$\boldsymbol{\mathcal{C}}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} = \dot{\boldsymbol{\mathcal{I}}}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{\mathcal{V}}^{\top}(\boldsymbol{q})\boldsymbol{m}\dot{\boldsymbol{\mathcal{V}}}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} \tag{39}$$

and the gravitational forces are

$$\boldsymbol{\mathcal{G}}(\boldsymbol{q}) = \boldsymbol{\mathcal{V}}^{\top}(\boldsymbol{q})\boldsymbol{m}\boldsymbol{\mathcal{G}}. \tag{40}$$

The constant matrix $\boldsymbol{\mathcal{G}} \in \mathbb{G}_{4,1}^{N\times 1}$ contains the gravitational acceleration with the information about the direction. In the usual case, all the elements, therefore, are equal to $g\boldsymbol{e}_3$. The recurring multivector matrix $\boldsymbol{\mathcal{V}}(\boldsymbol{q}) \in \mathbb{G}_{4,1}^{N\times N}$ can be found using the current centers of mass of the links X_j^{CoM} and the current axes of rotation of the joints, expressed as bivectors B'_k. An element of this matrix, therefore, becomes

$$\mathcal{V}_{j,k}(\boldsymbol{q}) = X_j^{\text{CoM}}(\boldsymbol{q}) \cdot B'_k. \tag{41}$$

The interpretation of this matrix is the computation of the lever arms of the centers of mass of the links w.r.t. each joint. Its time

derivative can be found to be

$$\dot{\boldsymbol{\mathcal{V}}}_{j,k}(\boldsymbol{q},\dot{\boldsymbol{q}}) = (\boldsymbol{I}\boldsymbol{\mathcal{V}}(\boldsymbol{q})\dot{\boldsymbol{q}})\,\boldsymbol{\mathcal{J}}^{G}(\boldsymbol{q}) + \left(\boldsymbol{I}\boldsymbol{\mathcal{X}}^{CoM}(\boldsymbol{q})\right)\dot{\boldsymbol{\mathcal{J}}}^{G}(\boldsymbol{q},\dot{\boldsymbol{q}}). \tag{42}$$

Note that $\boldsymbol{I}$ in this case is an $N \times N$ identity matrix, such that the expression $\boldsymbol{\mathcal{V}}(\boldsymbol{q})\dot{\boldsymbol{q}}$ becomes a square matrix instead of a vector. The same applies to the expression $\boldsymbol{I}\boldsymbol{\mathcal{X}}^{\text{CoM}}(\boldsymbol{q})$, where the matrix $\boldsymbol{\mathcal{X}}^{\text{CoM}}(\boldsymbol{q}) \in \mathbb{G}_{4,1}^{N\times 1}$ contains all centers of mass of the links.

As mentioned, we are not assuming the inertia to be constant in this article, since we want to use the inverse dynamics control scheme in our experiments, which requires an exact computation. Therefore, we have derived the influence of the link inertia, given the current joint state $\boldsymbol{\mathcal{I}}(\boldsymbol{q})$, as well as its time derivative $\dot{\boldsymbol{\mathcal{I}}}(\boldsymbol{q},\dot{\boldsymbol{q}})$. The inertia matrix can be found as a summation over the joint angles of the manipulator, accounting for the influence of each joint, i.e.,

$$\boldsymbol{\mathcal{I}}(\boldsymbol{q}) = \sum_{i=0}^{N} \boldsymbol{\mathcal{B}}_i^{\top}\mathcal{I}(\boldsymbol{\mathcal{B}}_i). \tag{43}$$

The bivector matrix $\boldsymbol{\mathcal{B}}_i$ is a $1 \times N$ row vector and contains the rotation generators of each joint w.r.t the current joint. The jth element of $\boldsymbol{\mathcal{B}}_i$ can, thus, be found as

$$B_{i,j} = \widetilde{R}_i(\boldsymbol{q}) \log^R\left(\boldsymbol{\mathcal{J}}_{ij}^{G}(\boldsymbol{q})\right) R_i(\boldsymbol{q}). \tag{44}$$

The expression $\log^R(\cdot)$ in these equations stands for the logarithmic map of the rotor part of the motor that is the ijth element of the geometric Jacobian. It, hence, returns a bivector with nonzero elements corresponding to the basis blades $\boldsymbol{e}_{23}$, $\boldsymbol{e}_{13}$, and $\boldsymbol{e}_{12}$.

Thus, the time derivative of the inertia matrix is

$$\dot{\boldsymbol{\mathcal{I}}}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \sum_{i=0}^{N} \boldsymbol{\mathcal{B}}_i^{\top}\left(\mathcal{I}(\dot{\boldsymbol{\mathcal{B}}}_i) + \widetilde{R}_i(\boldsymbol{q})\hat{B}_i^{w}R_i(\boldsymbol{q})\right). \tag{45}$$

The time derivative of the rotation generators $\dot{\boldsymbol{\mathcal{B}}}_i$ can be found in the same way as the elements of $\boldsymbol{\mathcal{B}}_i$, but using the time derivative of the geometric Jacobian

$$\dot{B}_{i,j} = \widetilde{R}_i(\boldsymbol{q}) \log^R\left(\dot{\boldsymbol{\mathcal{J}}}_{ij}^{G}(\boldsymbol{q},\dot{\boldsymbol{q}})\right) R_i(\boldsymbol{q}). \tag{46}$$

The variable $\hat{B}_i^{w}$ from (45) can be found as

$$\hat{B}_i^{w} = (I_3 B_i^{w}) \wedge \left(R_i(\boldsymbol{q})\mathcal{I}\left(\widetilde{R}_i(\boldsymbol{q})I_3 B_i^{w} R_i(\boldsymbol{q})\right)\widetilde{R}_i(\boldsymbol{q})\right). \tag{47}$$

Note that in this case, I_3 stands for $\boldsymbol{e}_{123}$, which is the pseudoscalar of the Euclidean GA $\mathbb{G}_3$, which is a subalgebra of CGA. B_i^{w}, on the other hand, is the bivector velocity that results from multiplying the geometric Jacobian with the joint velocity, i.e., $B_i^{w} = \boldsymbol{\mathcal{J}}_i^{G}(\boldsymbol{q})\dot{\boldsymbol{q}}$. The quantity $I_3 B_i^{w}$, therefore, is the angular velocity and is nonzero in $\boldsymbol{e}_1, \boldsymbol{e}_2$, and $\boldsymbol{e}_3$. The outer product in (47) causes the quantity $\hat{B}_i^{w}$ to be a bivector again, i.e., the elements $\boldsymbol{e}_{23}, \boldsymbol{e}_{13}$, and $\boldsymbol{e}_{12}$ are nonzero.

In all the above equations, $\mathcal{I}(\cdot)$ expresses the inertia tensor being applied to a multivector or to each multivector element in the matrix case. The inertia tensor is a grade-preserving operation since it maps bivectors to bivectors [55].

Finally, we find the manipulator inverse dynamics equation in GA to be

$$\boldsymbol{\tau}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}) = \boldsymbol{\tau}_{ext} + \boldsymbol{\mathcal{I}}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{\mathcal{I}}}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{\mathcal{V}}^{\top}(\boldsymbol{q})\boldsymbol{m}\left(\boldsymbol{\mathcal{V}}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{\mathcal{V}}}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{\mathcal{G}}\right). \tag{48}$$

Consequently, the forward dynamics of a serial manipulator can be expressed as

$$\ddot{\boldsymbol{q}}(\boldsymbol{q},\dot{\boldsymbol{q}},\boldsymbol{\tau}) = \left(\boldsymbol{\mathcal{I}}(\boldsymbol{q}) + \boldsymbol{\mathcal{V}}^{\top}(\boldsymbol{q})\boldsymbol{m}\boldsymbol{\mathcal{V}}(\boldsymbol{q})\right)^{-1} \left(\boldsymbol{\tau} - \boldsymbol{\tau}_{\text{ext}} - \boldsymbol{\mathcal{V}}^{\top}(\boldsymbol{q})\boldsymbol{m}\left(\dot{\boldsymbol{\mathcal{V}}}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{\mathcal{G}}\right) - \dot{\boldsymbol{\mathcal{I}}}(\boldsymbol{q},\dot{\boldsymbol{q}})\right). \tag{49}$$

In the experiments, we will validate these equations by employing an inverse dynamics control scheme on top of the MPC in order to convert the acceleration commands to torques.

V. Geometric Algebra for Optimal Control

In this section, we describe how GA can be used in optimal control problems. We first present a brief review of optimal control, then present its application to a point mass system, and finally show how it can be used for serial manipulators.

A. Optimal Control

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. Modeling these mathematically requires special care, since they will determine the quality of the resulting solution. Furthermore, optimal control can be applied as solver to an MPC problem, which requires fast convergence in order to achieve acceptable real-time control rates. We will show that the use of GA for geometric primitives improves the clarity of equations and, thus, reduces computational difficulties. The modeling 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

$$\min_{\boldsymbol{u}} L(\boldsymbol{x},\boldsymbol{u}) = l_f(\boldsymbol{x}_N) + \sum_{k=1}^{N-1} l_k(\boldsymbol{x}_k) + \|\boldsymbol{u}\|_R^2 \tag{50}$$

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. A popular method to solve this problem is the iterative linear–quadratic regulator (iLQR) [56]. It solves the problem by linearizing the nonlinear system around the current solution and by assuming a quadratic cost. The solution is then refined iteratively until convergence. We will be using iLQR to solve the problem in an MPC fashion in the experiments. This means that we are solving the regulation problem at each time step and apply only the first control command to the robot.

B. Optimal Control on the Motor Manifold

Employing homogeneous coordinates in 4-D GA effectively allows us to linearize rigid body motions in 3-D Euclidean space. In order to exploit this useful property, we demonstrate how to solve reaching tasks for rigid body motion using the motor manifold. The linear system is defined in the linear 6-D bivector space, i.e., the Lie algebra of the motor manifold. The state $\boldsymbol{x}$ is defined to be the stacked vector of the parameter vectors $\boldsymbol{b}$ and $\dot{\boldsymbol{b}}$ of the bivector B and its time derivative $\dot{B}$, respectively. From this follows the definition of the linear dynamical system as

$$\boldsymbol{x}_{t+1} = \begin{bmatrix} \boldsymbol{b}_{t+1} \\ \dot{\boldsymbol{b}}_{t+1} \end{bmatrix} = \boldsymbol{A} \begin{bmatrix} \boldsymbol{b}_t \\ \dot{\boldsymbol{b}}_t \end{bmatrix} + \boldsymbol{C}\boldsymbol{u}_t \tag{51}$$

where the command $\boldsymbol{u}$ corresponds to bivector accelerations.

Using the optimal control formulation that was presented in (50), we now need to define appropriate state costs based on GA. Naturally, the inverse kinematics cost function that was presented in Section VI-C can be used in order to define pose targets for reaching motions. This cost function is, therefore, the most equivalent to classical methods using, e.g., transformation matrices. Of course, when using this linear bivector system, the current motor M_t is not found using the forward kinematics, but instead using the exponential map, i.e.,

$$M_t = \exp(B(\boldsymbol{b}_t)). \tag{52}$$

The corresponding cost function, therefore, becomes

$$l(\boldsymbol{x}_t) = \left\|\boldsymbol{e}(\boldsymbol{x}_t)\right\|_2^2 = \left\| \log\left(\widetilde{M}_{\text{target}} M_t\right) \right\|_2^2. \tag{53}$$

Note that due to the logarithmic map that is used here, the error vector $\boldsymbol{e}(\boldsymbol{x}_t)$ is the parameter vector of a bivector and is, hence, 6-D as well. Since the motors include orientation as well as position, it essentially is an oriented point mass system.

Apart from defining target poses using the motor manifold, GA additionally offers the possibility to define targets using its geometric primitives and the accompanying incidence relationships. We exploit the null-space representations of the primitives for the formulation of the reaching objectives; more specifically, we use the OPNS representation. By definition of the OPNS, the outer product is zero for any point that is on a geometric primitive. The multivector valued error can, therefore, be defined as

$$E(\boldsymbol{q}) = X_d \wedge M_t X \widetilde{M}_t \tag{54}$$

with $X = \boldsymbol{e}_0$, i.e., the point at the origin, in this case, and X_d can be any geometric primitive that can be expressed in the algebra. It is important to highlight that other combinations are possible as well, and X is not restricted to be a point; it can, for example, also be a line with X_d being a point, which will be shown later in the form of a pointing task for a manipulator. Note that the structure of equation remains the same regardless of the combination of primitives.

C. Optimal Control for Serial Manipulators

In this section, we present the formulation of objective functions for optimal control problems with serial manipulators based on GA. Similarly to the previous section, the inverse kinematics cost function can be used to define target poses for a manipulator to reach. More interesting is to consider (54) for manipulators; of course, in this case, the motor again corresponds to the forward kinematics function. Using $X = \boldsymbol{e}_0$, therefore, means that the expression $M(\boldsymbol{q})X\widetilde{M}(\boldsymbol{q})$ corresponds to the tip of the end-effector. X_d again is free to be any geometric primitive. In all cases, the Jacobian can be found by applying the chain rule to the multivector expressions

$$\boldsymbol{\mathcal{J}}^E(\boldsymbol{q}) = X_d \wedge \left(\boldsymbol{\mathcal{J}}^A(\boldsymbol{q}) X \widetilde{M}(\boldsymbol{q}) + M(\boldsymbol{q}) X \widetilde{\boldsymbol{\mathcal{J}}^A}(\boldsymbol{q})\right) \tag{55}$$

where $\boldsymbol{\mathcal{J}}^A(\boldsymbol{q})$ is the analytic Jacobian that was presented in (24), and the reverse of a multivector matrix is defined as the elementwise multivector reverse.

Of course, depending on the combination of X and X_d, the resulting $E(\boldsymbol{q})$ will represent a different geometric meaning, which can be seen by the different nontrivial blades it holds. It is, however, known *a priori* what the resulting nontrivial blades are. From this, it follows that the embedding function $\mathcal{E}$ actually becomes dependent on X and X_d, i.e.,

$$\boldsymbol{J}^E(\boldsymbol{q}) = \mathcal{E}(X, X_d)\left[\boldsymbol{\mathcal{J}}^E(\boldsymbol{q})\right]. \tag{56}$$

The purpose of the embedding function, thus, is the removal of the trivial blades of the multivector, i.e., removing the zero rows from the matrix. This is in line with the goal of keeping the representations compact to allow for efficient computation. Furthermore, the embedding now allows the usage of off-the-shelf tools for optimal control.

Note that, in general, no special cases, such as division by zero, need to be considered here. The exception-free incidence property of CGA allows for the equations to be coded exactly as they are presented in this article.

VI. Experiments

In this section, we are presenting implementation details of the provided library *gafro* as well as benchmarks of the kinematics computation. Afterward, we show various experiments with the Franka Emika robot to demonstrate how GA can used to model different tasks.

A. 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 present 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 CGA. It exploits the sparsity of the multivector by only storing the data blades that are nonzero by the structure of the objects. The geometric, inner, and outer products are implemented as expression templates, which further exploit this structure by only evaluating the elements of the resulting type that are known to be nonzero. The types are evaluated at compile time and the

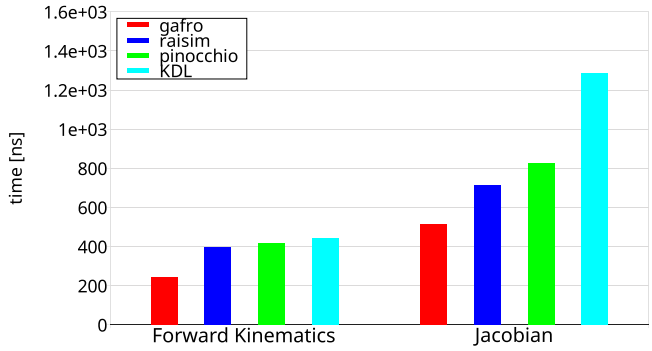


Fig. 4. Benchmarking results for *gafro* compared to Raisim, Pinocchio, and KDL. The benchmarks were all performed on an AMD Ryzen 7 4800U CPU using the compiler flags `-O3 -msse3 -march=native`. The presented results are the average of 10 000 executions with ten repetitions.

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 compare and benchmark *gafro* against several libraries that are commonly used in robotics applications. These libraries include Raisim [57], Pinocchio [58], and KDL [59]. An excerpt of the benchmarking results can be found in Fig. 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. Note that previous publications have already shown the advantages that dual quaternions have over homogeneous transformation matrices [60] in terms of computational complexity. Due to the close relationship that CGA motors have with dual quaternions, the same advantages apply to them as well. In [60], a 30–40% improvement in performance of dual quaternions compared to homogeneous transformation matrices was reported, which is similar to our findings for motors.

We want to point out that these benchmarks are preliminary results only, since there are some code optimizations that still need to be done. This especially applies to the computation of the dynamics that was presented in this article. While it is possible to do real-time control with our implementation, it is still naive in the sense that algorithmic improvements of the implementation will make the computation faster. Both these issues will be addressed in future work that will be dedicated to the implementation details as well as an in-depth benchmarking against a wider range of libraries. At this point, we will then also provide python bindings for the library.

B. Torque Control of Serial Manipulators

The optimal control methods derived in Section V-A are used in an MPC framework. In order to achieve fast online computation, we employ a double integrator system in the joint space, which results in acceleration commands for the control of the manipulator. However, standard practice is using torque commands, which means that we have to convert the accelerations to torques. This can be realized using an inverse dynamics controller; the required control command can, thus, be found as

$$\boldsymbol{u}_\tau = \boldsymbol{\tau}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}_d) + \boldsymbol{K}_p(\boldsymbol{q}_d - \boldsymbol{q}) + \boldsymbol{K}_d(\dot{\boldsymbol{q}}_d - \dot{\boldsymbol{q}}) \tag{57}$$

where the torque vector $\boldsymbol{\tau}$ is computed as presented in (48). $\boldsymbol{K}_p$ and $\boldsymbol{K}_d$ are the stiffness and damping gains, respectively.

While the formulation of the inverse dynamics controller follows standard practice, the computation of the torques $\boldsymbol{\tau}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}_d)$ is done using the GA approach that was presented in Section IV-D. The experiments on the real robot, therefore, not only show that GA can be used for tracking different geometric primitives online with MPC, but also verify the dynamics computation in GA. Most importantly, they validate the nontrivial inertia matrix that we derived in this article.

C. Inverse Kinematics

In order to evaluate the numerical inverse kinematics using the motor formulation, we repeated the following experiment 10 000 times. We sampled a random target from within the workspace of the Franka Emika robot and an initial joint configuration. Then, using the standard Gauss–Newton approach that we also described in Section IV-C, we computed the optimal solution. The solver success rate was 85.39% with a tolerance of $1e^{-6}$. The resulting final cost was in the order of 1×10^{-10} on average and was found within 11.2 iterations, which corresponds to a time of 79 μs on our system. We considered only the successful solver for these statistics. Note that this inverse kinematics solution presented in this article is meant as an explanatory example and proof of concept. It is not meant to compete in this form with existing IK solvers, but rather it should motivate to integrate GA into them. A potential future work could, therefore, be augmenting state-of-the-art solvers like TRAC-IK [61] with GA. Currently, TRAC-IK uses KDL in its implementation to calculate the forward kinematic chain. Based on our benchmarks, using *gafro* instead of KDL would lead to a significant increase in performance.

D. Point Mass System

In Sections IV-C and V-B, we first presented the cost function to minimize the difference between two motors and then optimal control for an oriented point mass formulated as a linear system in the bivector space. Here, we present an optimal trajectory for such a system using several target motors. This example of a control problem using several target motors along the trajectory is shown in Fig. 5.

Note that the same objective can be formulated for manipulators as well, which would correspond to reaching target poses with the end-effector, which makes it similar to classical methods, albeit with a different mathematical formulation. Therefore, we omit showing it for manipulators for brevity and concentrate on modeling and reaching tasks using the geometric primitives in the following sections.

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

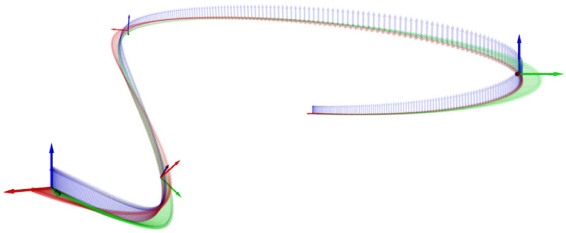


Fig. 5. Optimal trajectory for an oriented point mass system. We placed four target motors along the trajectory at $T/4$, $T/2$, $3T/4$, and T, respectively. The target motors are highlighted along the trajectory. The optimal trajectory was then found using the system defined in (51) and the cost function from (32).

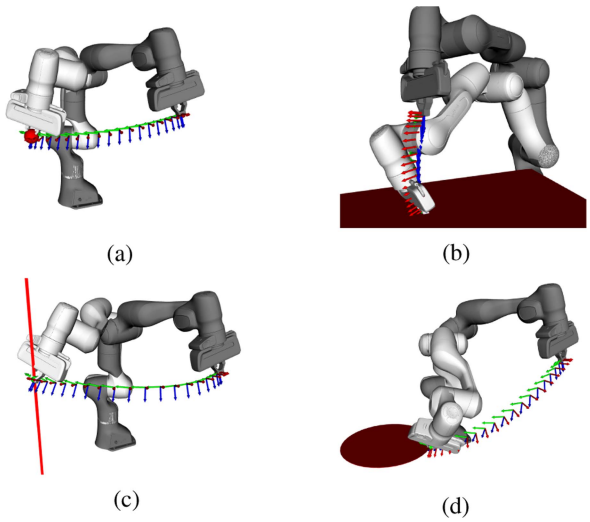


Fig. 6. Examples of optimal trajectories for reaching tasks using different geometric primitives. The initial configuration is always shown in gray and the final one in white. The target geometric primitive is shown in red. And the trajectory is depicted as the frames corresponding to the end-effector. (a) Reaching a point. (b) Reaching a plane. (c) Reaching a line. (d) Reaching a circle.

E. Reaching Tasks

Using the cost function formulation of GA that was presented in (54), 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; however, this remains the subject of further investigations. We present optimal trajectories that were computed using the iLQR to explain how different geometric primitives can be reached using the same structure of the cost function, which is shown in Fig. 6. In the experiments using the real Franka Emika, we then use nominal MPC, which results in an offset for the steady state. This effect is expected but negligible in our work, since the focus of this article is on the modeling aspects. Since we use an MPC framework, we do not need to use fixed points for the reaching but can have movable targets. In practice, we use ArUco markers to track the target online or we disturb the robot while it moves.

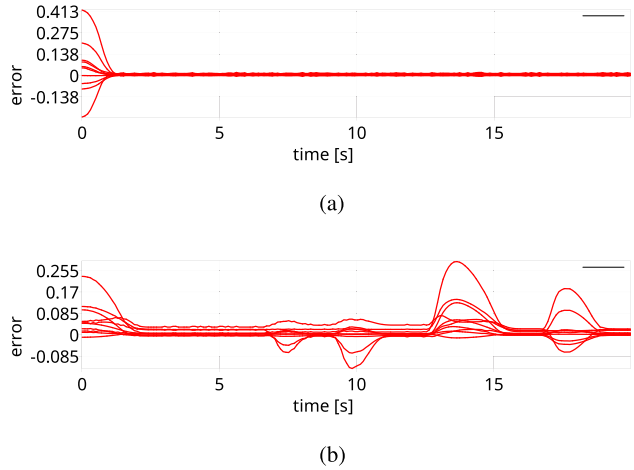


Fig. 7. Error of reaching a fixed target point in MPC for a total duration of 20 s. The individual lines show the elements of resulting 10-D error vector that results from the outer product of two points. (a) Regulation of the end-effector to a target point using MPC without disturbing it. It can be seen that the offset that is induced by the nominal MPC is only very small. (b) Regulation of the end-effector to a target point using MPC while disturbing it.

1) Reaching a Point: Reaching a point means that the desired geometric primitive is a point, i.e., $X_d = P$ in (54). The reference primitive X_r is a point as well and represents the tip of the end-effector. Fig. 6(a) shows an example trajectory for reaching a fixed point from a random initial configuration.

In the real robot experiments, we used a single ArUco marker for reaching a movable point. For safety reasons, the target point was set 10 cm above the marker. We then moved the marker around allowing the robot to follow the reference. We present the results of the real experiment in Fig. 7. In both plots that are presented, we show the values of the 10-D error vector that results from the outer product of two points. If the magnitude of this vector is zero, it means that the reference point is in the null-space of the desired point w.r.t to the outer product. In the case of points, this means that they are identical and the target is reached. Fig. 7(a) presents the static case where we neither moved the point nor disturbed the robot while it moved. We did both of these in the plot shown in Fig. 7(b). It can be seen that the MPC controller is fast and reactive, reaching the targets in a stable manner.

Reaching a point is in this context a trivial example, since it can be easily done using classical methods as well, but it serves to show that reaching problems can be solved for all geometric primitives in the same way as they are solved for a point using GA.

2) Reaching a Point Pair: Using a point pair as the target presents a special opportunity to model a control problem with options. A point pair is the result of the outer product of two points. From this outer product null-space representation, it follows that the outer product of the point pair 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 point pair.

The two possibilities are shown in Fig. 8, where Fig. 8(a) shows the robot reaching the first point and Fig. 8(b) the

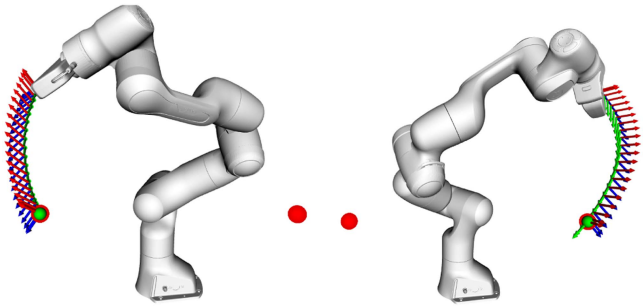


Fig. 8. Depending on the initial configuration, either the left or the right point of the point pair is reached. (a) Left. (b) Right.

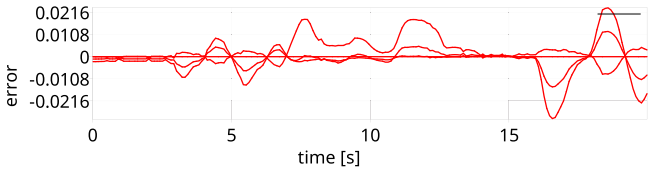


Fig. 9. Components of the error vector resulting from the outer product of a line and a point. Error of reaching a fixed target point in MPC for a total duration of 20 s. The individual lines show the resulting error vector that results from the outer product of a line and a point.

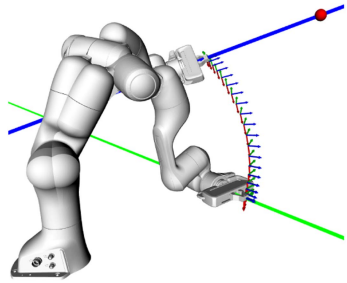


Fig. 10. Optimal trajectory example for a pointing task. The target is shown as the red point. The pointing line is defined to be collinear to the z-axis of the end-effector frame. It is shown in green for the initial configuration and in blue for the final configuration.

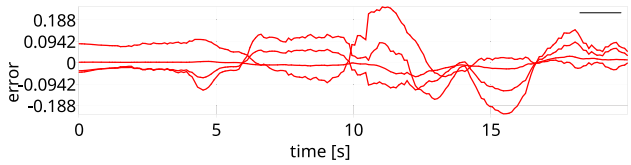


Fig. 11. Experimental results of the pointing task showing the components of the error vector during 20 s. The ArUco marker representing the target point was constantly moved around, which is the reason why the error vector is more jittery than in other experiments.

second one. The point that is reached depends on the initial configuration, and there are no conditional statements required. The corresponding Jacobian is, thus, always computed in the same way, i.e., as presented in (55), and is valid without exceptions. The same is true for the other geometric primitives that we are considering here, but we wanted to specifically highlight the point pair primitive due to its power to model a binary target. We also want to point out here that the point pair primitive would be very suitable for modeling dual-arm manipulation tasks. In this scenario, the point pair would represent the end-effector positions of the two manipulators.

3) Reaching a Plane: A plane in GA is represented by three individual points and $\boldsymbol{e}_\infty$ using the outer product null-space, i.e., $X_d = E = P_1 \wedge P_2 \wedge P_3 \wedge \boldsymbol{e}_\infty$. Note that we do not need to know the orientation, i.e., its normal vector, of the plane in order to define it. It is sufficient to know three points that lie in the plane. When multiplying the plane with a point using the outer product, any point that lies in the plane will result in zero, $E \wedge P = 0$ if $P \in E$. Equation (54) will, therefore, minimize the reaching motion to the plane from any random initial configuration, as shown in Fig. 6(b).

4) Reaching a Line: A line is similar to a plane, but requires only two known points along the line in order to construct it. We show an optimal trajectory for reaching a line in Fig. 6(c). For the real robot experiment, we again used an ArUco marker; in this case, one construction point was on the marker and the other one was 10 cm above it. The target line is, therefore, always perpendicular to the yz plane. In Fig. 9, we show the results of the experiment. The corresponding error vector has six components and is geometrically equivalent to a circle.

5) Reaching a Circle: A target circle is constructed by the outer product of three points, i.e., $X_d = C = P_1 \wedge P_2 \wedge P_3$. These three points uniquely define the circle, and no further knowledge about its radius or orientation is required [but both of these can, of course, be obtained from the circle for the visualization shown in Fig. 6(d)]. The target is here only the boundary of circle (not the full disc).

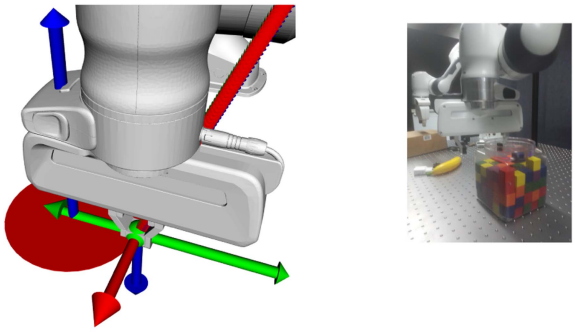
F. Pointing Task

The modeling of a pointing task only requires the usage of a line instead of a point for X in (54). A possible scenario where this task would be applied is tracking an object with a robot arm endowed with a camera. In this case, the line can 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. The setup is shown in Fig. 10. Fig. 11 shows the moving target.

G. Circular Object Grasping Task

In this task, the goal is to give an object with a round opening to the robot. The setup is depicted in Fig. 12. The opening is modeled as a circle, i.e., the robot can grasp the object all around its opening. However, two additional constraints on the orientation are necessary to model this task. These constraints are shown in Fig. 12(a). The end-effector is required to be perpendicular to the plane that the circle lies in. A plane in GA can be obtained from a circle by a multiplying the circle with $\boldsymbol{e}_\infty$, and the normal vector is then obtained as

$$\boldsymbol{n}_E = E^* - 0.5(E^* \cdot \boldsymbol{e}_0)\boldsymbol{e}_\infty. \tag{58}$$



(a) (b)

Fig. 12. Experiment setup of the circular object grasping task. (a) Constraints defining the circular object grasping task: 1) the green point representing the end-effector position needs to lie on the circle (i.e., the boundary of the red disc); 2) the green arrows representing the y-axis of the end-effector frame and the radial vector of the circle must be collinear; and 3) the blue arrows representing the z-axis of the end-effector frame and the normal vector of the circle must be collinear and pointing in opposite directions. (b) Franka Emika robot grasping a box with a circular opening. An ArUco marker is attached to the box to mark its location. The three points defining the circular opening are measured with respect to the marker frame.

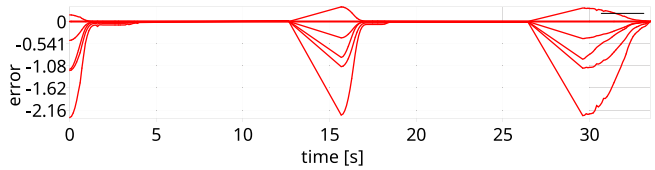


Fig. 13. Experimental results of the circular object grasping task. In this figure, three repetitions are depicted; the location of the target box was changed in-between.

The second constraint is that the direction that the gripper is actuated in needs to be perpendicular to the circle, which mathematically means that this direction needs to be coaxial with the line that connects the grasping position and the center of the circle when it is projected into the plane of the circle. The projection of a point P to the plane E is computed as

$$P' = (E \cdot P)E^{-1}. \tag{59}$$

In Fig. 12, we show the setup of the experiment with the Franka Emika robot. The box that we used has a circular opening, and we defined it by measuring three points relative to an ArUco marker that we attached to the side of the box. Then, during the experiment, the robot was reaching for the box while satisfying the aforementioned constraints using MPC. As soon as it reached (i.e., the cost was below a threshold), the robot closed its gripper in order to hold the box. We repeated this experiment multiple times and changed the box position by holding in our hands for giving it to the robot. An excerpt of the resulting error vector over time can be found in Fig. 13.

VII. Conclusion

In this article, we presented the usage of GA for the modeling of optimal control tasks and how to use the dynamics of serial manipulators computed with GA for inverse dynamics control.

The provided library, *gafro*, is currently specialized for CGA. The implementation of the multivectors and expressions that define the algebra is generic. Thus, it would be possible to use GAs 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 $\mathbb{G}_{6,3}$ [45] or ellipsoids and hyperboloids in $\mathbb{G}_{9,6}$ [46], 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 article, since the properties of the different GAs, such as the outer product null-space, 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.

A possible extension and application of this article would be grasping and in-hand manipulation. Using the geometric representations and the corresponding optimization functions presented in this article, we can actually very easily derive a model for grasping in GA. The three contact types, i.e., point, line, and plane, can directly be represented as geometric primitives. In most cases, the contact points are surface points of objects. Especially, the most commonly used point-on-plane model is always stable.

Appendix

A. Structure of a Multivector in Conformal Geometric Algebra

CGA has 32 basis blades, following the rule $k = 2^{r+p+q}$. Grade 5 is the highest grade of CGA, consequently making $\boldsymbol{e}_{0123\infty}$ the pseudoscalar of the algebra. In practice, the multivectors are sparse, and usually, at most ten elements are nonzero. We call the vector containing the known nonzero blades of a multivector its parameter vector. Note that the grade 0 element is a scalar, which means that vectors and matrices known from linear algebra are actually part of this algebra and can, therefore, be seamlessly multiplied with multivector matrices.

TABLE I
Basis Blades of CGA

grade 0	1
grade 1	$\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3, \boldsymbol{e}_\infty, \boldsymbol{e}_0$
grade 2	$\boldsymbol{e}_{23}, \boldsymbol{e}_{13}, \boldsymbol{e}_{12}, \boldsymbol{e}_{1\infty}, \boldsymbol{e}_{2\infty}, \boldsymbol{e}_{3\infty}, \boldsymbol{e}_{01}, \boldsymbol{e}_{02}, \boldsymbol{e}_{03}, \boldsymbol{e}_{0\infty}$
grade 3	$\boldsymbol{e}_{123}, \boldsymbol{e}_{12\infty}, \boldsymbol{e}_{13\infty}, \boldsymbol{e}_{23\infty}, \boldsymbol{e}_{012}, \boldsymbol{e}_{013}, \boldsymbol{e}_{023}, \boldsymbol{e}_{01\infty}, \boldsymbol{e}_{02\infty}, \boldsymbol{e}_{03\infty}$
grade 4	$\boldsymbol{e}_{123\infty}, \boldsymbol{e}_{0123}, \boldsymbol{e}_{012\infty}, \boldsymbol{e}_{023\infty}, \boldsymbol{e}_{013\infty}$
grade 5	$\boldsymbol{e}_{0123\infty}$

B. Geometric Primitives in Conformal Geometric Algebra

Table II shows the geometric primitives that are available in CGA. We define the outer product null-space as the primal representation, because of its more convenient construction of the primitives. Note that all primitives can be used as the argument of the sandwich product $MX\widetilde{M}$, i.e., applying rigid body transformations to the primitives. Higher dimensional algebras,

TABLE II
GEOMETRIC PRIMITIVES IN CGA [48]

	OPNS (primal)	**IPNS** (dual)
Point P	$\mathcal{C}(\boldsymbol{x})$	$\mathcal{C}(\boldsymbol{x})$
Point Pair A	$P_1 \wedge P_2$	$S_1 \wedge S_2 \wedge S_3$
Sphere S	$P_1 \wedge P_2 \wedge P_3 \wedge P_4$	$P - \frac{1}{2}\rho^2 \boldsymbol{e}_\infty$
Plane E	$P_1 \wedge P_2 \wedge P_3 \wedge \boldsymbol{e}_\infty$	$\boldsymbol{x} + \frac{1}{2}\boldsymbol{x}^2\boldsymbol{e}_\infty - \frac{1}{2}\rho^2\boldsymbol{e}_\infty$
Line L	$P_1 \wedge P_2 \wedge \boldsymbol{e}_\infty$	$E_1 \wedge E_2$
Circle C	$P_1 \wedge P_2 \wedge P_3$	$S_1 \wedge S_2$

such as $\mathbb{G}_{9,6}$, extend these primitives by quadric surfaces such as ellipsoids and hyperboloids.

There are several geometric operations that allow for reasoning about the relationships between primitives such as projections and intersections. The latter can actually be found as well in the construction of the geometric primitives; for example, a circle in its IPNS representation is constructed as the outer product of two spheres, i.e., the intersection of two spheres.

C. Embedding

This section explains the embedding function $\mathcal{E}[\boldsymbol{\mathcal{X}}]$ that is used to obtain the nontrivial parameter matrix of a multivector matrix. Suppose that you have an arbitrary multivector matrix $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I\times J}$ with elements X_{ij} that have K known nontrivial blades $\boldsymbol{e}_k$. The resulting parameter matrix is then of size $IK \times J$, i.e., the parameter vectors of each multivector element get expanded along the columns of the matrix. The embedding function $\mathcal{E}$ is, therefore, defined as

$$\boldsymbol{X} = \mathcal{E}\left[\boldsymbol{\mathcal{X}}\right] = \begin{bmatrix} X_{11,1} & \cdots & X_{1\,J,1} \\ \vdots & \ddots & \vdots \\ X_{11,K} & \cdots & X_{1\,J,1} \\ \vdots & \ddots & \vdots \\ X_{I1,K} & \cdots & X_{IJ,1} \\ \vdots & \ddots & \vdots \\ X_{I1,K} & \cdots & X_{IJ,K} \end{bmatrix}. \tag{60}$$

Note that the embedding function can be considered an implementation detail that ensures a minimal memory consumption and easy integration with off-the-shelf optimization solvers using matrix representations. Therefore, it is handled automatically by the library without the user having to set it explicitly.

D. Derivation of the Jacobian of the Motor Logarithmic Map

From (14), we know that $B = \log(M)$. We define the parameters of the motor and bivector as follows:

$$M = m_1 + m_2\boldsymbol{e}_{23} + m_3\boldsymbol{e}_{13} + m_4\boldsymbol{e}_{12} + m_5\boldsymbol{e}_{1\infty} + m_6\boldsymbol{e}_{2\infty} + m_7\boldsymbol{e}_{3\infty} + m_8\boldsymbol{e}_{123\infty} \tag{61}$$

and

$$B = b_1\boldsymbol{e}_{23} + b_2\boldsymbol{e}_{13} + b_3\boldsymbol{e}_{12} + b_4\boldsymbol{e}_{1\infty} + b_5\boldsymbol{e}_{2\infty} + b_6\boldsymbol{e}_{3\infty}. \tag{62}$$

Standard results show that the motor M can be split into a rotor R and a translator T, such that $M = TR$

$$R = -\boldsymbol{e}_0 \cdot M\boldsymbol{e}_\infty \tag{63}$$

$$T = M\widetilde{R}. \tag{64}$$

Using (9) and (10), it becomes straightforward to derive the bivector components b_i in function of the motor components m_i

$$b_1 = -m_2\frac{2\cos^{-1}(m_1)}{\sin(\cos^{-1}(m_1))} \tag{65}$$

$$b_2 = -m_3\frac{2\cos^{-1}(m_1)}{\sin(\cos^{-1}(m_1))} \tag{66}$$

$$b_3 = -m_4\frac{2\cos^{-1}(m_1)}{\sin(\cos^{-1}(m_1))} \tag{67}$$

$$b_4 = -2(m_1m_5 + m_4m_6 + m_3m_7 + m_2m_8) \tag{68}$$

$$b_5 = -2(-m_4m_5 + m_1m_6 + m_2m_7 - m_3m_8) \tag{69}$$

$$b_6 = -2(-m_3m_5 - m_2m_6 + m_1m_7 + m_4m_8). \tag{70}$$

The Jacobian of the motor logarithmic map can be found as the partial derivatives of the bivector components b_i w.r.t the motor components m_i, i.e.,

$$\boldsymbol{J}_{\mathcal{M}\to\mathbb{B}}(M) = \begin{bmatrix} \frac{\partial b_1}{\partial m_1} & \cdots & \frac{\partial b_1}{\partial m_8} \\ \vdots & \ddots & \vdots \\ \frac{\partial b_6}{\partial m_1} & \cdots & \frac{\partial b_6}{\partial m_8} \end{bmatrix}. \tag{71}$$

Using (65), the nontrivial partial derivatives can be found as follows:

$$\frac{\partial b_1}{\partial m_1} = -2m_2\left(\frac{1}{m_1^2-1} + m_1\cos^{-1}(m_1)\right) \tag{72}$$

$$\frac{\partial b_2}{\partial m_1} = -2m_3\left(\frac{1}{m_1^2-1} + m_1\cos^{-1}(m_1)\right) \tag{73}$$

$$\frac{\partial b_3}{\partial m_1} = -2m_4\left(\frac{1}{m_1^2-1} + m_1\cos^{-1}(m_1)\right) \tag{74}$$

$$\frac{\partial b_1}{\partial m_2} = \frac{\partial b_2}{\partial m_3} = \frac{\partial b_3}{\partial m_4} = \frac{-2\cos^{-1}(m_1)}{\sin(\cos^{-1}(m_1))} \tag{75}$$

$$\frac{\partial b_4}{\partial m_1} = -\frac{\partial b_5}{\partial m_4} = -\frac{\partial b_6}{\partial m_3} = -2m_5 \tag{76}$$

$$\frac{\partial b_4}{\partial m_2} = -\frac{\partial b_5}{\partial m_3} = \frac{\partial b_6}{\partial m_4} = -2m_8 \tag{77}$$

$$\frac{\partial b_4}{\partial m_3} = \frac{\partial b_5}{\partial m_2} = \frac{\partial b_6}{\partial m_1} = -2m_7 \tag{78}$$

$$\frac{\partial b_4}{\partial m_4} = \frac{\partial b_5}{\partial m_1} = -\frac{\partial b_6}{\partial m_2} = -2m_6 \tag{79}$$

$$\frac{\partial b_4}{\partial m_5} = \frac{\partial b_5}{\partial m_6} = \frac{\partial b_6}{\partial m_7} = -2m_1 \tag{80}$$

$$\frac{\partial b_4}{\partial m_6} = -\frac{\partial b_5}{\partial m_5} = \frac{\partial b_6}{\partial m_8} = -2m_4 \tag{81}$$

$$\frac{\partial b_4}{\partial m_7} = -\frac{\partial b_5}{\partial m_8} = -\frac{\partial b_6}{\partial m_5} = -2m_3 \tag{82}$$

$$\frac{\partial b_4}{\partial m_8} = \frac{\partial b_5}{\partial m_7} = -\frac{\partial b_6}{\partial m_6} = -2m_2 \tag{83}$$

which concludes the derivation.

REFERENCES

[1] M. Kamarianakis and G. Papagiannakis, "An all-in-one geometric algorithm for cutting, tearing, and drilling deformable models," *Adv. Appl. Clifford Algebras*, vol. 31, no. 3, 2021, Art. no. 58.

[2] S. Breuils, K. Tachibana, and E. Hitzer, "New applications of Clifford's geometric algebra," *Adv. Appl. Clifford Algebras*, vol. 32, no. 2, 2022, Art. no. 17.

[3] P. Joot, *Geometric Algebra for Electrical Engineers*. Scotts Valley, CA, USA: CreateSpace, 2019.

[4] D. Hestenes, G. Sobczyk, and J. S. Marsh, "Clifford algebra to geometric calculus. A unified language for mathematics and physics," *Amer. J. Phys.*, vol. 53, no. 5, pp. 510–511, 1985.

[5] E. Bayro-Corrochano, "A survey on quaternion algebra and geometric algebra applications in engineering and computer science 1995–2020," *IEEE Access*, vol. 9, pp. 104326–104355, 2021.

[6] L. Kavan, S. Collins, C. O'Sullivan, and J. Zara, "Dual quaternions for rigid transformation blending," Trinity College Dublin, Dublin, Ireland, Tech. Rep. TCD-CS-2006-46, 2006.

[7] G. Leclercq, P. Lefèvre, and G. Blohm, "3D kinematics using dual quaternions: Theory and applications in neuroscience," *Front. Behav. Neurosci.*, vol. 7, 2013, Art. no. 7.

[8] F. F. A. Silva, J. J. Quiroz-Omaña, and B. V. Adorno, "Dynamics of mobile manipulators using dual quaternion algebra," *J. Mech. Robot.*, vol. 14, no. 6, 2022, Art. no. 061005.

[9] M. d. P. A. Fonseca, B. V. Adorno, and P. Fraisse, "Coupled task-space admittance controller using dual quaternion logarithmic mapping," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 6057–6064, Oct. 2020.

[10] M. M. Marinho, L. F. C. Figueredo, and B. V. Adorno, "A dual quaternion linear-quadratic optimal controller for trajectory tracking," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 4047–4052.

[11] M. M. Marinho, B. V. Adorno, K. Harada, and M. Mitsuishi, "Dynamic active constraints for surgical robots using vector-field inequalities," *IEEE Trans. Robot.*, vol. 35, no. 5, pp. 1166–1185, Oct. 2019.

[12] S. Calinon, "Gaussians on Riemannian manifolds: Applications for robot learning and adaptive control," *IEEE Robot. Autom. Mag.*, vol. 27, no. 2, pp. 33–45, Jun. 2020.

[13] N. Jaquier, L. Rozo, D. G. Caldwell, and S. Calinon, "Geometry-aware manipulability learning, tracking, and transfer," *Int. J. Robot. Res.*, vol. 40, nos. 2/3, pp. 624–650, 2021.

[14] F. Marić, L. Petrović, M. Guberina, J. Kelly, and I. Petrović, "A Riemannian metric for geometry-aware singularity avoidance by articulated robots," *Robot. Auton. Syst.*, vol. 145, 2021, Art. no. 103865.

[15] F. Marić, M. Giamou, A. W. Hall, S. Khoubyarian, I. Petrovic, and J. Kelly, "Riemannian optimization for distance-geometric inverse kinematics," *IEEE Trans. Robot.*, vol. 38, no. 3, pp. 1703–1722, Jun. 2022.

[16] M. C. L. Belon, "Applications of conformal geometric algebra in mesh deformation," in *Proc. 24th Conf. Graph., Patterns, Images*, 2013, pp. 39–46.

[17] H. Hadfield, S. Achawal, J. Lasenby, A. Lasenby, and B. Young, "Exploring novel surface representations via an experimental ray-tracer in CGA," *Adv. Appl. Clifford Algebras*, vol. 31, no. 2, 2021, Art. no. 16.

[18] W. B. Lopes and C. G. Lopes, "Geometric-algebra adaptive filters," *IEEE Trans. Signal Process.*, vol. 67, no. 14, pp. 3649–3662, Jul. 2019.

[19] Y. He, R. Wang, X. Wang, J. Zhou, and Y. Yan, "Novel adaptive filtering algorithms based on higher-order statistics and geometric algebra," *IEEE Access*, vol. 8, pp. 73767–73779, 2020.

[20] A. Aristidou and J. Lasenby, "Inverse kinematics solutions using conformal geometric algebra," in *Guide to Geometric Algebra in Practice*, L. Dorst and J. Lasenby, Eds. London, U.K.: Springer, 2011, pp. 47–62.

[21] A. Aristidou, Y. Chrysanthou, and J. Lasenby, "Extending FABRIK with model constraints," *Comput. Animation Virtual Worlds*, vol. 27, no. 1, pp. 35–57, 2016.

[22] D. Kolpashchikov, O. Gerget, and V. Danilov, "FABRIKx: Tackling the inverse kinematics problem of continuum robots with variable curvature," *Robotics*, vol. 11, no. 6, 2022, Art. no. 128.

[23] I. Zaplana, H. Hadfield, and J. Lasenby, "Closed-form solutions for the inverse kinematics of serial robots using conformal geometric algebra," *Mech. Mach. Theory*, vol. 173, 2022, Art. no. 104835.

[24] E. Bayro-Corrochano and J. Zamora-Esquivel, "Differential and inverse kinematics of robot devices using conformal geometric algebra," *Robotica*, vol. 25, no. 1, pp. 43–61, 2007.

[25] L. Lechuga-Gutierrez, E. Macias-Garcia, G. Martínez-Terán, J. Zamora-Esquivel, and E. Bayro-Corrochano, "Iterative inverse kinematics for robot manipulators using quaternion algebra and conformal geometric algebra," *Meccanica*, vol. 57, no. 6, pp. 1413–1428, 2022.

[26] H. Hadfield and J. Lasenby, "Constrained dynamics in conformal and projective geometric algebra," in *Advances in Computer Graphics*, N. Magnenat-Thalmann, C. Stephanidis, and E. Wu, Eds., vol. 12221. Cham, Switzerland: Springer, 2020, pp. 459–471.

[27] C. A. Arellano-Muro, G. Osuna-González, B. Castillo-Toledo, and E. Bayro-Corrochano, "Newton-Euler modeling and control of a multi-copter using motor algebra," *Adv. Appl. Clifford Algebras*, vol. 30, no. 2, 2020, Art. no. 19.

[28] E. Bayro-Corrochano, J. Medrano-Hermosillo, G. Osuna-González, and U. Uriostegui-Legorreta, "Newton-Euler modeling and Hamiltonians for robot control in the geometric algebra," *Robotica*, vol. 40, no. 11, pp. 4031–4055, 2022.

[29] E. Bayro-Corrochano, A. M. Garza-Burgos, and J. L. Del-Valle-Padilla, "Geometric intuitive techniques for human machine interaction in medical robotics," *Int. J. Soc. Robot.*, vol. 12, no. 1, pp. 91–112, 2020.

[30] L. González-Jiménez, O. Carbajal-Espinosa, A. Loukianov, and E. Bayro-Corrochano, "Robust pose control of robot manipulators using conformal geometric algebra," *Adv. Appl. Clifford Algebras*, vol. 24, no. 2, pp. 533–552, 2014.

[31] J. Zamora-Esquivel and E. Bayro-Corrochano, "Robot object manipulation using stereoscopic vision and conformal geometric algebra," *Appl. Bionics Biomech.*, vol. 8, nos. 3/4, pp. 411–428, 2011.

[32] 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, Switzerland: Springer, 2021, pp. 111–131.

[33] S. Breuils, V. Nozick, and L. Fuchs, "Garamon: A geometric algebra library generator," *Adv. Appl. Clifford Algebras*, vol. 29, no. 4, 2019, Art. no. 69.

[34] D. Fontijne, "Gaigen 2: A geometric algebra implementation generator," in *Proc. 5th Int. Conf. Generative Program. Compon. Eng.*, 2006, pp. 141–150.

[35] E. V. Sousa and L. A. F. Fernandes, "TbGAL: A tensor-based library for geometric algebra," *Adv. Appl. Clifford Algebras*, vol. 30, no. 2, 2020, Art. no. 27.

[36] J. Ong, "GAL," GitHub, 2019. [Online]. Available: https://github.com/jeremyong/gal

[37] P. Colapinto, "Versor: Spatial computing with conformal geometric algebra," 2011. [Online]. Available: http://versor.mat.ucsb.edu

[38] B. V. Adorno and M. M. Marinho, "DQ robotics: A library for robot modeling and control," *IEEE Robot. Autom. Mag.*, vol. 28, no. 3, pp. 102–116, Sep. 2021.

[39] E. Bayro-Corrochano, *Geometric Algebra Applications: Robot Modelling and Control*, vol. 2. Cham: Switzerland: Springer, 2020.

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

[41] A. Macdonald, "A survey of geometric algebra and geometric calculus," *Adv. Appl. Clifford Algebras*, vol. 27, no. 1, pp. 853–891, 2017.

[42] C. Gunn, "Geometric algebras for Euclidean geometry," *Adv. Appl. Clifford Algebras*, vol. 27, no. 1, pp. 185–208, 2017.

[43] T.-A. Johansen, J. J. C. Sanchez, and R. Kristiansen, "Dual quaternion control: A review of recent results within motion control," *Nonlinear Stud.*, vol. 26, no. 4, 2019, Art. no. 25.

[44] E. Bayro-Corrochano, *Geometric Algebra Applications*, vol. 1. Cham, Switzerland: Springer, 2019.

[45] J. Zamora-Esquivel, "G 6,3 Geometric algebra: Description and implementation," *Adv. Appl. Clifford Algebras*, vol. 24, no. 2, pp. 493–514, 2014.

[46] S. Breuils, V. Nozick, and E. Hitzer, "Quadric conformal geometric algebra of R9,6," *Adv. Appl. Clifford Algebras*, vol. 28, pp. 1–16, 2018.

[47] E. J. Bayro-Corrochano, G. Altamirano-Escobedo, A. Ortiz-Gonzalez, V. Farias-Moreno, and N. Chel-Puc, "Computing in the conformal space objects, incidence relations, and geometric constrains for applications in AI, GIS, graphics, robotics, and human-machine interaction," *IEEE Access*, vol. 10, pp. 112742–112756, 2022.

[48] C. Perwass, *Geometric Algebra With Applications in Engineering* (ser. Geometry and Computing). Berlin, Germany: Springer, 2009.
[49] E. Hitzer, C. Lavor, and D. Hildenbrand, "Current survey of Clifford geometric algebra applications," *Math. Methods Appl. Sci.*, 2022, Art. no. mma.8316.
[50] B. V. Adorno, "Two-arm manipulation: From manipulators to enhanced human-robot collaboration," Ph.D. dissertation, Dept. Robot., Univ. Montpellier II—Sciences et Techniques du Languedoc, Montpellier, France, 2011.
[51] L. Tingelstad, "Automatic differentiation and optimization of multivectors: Estimating motors in conformal geometric algebra," Ph.D. dissertation, Dept. Mech. Ind. Eng., Norwegian Univ. Sci. Technol., Trondheim, Norway, 2017.
[52] L. Tingelstad and O. Egeland, "Motor parameterization," *Adv. Appl. Clifford Algebras*, vol. 28, no. 2, 2018, Art. no. 34.
[53] E. Bayro-Corrochano and D. Kähler, "Motor algebra approach for computing the kinematics of robot manipulators," *J. Robot. Syst.*, vol. 17, no. 9, pp. 495–516, 2000.
[54] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics (ser. Advanced Textbooks in Control and Signal Processing)*, M. J. Grimble and M. A. Johnson, Eds. London, U.K.: Springer, 2009.
[55] C. Doran and A. Lasenby, *Geometric Algebra for Physicists*, 1st ed. Cambridge, U.K.: Cambridge Univ. Press, 2003.
[56] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 4906–4913.
[57] 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.
[58] J. Carpentier et al., "The pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *Proc. IEEE/SICE Int. Symp. Syst. Integr.*, 2019, pp. 614–619.
[59] R. Smits, "KDL: Kinematics and dynamics library," Accessed: Oct. 25, 2022. [Online]. Available: http://www.orocos.org/kdl
[60] N. T. Dantam, "Robust and efficient forward, differential, and inverse kinematics using dual quaternions," *Int. J. Robot. Res.*, vol. 40, nos. 10/11, pp. 1087–1105, 2021.
[61] P. Beeson and B. Ames, "TRAC-IK: An open-source library for improved solving of generic inverse kinematics," in *Proc. IEEE-RAS 15th Int. Conf. Humanoid Robots*, 2015, pp. 928–935.

Tobias Löw (Student Member, IEEE) received the B.Sc. and M.Sc. degrees in mechanical engineering from ETH Zürich, Zürich, Switzerland, in 2018 and 2020, respectively. He is currently working toward the Ph.D. degree in electrical engineering with the École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.

He is also a Research Assistant with the Robot Learning and Interaction Group, Idiap Research Institute, Martigny, Switzerland. He conducted his master's thesis with the Robotics and Autonomous Systems Group, CSIRO, Brisbane City, QLD, Australia. His research interests include exploiting geometric and structural methods for optimization problems in robotics.

Sylvain Calinon (Member, IEEE) received the Ph.D. degree in robotics from the École Polytechnique Fedérale de Lausanne (EPFL), Lausanne, Switzerland, in 2007.

He is currently a Senior Researcher with Idiap Research Institute, Martigny, Switzerland, and a Lecturer with EPFL. From 2009 to 2014, he was a Team Leader with the Italian Institute of Technology, Genoa, Italy. From 2007 to 2009, he was a Postdoctoral Researcher with EPFL. His research interests include human–robot collaboration, robot learning, and model-based optimization.