Robot Manipulation with Geometric Algebra: A Unified Geometric Framework for Control and Optimization

Tobias Löw

July 4, 2025

Acknowledgements

Life is a journey and an adventure that you cannot go through alone. This PhD was part of mine, and there have been many people along the way supporting me. To all of them I want to express my deepest gratitude, without them this would not have been possible.

First, I would like to thank my advisors Prof. Jean-Philippe Tiran and Dr. Sylvain Calinon for giving me the opportunity to pursue a doctorate jointly with the Idiap Research Institute and EPFL. I am very grateful to Sylvain for his availability, many discussions, and his support of my chosen research direction. I would also like to thank the RLI group members, Amir, Teng, Yiming, Yan, Ante, James, Martin, Hakan, Suhan, Teguh, Riccardo, João, Mattia, Jérémy, Guillaume, for many valuable discussions at the coffee machine. A special thanks goes to Cem for many great discussions and collaborations that contributed directly to this thesis.

During my masters, I had the opportunity to work at CSIRO in Brisbane, Australia, and to get a first glance at the world of research. I would like to thank Paulo Borges and Tirthankar Bandyophadhay for giving me this incredible opportunity and to provide me with their support.

While doing research is a very fulfilling challenge, this thesis would not have been possible without the many people helping me clear my head during countless amazing adventures. Therefore, I would like to thank: Sevi for starting this journey together with long study hours during our bachelors and many camping adventures; Juliette, Damien, Laure, Soizic, Lucas and Elea for many days and nights spent laughing during our adventures; Sha, Nathaly and Jonathan for being incredible housemates; Kyra for a friendship across continents; Florian for many hours of riding our bikes together; Vera for being a reliable hiking partner; Roli for many years of friendship and celebrating at concerts; Sophia for an amazing friendship that started at the other side of the world; Sophie for guiding me through many yoga sessions; Michel for taking care of our bees together.

Lastly, I would like to thank my family, my parents and my sister, Hannah, for always supporting me and believing in me. Without them, I would not be where I am right now.

Sion, July 4, 2025
Tobias Löw

Zusammenfassung

Die fortschreitende Entwicklung von Robotersystemen von strukturierten industriellen Umgebungen hin zu unstrukturierten, menschenzentrierten Umgebungen erfordert neue Ansätze, um die Anpassungsfähigkeit, Sicherheit und Robustheit zu gewährleisten. In dieser Arbeit wird die These vertreten, dass geometrisch kohärente Darstellungen für die Bewältigung dieser Herausforderungen von entscheidender Bedeutung sind. Daher wird in dieser Arbeit ein einheitlicher geometrischer Rahmen für die Manipulation von Robotern unter Verwendung der konformen geometrischen Algebra (CGA) vorgestellt. Die CGA ist eine mathematische Sprache, die geometrische Grundelemente (Punkte, Linien, Ebenen, Kugeln) und Transformationen von starren Körpern (Drehungen, Verschiebungen, Skalierungen) in einer einzigen algebraischen Struktur vereint. Durch die Verwendung der CGA und ihrer Fähigkeit, geometrische Intuition direkt in algebraische Operationen zu übersetzen, werden grundlegende Herausforderungen der Robotik wie Kinematik, Dynamik, optimale Steuerung und kooperative Manipulation bewältigt. Die Ergebnisse ermöglichen eine intuitive Modellierung von Manipulationsaufgaben durch geometrische Grundelemente und deren Ähnlichkeitstransformationen, die Skalierung, Rotation und Verschiebung vereinen. Die vorgestellten Modellierungsansätze ermöglichen eine Generalisierung von einarmigen auf zweiarmige und mehrarmige Systeme. Wir zeigen, wie dieser Ansatz in optimalen Regelungsformulierungen sowie in Regelungsanwendungen, bei denen die Aufgabenziele einheitlich über alle Grundoperationen hinweg formuliert werden und keine anwendungsspezifischen Anpassungen erforderlich sind, eingesetzt werden kann und so eine kompatible Interaktion ermöglicht wird. Die praktischen Beiträge umfassen die *gafro*-Bibliothek, eine Open-Source-Implementierung der CGA, die speziell für die Robotik entwickelt wurde. Die Beiträge ermöglichen nicht nur theoretische Fortschritte in der Robotik, sondern liefern auch praktische Werkzeuge für die Anwendung in der Realität. Wir zeigen dies durch die Lösung von Aufgaben im Bereich der zweiarmigen Admittanzsteuerung und der taktilen ergodischen Oberflächenabdeckung.

Schlüsselbegriffe Geometrische Algebra, Optimale Regelung, Kooperative Manipulation, Kraftregelung, Zweiarmige Manipulation

Abstract

The ongoing transition of robotic systems from structured industrial settings to unstructured human-centric environments necessitates novel approaches to ensure adaptability, safety, and robustness. This thesis supports the view that geometrically coherent representations are pivotal in addressing these challenges. Therefore, this work presents a unified geometric framework for robot manipulation using Conformal Geometric Algebra (CGA), a mathematical language that seamlessly integrates geometric primitives (points, lines, planes, spheres) and rigid-body transformations (rotations, translations, scaling) into a single algebraic structure. By leveraging CGA's capacity to encode geometric intuition directly into algebraic operations, we address key challenges in robotics, including kinematics, dynamics, optimal control, and cooperative manipulation. Our results enable intuitive modeling of manipulation tasks through geometric primitives and their similarity transformations, which unify scaling, rotation, and translation. The presented modeling approaches generalize from single-, to dual-, and multi-arm systems. We show how this approach can be used in optimal control formulations as well as force control applications, where task objectives are formulated uniformly across primitives, avoiding case-specific adjustments and enabling compliant interactions. Practical contributions include the *gafro* library, an open-source CGA implementation tailored for robotics. The contributions not only provide theoretical advancements in geometric robotics but also deliver practical tools for real-world deployment. We demonstrate this by solving tasks around dual-arm admittance control and tactile ergodic surface coverage.

Keywords Geometric Algebra, Optimal Control, Cooperative Manipulation, Force Control, Dual-Arm Manipulation

List of Figures

List of Tables

Contents

Notation

We use the following notation throughout the thesis: x to denote scalars, $\boldsymbol{x}$ for vectors, $\boldsymbol{X}$ for matrices, X for multivectors and $\boldsymbol{\mathcal{X}}$ for matrices of multivectors. The operator $\mathcal{E}(\cdot)$ extracts the minimal parameter vector of a multivector, and $\mathcal{E}^{-1}(\cdot)$ is its inverse.

1. Introduction

1.1. Motivation and Challenges

Currently, in a profound technological shift, robots are transitioning from structured industrial environments to unstructured human-centric spaces. These scenarios range from cluttered homes and busy urban streets to intricate surgical theaters and disaster zones. While advances in computational power, structural design, sensors, and algorithm development certainly drive this evolution forward, the push towards unstructured environments comes with tremendous challenges. The fundamental question is how should robots model, perceive, and interact with their surroundings in order to achieve the required adaptability and robustness. This thesis supports the view that geometrically coherent representations are essential for this transformational shift. Well-chosen representations that capture the intrinsic geometric structure of robotic tasks subsequently enable efficient computation, safe interaction, and scalable generalization. For this purpose, in this thesis, we use the mathematical theory of geometric algebra that unifies the aforementioned approaches into a single algebraic framework. It contains vectors, quaternions, complex numbers and more generally geometric objects that can all be used for consistent, concise and intuitive geometric reasoning and calculation [102]. We present how to exploit this unifying geometric framework in complex manipulation scenarios through the geometric primitives, their transformations and their variations.

The first robotics revolution saw the introduction of industrial robots for automating repetitive manufacturing tasks like welding, painting and assembly. Their success was founded on the well-defined workspace environments, where geometry, object poses, and task sequences were known ahead of time [194]. The deterministic control algorithms relied on precise kinematic models and were shielded from external perturbations. Trajectories for six-axis robotic arms were preprogrammed and employed in automotive assembly lines. Accurate calibration then ensured repeatability [59]. While these systems are capable of repeating the same tasks without failures, they cannot deal with the variability encountered in unstructured settings, since they lack the necessary flexibility. These limitations become obvious, when robots are expected to operate in human environments. Navigating tight living spaces, manipulating different objects, while adapting to particular contexts are the tasks of assistive service robots [166]. Urban environments where self-driving cars are deployed require decision-making under uncertainty to deal with the chaotic movements of people, other vehicles, and unforeseen obstacles such as road blockages [90]. High precision is demanded from surgical robots when operating on the human body, to avoid damaging the soft tissue that can deform and vary with different anatomies [208]. Search and rescue robots are challenged by traversing and manipulating debris, and detecting survivors, while not being able to rely on prior maps [165]. Even in industrial settings, low-volume and high-mix production, as well as surface operations that require contact force industrial robots to abandon the traditional control paradigms. The common challenges across these scenarios are the safety constraints

to not harm or damage, partial observability and environmental variability leading to uncertainty, and dynamic interactions coming from moving objects and agents. Simple rule-based algorithms are not sufficient to solve these challenges. Instead, they need to exploit the geometric structure for probabilistic reasoning, adaptive learning, and model-based control.

Many problems in robotics are fundamentally problems of geometry. Whether manipulating objects, avoiding obstacles, or interpreting sensor data: spatial relationships dictate system behavior. Consider for example object manipulation, where the relationship between the pose of a gripper and the object is defined by a rigid body transformation in $SE(3)$. Navigation requires collision avoidance, where distances are computed based on the geometries of the robot and the obstacles. Projective geometry is used in perception when inferring three-dimensional structures from images. Hence, each robotic task has an intrinsic geometry. Since the problem and its geometric structure are deeply interconnected, we need to choose representations that intuitively allow incorporating the geometry of the problem. In other words, the properties of a function inform about the domain it is defined on and in return the domain structure enforces other properties ([40]). Geometric priors, such as symmetry and invariance, can be exploited to reduce the complexity and constrain learning spaces. For example, $SE(3)$-equivariant networks improve generalization by ensuring that predictions transform consistently under rigid body transformations [70]. SLAM algorithms can be improved by considering scale-invariant features [175].

Geometric methods in robotics reach back to screw theory formalized by [17]. The concept of screws unifies translational and rotational motion using helical axes. Based on screws, a dual formulation for force and motion was derived: twists describe instantaneous linear and angular velocities, while wrenches represent forces and torques. In practice, screw theory is for example applied to parallel manipulators to compute actuator forces for desired motions [84]. The manifolds encountered in robotics are often non-Euclidean. Here, Riemannian geometry generalizes concepts from flat Euclidean space to curved manifolds. A simple example are configurations spaces with joint limits that form a manifold that requires using geodesics for motion planning [41]. Rigid transformations are very prominent in robotics and can be properly represented via Lie groups such as $SE(3)$. The associated Lie algebra ($\mathfrak{se}(3)$) linearizes group operations, enabling efficient integration of angular velocities. Its applications include consistent sensor fusion state estimation via the invariant extended Kalman filter [35]. By augmenting quaternions with dual numbers, dual quaternions were constructed to provide a compact, concise, and singularity-free, and computational representation of rigid transformations. They enable smooth interpolation that avoid discontinuities [121] and efficient inverse kinematics [62]. In robotics, we often use homogeneous coordinates that embed 3D Euclidean space into a projective space to enable the representation of rigid transformations as 4x4 matrices [95].

1.2. State of the Art

In the previous section, we have motivated the general importance of geometric methods in robotics. In this section, we give a more detailed overview of recent geometric methods in robotics. Since this thesis advocates for the usage of geometric algebra in robotics as a unifying geometric framework, we put special focus on existing work that uses geometric algebra and the closely related dual quaternions.

Geometry plays a crucial role in robot learning by providing a principled approach for understanding and representing the relationships between different data modalities [45, 111] since often there are non-Euclidean spaces involved that require the adaption of traditional approaches to those manifolds [40, 160]. Accordingly, an increased effort has been put into deriving methods that are considering the underlying geometry of the robot's configuration and task spaces [156, 155, 52, 126]. The different approaches include powerful mathematical techniques such as Riemannian manifolds and Lie groups, which have been used especially at the intersection of learning and control, by using manipulability ellipsoids [112], vector fields on Lie groups [211], dynamical systems on Riemannian manifolds [186] or even learning manifolds [26]. Leveraging the intrinsic geometry provides theoretical guarantees and practical performance increases for optimization and control [129, 34, 202, 213].

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 [45] 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 [112]. In [156] 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 [155]. Another aspect when developing robotic algorithms, apart from the modeling of the robot itself, is the modeling of the tasks and the environment. Here, an increased focus has been put on exploiting the underlying geometry of the problems. The different approaches include powerful mathematical techniques such as Riemannian manifolds and Lie groups. They have been used especially at the intersection of learning and control. Various works presented approaches for representing robot skills using Riemannian manifolds and it was shown that different manifolds can be used for that purpose [45]. The manifold can be predefined, such as learning orientations [186], and leveraged in a dictionary of different manifolds [209]. In [112], the authors presented a framework for geometry aware manipulability learning, and then tracking the learned trajectory of manipulability ellipsoids and transferring it to different systems. These works used the manifold of symmetric positive-definite matrices for the skill representation. The underlying Riemannian manifold can, however, also be learned to obtain motion skills

and then later be used for reactive motion generation [26]. Riemannian motion policies are another geometric framework for reactive control [52]. In addition to these geometric learning and control methods, Riemannian optimization is also used to exploit geometric structures for solving the inverse kinematics problem based on distance geometry [155].

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. Geometric algebra 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, geometric algebra can be seen as an extension and generalization of dual quaternions [18], since dual quaternions can be identified with a certain Clifford algebra, which forms the foundation of geometric algebra. 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 [120]. In robotics, there have been various works describing the kinematics and dynamics of robots in dual quaternion algebra [133, 6]. Efficient control is an important aspect for using real robots and dual quaternions have been used to design admittance controllers [80] and LQR controller for trajectory tracking [157]. Collision avoidance is an important aspect of control and in [158] vector field inequalities based on dual quaternions were proposed to handle them during surgical tasks.

Geometric algebra has been applied successfully in a variety of different applications and fields. For example in the field of computer graphics, which started to re-popularize it, it found applications in mesh deformation [28] as well as ray casting and surface representation [93]. In the domain of image processing techniques for adaptive filtering have been devised [143], [98].

A popular example in robotics to show the strengths of geometric algebra is solving the inverse kinematics problem. There have been various methods that proposed to utilize the geometric primitives and their intersection such as FABRIK [13] which finds an iterative solution and has also been extended to include model constraints [12]. Recently another extension, called FABRIKx [128], was 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 [222], while [24] presented the differential and inverse kinematics of robots using conformal geometric algebra and an iterative inverse kinematics solution was derived in [132]. Recent work has approached the topic of formulating constrained dynamics in conformal geometric algebra [94]. Newton-Euler modeling has been proposed for multi-copters using motor algebra in [11] and for robot control using conformal geometric algebra in [23]. 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 [21]. Conformal geometric algebra was presented for robust pose control of manipulators in [89] and [221] used it for robot object manipulation. Mathematically, geometric algebra presents fresh insights on screw theory [65], which is often used in robotics to express kinematic relationships.

1.3. Contributions and Thesis Outline

This thesis is organized into three main parts. The first part introduces the mathematical background on geometric algebra and its application to the general modeling of robotic systems. The second part outlines how geometric algebra can be used to formulate different optimization problems for manipulation tasks. In the third part, we focus on the practical aspects of geometric algebra by including it in real control systems. The contributions of each part are described below in more detail.

Part I: Mathematical Fundamentals Chapter 2 introduces the mathematical background on geometric algebra. It explains the fundamental theory of how this family of algebras is constructed from vector spaces. Furthermore, we present the relations, definitions and equations that are necessary for understanding the rest of this thesis. Afterward, we introduce the specific variant known as conformal geometric algebra (CGA) that is used throughout this thesis. Here, we focus on the geometric primitives that CGA is capable of representing as algebraic objects and on the transformation groups that are contained within CGA. The geometric primitives and the transformations on them are the two fundamental building blocks for ideas that are presented in this thesis.

Chapter 3 then provides the mathematical details on using geometric algebra for the modeling of robot kinematics and dynamics. It gives insights on the forward kinematics and the related Jacobians using the conventional terminology, and shows optimization-based inverse kinematics. We then present the dynamics modeling of serial kinematic chains. We extend the Lagrangian dynamics of serial manipulator in CGA to include a non-trivial inertia tensor and subsequently use the derived dynamics in an inverse dynamics control scheme. Furthermore, we propose a recursive algorithm for the computation of the forward dynamics. We use the results of this chapter to draw parallels to classical control schemes, such as differential kinematics and inverse dynamics control.

Part II: Modeling of Optimization Problems for Manipulation Tasks After introducing the basics of geometric algebra for robotics, this part focuses on modeling optimization problems for manipulation tasks. The part is separated into four chapters. The aim of this part is to utilize CGA to formulate optimal control problems for

manipulation tasks in a geometrically uniform manner. Optimal control deals with the problem of finding a control sequence minimizing an objective function which encodes the requirements of the task while at the same time adhering to the constraints of the robot and the environment.

Chapter 4 leverages the capabilities of CGA to achieve uniform modeling of manipulation tasks across different geometric primitives resulting in a low symbolic complexity of the resulting expressions and a geometric intuitiveness. We show that this formulation can be used to achieve compliant behaviors according to geometric primitives when employed using model-predictive control on a torque-controlled robot.

Chapter 5 extends the optimal control formulation to bimanual systems, which is building on the cooperative dual-task space that was originally defined using dual quaternions [4]. Here, geometric algebra allows for the simultaneous representation of both end-effector positions as a pointpair primitive, which enables automatic decision-making of the system on which arm should reach for a target without conditional statements.

The ideas from the cooperative dual-task space are then further extended in Chapter 6. That chapter introduces the cooperative control for multiple parallel kinematic chains based on cooperative geometric primitives. These primitives are found via the outer product of the end-effector of the kinematic chains. Using similarity transformations, we then define a modeling strategy that closely mimics single manipulators. This way, classical control strategies are readily available for these complex systems with many degrees of freedom. By integrating this approach with optimal control and teleoperation, we then show the simplicity in defining cooperative control objectives.

The previous chapters explored manipulation tasks from the perspective of optimization given ideal conditions. In Chapter 7 we explore a probabilistic perspective on the geometric primitives and their transformations. This way, sensor noise, actuator hysteresis, or other sources of modeling or perception uncertainty can be considered. Furthermore, the probabilistic perspective presents a gateway to including statistical learning of manipulation tasks into the geometric algebra optimization framework. Therefore, this chapter derives precision weighted cost functions of the objective functions from the previous chapters by considering the covariance propagation through the products of geometric algebra.

Part III: Practical Applications Using the theoretical insights from the previous parts, this part aims at applying the geometric insights in practice. Apart from theoretical contributions, this thesis contributes an open-source library called ***g**eometric **a**lgebra **f**or **ro**botics (gafro)* that implements all the presented formulations and algorithms. The library is based on a fast and efficient custom implementation of CGA using expression templates. In contrast to existing GA libraries, this 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 explain the details of this library in Chapter 8 and provide examples in the form of a short tutorial on how to use this library in practice.

Chapter 9 then uses the cooperative dual-task from the previous part to formulate a dual-arm admittance control scheme in CGA. This controller is deployed in a real-world industrial application for lifting big and bulky objects. The particular use-case comes from our industrial partner in Horizon Europe project SESTOSENSO (`https://sestosenso.eu`) that this thesis is a part of.

As continuous physical interaction between robots and their environment is a requirement in many industrial and household tasks, Chapter 10 proposes a closed-loop control method that is constrained to surfaces. Due to the complex tactile information, these tasks are notoriously difficult to model and to sense. Thus, planning open-loop trajectories is extremely challenging and likely to fail. One of the challenges was to keep contact with the surface without applying excessive force. To solve this issue, we use geometric algebra for an intuitive way of representing a line corresponding to the surface orientation at a given position. This formulation will cause the system to track the line while being free to move along it, which lets it stay in contact with the target surface by simultaneously exerting a desired force along that line. Accordingly, the force and the line controller can simultaneously be active without conflicting objectives or rigorous parameter tuning.

Chapter 11 concludes this thesis by discussing the presented results and providing ideas for future work.

Part I.

Mathematical Fundamentals

2. Geometric Algebra

2.1. Introduction

Geometric Algebra (GA) can be seen as a *high-level mathematical language* for geometry that unifies several known concepts, which makes it a very effective tool when the physics of a system need to be modeled. The roots of geometric algebra can be found in Clifford algebra, which was a unification of quaternions and Grassmann algebra [38]. The result was the geometric product, which is the sum of an inner and an outer product. This unfamiliar concept actually leads to algebraic tools that allow for the simplification of many otherwise complex equations, making them more intuitive to handle. A well-known example for this simplification are the Maxwell equations, which reduce to only a single equation in geometric algebra $\left(\boldsymbol{\nabla} + \frac{1}{c}\frac{\partial}{\partial t}\right) F = J$ [114].

The representational advantage of geometric algebra is the geometric significance of its elements, meaning that an object can directly represent geometric primitives, such as lines, spheres and planes, as well as orthogonal transformations, such as rotations, translations, scaling and projections. This allows the direct extraction of geometric information about the problem from the equations. Furthermore, its elements, called multivectors, avoid the parameter redundancy of other representations such as matrices, leading to less memory consumption and optimized computation compared to analytic geometry or vector calculus, which makes it an amenable framework for real-time applications. In engineering the validity of equations is usually determined by a dimensional check of the quantities of the formula. These quantities are of a certain algebraic order when using geometric algebra, which adds a structural check for the validity. These properties were some fundamental criteria in the design of geometric algebra, along with the possibility to formulate basic equations in a coordinate-free manner and to smoothly transfer information between formalisms [101].

Geometric algebra (GA) can be considered a high-level mathematical language for geometric reasoning. As such it is very well suited for general problems in robotics. GA unifies the geometric understanding of screw theory, the thoroughness of Lie Algebra and the simplicity of spatial algebra. The representational advantage of geometric algebra is that its elements directly represent geometric objects that can be manipulated by algebraic operations. Complex relations and algorithms can be formulated in a simplified and coordinate-independent way. Furthermore, the existence of different geometric primitives in the same algebra allows for the uniform definition of distance functions, which we will later show in the examples for solving inverse kinematics problems. Dual quaternion algebra is closely related to GA due to their common roots in Clifford algebra. GA is, however, more general and can be defined over any dimension.

The story of geometric algebra in engineering is the story of an algebraic framework that greatly simplifies well-known equations, the most popular example being the Maxwell equations, which reduce to a single equation in geometric algebra. In [18], a survey is presented detailing this story of the development of GA in engineering ap-

plications and how it is a powerful geometric language that connects and unifies many mathematical concepts. Another recent survey showing how the applications of GA include physics, electrical engineering, computer graphics to quantum computing, neural networks, signal processing and robotics can be found in [104].

2.2. Basics of Geometric Algebra

Geometric algebras are a family of algebras that are defined over vector spaces to allow operations with subspaces. Geometric algebra is also known as Clifford algebra. In general, Clifford algebras can be defined for vector spaces over arbitrary fields, e.g. also over the complex numbers $\mathbb{C}$, but usually geometric algebra refers to a Clifford algebra that is generated by a vector space over the field of the real numbers $\mathbb{R}$. In this thesis, we only consider this case and first give some basic definitions. This introduction is not exhaustive and only touches on the surface of the most important concepts that are needed for understanding the rest of this thesis. For thorough introductions we recommend the excellent books [179, 144, 68].

Let $(\mathbb{R}^{p,q,r}, g)$ denote a quadratic space, i.e. a $(p+q+r)$-dimensional vector space $\mathbb{R}^{p,q,r}$ with a quadratic form g defined as $g : \mathbb{R}^{p,q,r} \times \mathbb{R}^{p,q,r} \to \mathbb{R}$. Here, p, q, and r are the number of basis vectors that square to 1, -1, and 0, respectively. Formally, a geometric algebra $\mathbb{G}_{p,q,r}$ is then defined as an associative algebra over the quadratic space $(\mathbb{R}^{p,q,r}, g)$ that forms a ring. The field $\mathbb{R}$ and the vector space $\mathbb{R}^{p,q,r}$ can be regarded as subspaces of $\mathbb{G}_{p,q,r}$. A general element $X \in \mathbb{G}_{p,q,r}$ is called a multivector. Given two multivectors $X, Y \in \mathbb{G}_{p,q,r}$ their algebraic product is denoted by juxtaposition, i.e. XY, and is called the *geometric product*.

Squaring a vector $\boldsymbol{x} \in \mathbb{R}^{p,q,r} \subset \mathbb{G}_{p,q,r}$ using the geometric product yields a scalar, i.e. $\boldsymbol{x}\boldsymbol{x} \in \mathbb{R}$. This property separates geometric algebra from other associative algebras. Given two vectors $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{G}_{p,q,r}$, their geometric product can be decomposed into a symmetric and an anti-symmetric part

$$\boldsymbol{x}_1\boldsymbol{x}_2 = \frac{1}{2}\left(\boldsymbol{x_1}\boldsymbol{x_2} + \boldsymbol{x_2}\boldsymbol{x_1}\right) + \frac{1}{2}\left(\boldsymbol{x_1}\boldsymbol{x_2} - \boldsymbol{x_2}\boldsymbol{x_1}\right) = \boldsymbol{x}_1 \cdot \boldsymbol{x}_2 + \boldsymbol{x}_1 \wedge \boldsymbol{x}_2 \tag{2.1}$$

The symmetric part is the *inner product* of vectors

$$\boldsymbol{x}_1 \cdot \boldsymbol{x}_2 \triangleq \frac{1}{2}\left(\boldsymbol{x_1}\boldsymbol{x_2} + \boldsymbol{x_2}\boldsymbol{x_1}\right) = \frac{1}{2}\left((\boldsymbol{x}_1 + \boldsymbol{x}_2)^2 - \boldsymbol{x}_1^2 - \boldsymbol{x}_2^2\right). \tag{2.2}$$

The anti-symmetric part is the *outer product* of vectors

$$\boldsymbol{x}_1 \wedge \boldsymbol{x}_2 \triangleq \frac{1}{2}\left(\boldsymbol{x_1}\boldsymbol{x_2} - \boldsymbol{x_2}\boldsymbol{x_1}\right). \tag{2.3}$$

The inner product is related to the metric of the algebra, whereas the outer product spans vectors to k-vectors, where k refers to the number of linearly independent basis vectors. For example, the quantity $X_{12} = \boldsymbol{x}_1 \wedge \boldsymbol{x}_2$ is called a bivector.

Denoting the set of basis vectors of $\mathbb{R}^{p,q,r}$ as $\bar{\mathbb{R}}^{p,q,r}$ and considering the geometric product of the basis vector $\boldsymbol{e}_i \in \bar{\mathbb{R}}^{p,q,r}$, all possible combinations form the algebraic basis $\bar{\mathbb{G}}_{p,q,r}$ of $\mathbb{G}_{p,q,r}$. Consequently, this basis then has 2^{p+q+r} elements which are called basis blades. Let $\mathbb{E} \subset \bar{\mathbb{R}}^{p,q,r}$, then $\boldsymbol{e}_{\mathbb{E}}$ denotes the basis blade

$$\boldsymbol{e}_{\mathbb{E}} \triangleq \bigwedge_{k=1}^{|\mathbb{E}|} \mathbb{E}\left[k\right]. \tag{2.4}$$

For example, if $\mathbb{E} = (\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3)$, then $\boldsymbol{e}_{\mathbb{E}} = \boldsymbol{e}_1 \wedge \boldsymbol{e}_2 \wedge \boldsymbol{e}_3$ and by convention it is written as $\boldsymbol{e}_{123}$. A general multivector $X \in \mathbb{G}_{p,q,r}$ then is the linear combination of basis blades. As a vector space (not as an algebra) $\mathbb{G}_{p,q,r}$ is isomorphic to the exterior algebra $\bigwedge^{p,q,r}$, meaning that it is a graded algebra. Here, the *grade* of a basis blade $\boldsymbol{e}_{\mathbb{E}} \in \mathbb{G}_{p,q,r}$ with $\mathbb{E} \subset \bar{\mathbb{R}}^{p,q,r}$ is defined as

$$\mathrm{gr}(\boldsymbol{e}_{\mathbb{E}}) = |\mathbb{E}|,$$

i.e. the grade refers to the number of basis vectors in a blade. The highest grade basis blade is called the pseudoscalar, and is denoted as I. Let $\boldsymbol{e}_i \triangleq \bar{\mathbb{G}}_{p,q,r}\left[i\right]$ then the *grade projection* is defined as

$$\left\langle \boldsymbol{e}_i \right\rangle_k = \begin{cases} \boldsymbol{e}_i, & \mathrm{gr}(\boldsymbol{e}_i) = k \\ 0, & \mathrm{gr}(\boldsymbol{e}_i) \neq k \end{cases}. \tag{2.5}$$

Given this definition, scalars are homogeneous multivectors of grade 0, i.e. $\mathbb{R} = \left\langle \mathbb{G}_{p,q,r} \right\rangle_0$, and vectors are homogeneous multivectors of grade 1, i.e. $\mathbb{R}^{p,q,r} = \left\langle \mathbb{G}_{p,q,r} \right\rangle_1$.

Using the grade projection, we can extend the inner and outer products to general multivectors $X_1, X_2 \in \mathbb{G}_{p,q,r}$. The inner product for general multivectors is defined as

$$\left\langle X_1 \right\rangle_k \cdot \left\langle X_2 \right\rangle_l = \left\langle X_1 X_2 \right\rangle_{|k-l|}, \tag{2.6}$$

and the outer product as

$$\left\langle X_1 \right\rangle_k \wedge \left\langle X_2 \right\rangle_l = \left\langle X_1 X_2 \right\rangle_{k+l}. \tag{2.7}$$

Also based on the geometric product, we can define other products, in particular the commutator product

$$X \times Y = \frac{1}{2}(XY - YX), \tag{2.8}$$

and the anti-commutator product

$$X \bar{\times} Y = \frac{1}{2}(XY + YX). \tag{2.9}$$

For non-degenerate geometric algebras, i.e. $r = 0$, we can define the *dual* of a basis blade as

$$\boldsymbol{e}_i^* = \boldsymbol{e}_e I^{-1}, \tag{2.10}$$

where I denotes the unit pseudoscalar of $\mathbb{G}_{p,q}$.

Later, for applying transformations to multivectors, we will need an operator called the *reverse* of a multivector. Let $X \in \mathbb{G}^k_{p,q,r}$ be a blade, i.e. $X = \bigwedge_{i=1}^k \boldsymbol{x}_i$, then the reverse of X, denoted as $\widetilde{X}$, is defined as

$$\widetilde{X} = \bigwedge_{i=k}^{1} \boldsymbol{x}_i = (-1)^{k(k-1)/2} X. \tag{2.11}$$

2.2.1. Example: 2D Geometric Algebra

In order to further motivate geometric algebra and to better explain the above definitions, we introduce here the geometric algebra of the two-dimensional Euclidean plane. The underlying quadratic space is $\mathbb{R}^2$, which means it has two basis vectors $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ and both of them square to 1. The geometric algebra of this space is consequently denoted as $\mathbb{G}_2$, and its algebraic basis is

$$\bar{\mathbb{G}}_2 = (1, \boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_{12}). \tag{2.12}$$

Clearly, the pseudoscalar of $\mathbb{G}_2$ is $I = \boldsymbol{e}_{12}$. Evaluating its square yields

$$\begin{aligned} \boldsymbol{e}_{12}^2 &= \boldsymbol{e}_{12}\boldsymbol{e}_{12} = (\boldsymbol{e}_1 \wedge \boldsymbol{e}_2)(\boldsymbol{e}_1 \wedge \boldsymbol{e}_2) && (2.13)\\ &= -\boldsymbol{e}_{12}\boldsymbol{e}_{21} = -(\boldsymbol{e}_1 \wedge \boldsymbol{e}_2)(\boldsymbol{e}_2 \wedge \boldsymbol{e}_1) && (2.14)\\ &= -\boldsymbol{e}_1\boldsymbol{e}_1 && (2.15)\\ &= -1. && (2.16) \end{aligned}$$

This is a very important result, because we have effectively introduced complex numbers simply by definition of a real geometric algebra. Other important isomorphisms can be found as $\mathbb{G}_0 = \mathbb{R}, \mathbb{G}_{0,1} = \mathbb{C}, \mathbb{G}_{0,0,1} = \mathbb{D}$, and $\mathbb{G}_{0,2} = \mathbb{H}$.

2.2.2. Versors

For a non-degenerate geometric algebra $\mathbb{G}_{p,q}$, the set of multivectors constructed from the geometric product of $n \in \mathbb{N}^+$ invertible vectors is called the set of versors, i.e. $V \prod_{i=1}^n \boldsymbol{x}_i$. Combined with the geometric product they form the Clifford group $\Gamma(p, q)$. The versors are invertible, i.e. $VV^{-1} = 1$. When restricted to be of unit norm, we find the $Pin(p, q)$ group and the inverse becomes the reverse $V^{-1} = \widetilde{V}$. The subgroup when n is even, is called the $Spin(p, q)$ group. The unit vectors $\boldsymbol{x}_i$ that form the versors can be seen as

hyperplanes in $\mathbb{R}^{p,q}$. Therefore, the $Pin(p,q)$ and $Spin(p,q)$ group represent arbitrary orthogonal transformations as product of vectors, since by the Carthan-Dieudonné theorem orthogonal transformations are found as multiple reflections in hyperplanes. More concretely, these groups are double covers of the Lie groups $O(p,q)$ and $SO(p,q)$, respectively, i.e. the orthogonal and special orthogonal groups. Transforming an arbitrary multivector $X \in \mathbb{G}_{p,q}$ is achieved via a sandwiching operation

$$Y = VX\widetilde{V}. \tag{2.17}$$

This operation is grade preserving, i.e. it leaves the number of k basis vectors in outer product representation unchanged and thus does not change what the multivectors represent. Herein lies one of the advantages of geometric algebra, as it extends linear transformations naturally from vectors to the entire algebra, i.e. transformations can be applied to all multivectors in a uniform manner, they are a more general representation of rigid body transformations and can also be used to transform all geometric primitives that are part of the algebra and not just vectors. This extension is called outermorphism.

2.3. Conformal Geometric Algebra

This thesis uses the specific variant of geometric algebra that is known as Conformal Geometric Algebra (CGA) and denoted as $\mathbb{G}_{4,1}$. Accordingly, the underlying vector space of CGA is $\mathbb{R}^{4,1}$, which extends the Euclidean space $\mathbb{R}^3$, characterized by the three basis vectors $\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3$, by two additional basis vectors $\boldsymbol{e}_4$ and $\boldsymbol{e}_5$, where $\boldsymbol{e}_4^2 = 1$ and $\boldsymbol{e}_5^2 = -1$. It is clearly a pseudo-Euclidean space, in fact it has a Minkowski signature. It can be understood as extending the Euclidean space $\mathbb{R}^3$ with a Minkowski plane $\mathbb{R}^{1,1}$, i.e. $\mathbb{R}^{4,1} = \mathbb{R}^3 \oplus \mathbb{R}^{1,1}$ [137]. In practice, the conformal model is found by a change of basis, which introduces two new basis vectors in order to obtain a null basis. These basis vectors are

$$\boldsymbol{e}_0 = \frac{1}{2}(\boldsymbol{e}_5 - \boldsymbol{e}_4) \quad \text{and} \quad \boldsymbol{e}_\infty = \boldsymbol{e}_4 + \boldsymbol{e}_5, \tag{2.18}$$

which can be understood as a point at the origin and one at infinity. This effectively leads to a non-orthogonal basis with a metric tensor of the following form

	$\boldsymbol{e}_0$	$\boldsymbol{e}_1$	$\boldsymbol{e}_2$	$\boldsymbol{e}_3$	$\boldsymbol{e}_\infty$
$\boldsymbol{e}_0$	0	0	0	0	-1
$\boldsymbol{e}_1$	0	1	0	0	0
$\boldsymbol{e}_2$	0	0	1	0	0
$\boldsymbol{e}_3$	0	0	0	1	0
$\boldsymbol{e}_\infty$	-1	0	0	0	0

(2.19)

Since the underlying vector space $\mathbb{R}_{4,1}$ is five-dimensional, the algebraic basis of CGA

consequently consists of 32 basis blades of grades zero to five. This structure can be seen in Figure 2.1.

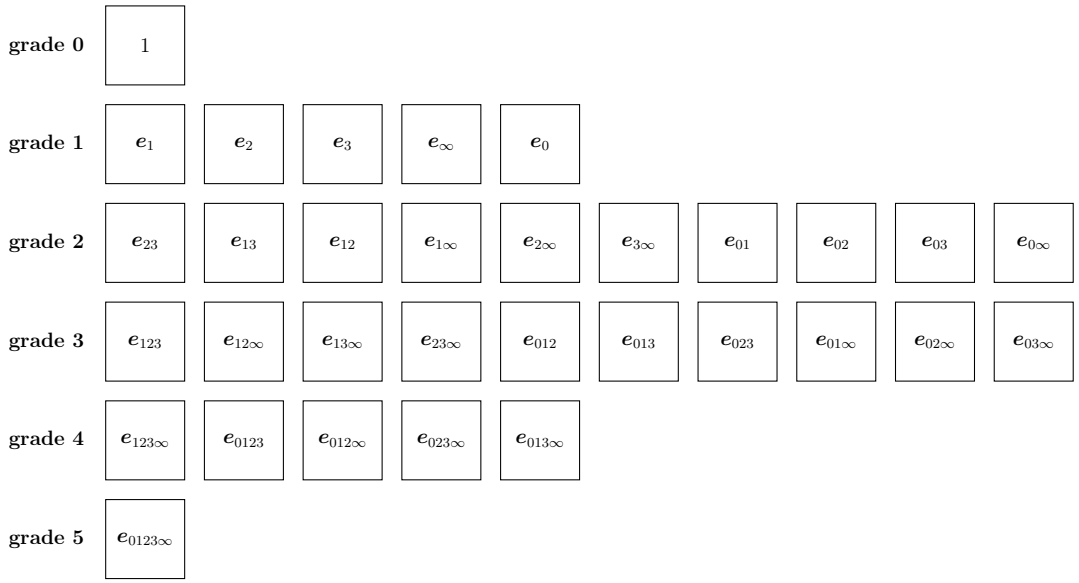


Figure 2.1.: Structure of conformal geometric algebra with the 32 basis blades, divided into the different grades. Grade 0 and 5 are the scalar and pseudoscalar, respectively. Grade 1 are vectors. Grades 2 to 4 are called bi-, tri- and quadvectors.

2.3.1. Geometric Primitives

Points are the basic geometric primitives that can be used to construct others by the spanning operation of the outer product. Euclidean points $\boldsymbol{x}$ are embedded in CGA by using the conformal embedding

$$P(\boldsymbol{x}) = \boldsymbol{e}_0 + \boldsymbol{x} + \frac{1}{2}\boldsymbol{x}^2\boldsymbol{e}_\infty. \tag{2.20}$$

These conformal points form the basic building blocks for geometric primitives that can be represented in the algebra. Note that, this nonlinear embedding turns flat Euclidean space into a parabolic space. Furthermore, this embedding is similar to how we traditionally embed vectors in $\mathbb{R}^3$ into $\mathbb{R}^4$ when using homogeneous coordinates.

In general, geometric primitives, such as lines, circles and spheres, can be constructed from conformal points using the outer product, i.e.

$$X = \bigwedge_{i=1}^{n} P_i. \tag{2.21}$$

In the above equation, depending on the number of points n and the presence of the point at infinity $\boldsymbol{e}_\infty$, different geometric primitives can be constructed:

a line can be constructed from two points passing through it and the point at infinity

$$L = P_1 \wedge P_2 \wedge \boldsymbol{e}_\infty; \tag{2.22}$$

a circle can be constructed from any three distinct points lying on its orbit

$$C = P_1 \wedge P_2 \wedge P_3; \tag{2.23}$$

a plane can be constructed from three points and a point at infinity

$$E = E_1 \wedge E_2 \wedge E_3 \wedge \boldsymbol{e}_\infty; \tag{2.24}$$

a sphere can be constructed from four points.

$$S = E_1 \wedge E_2 \wedge E_3 \wedge E_4. \tag{2.25}$$

The geometric primitives of CGA are in general nullspace representations with respect to either the inner (IPNS) or the outer (OPNS) product, meaning that a geometric primitive is defined by the set of all Euclidean points that result in zero upon multiplication when embedded in CGA, i.e.

$$\text{IPNS: } \mathbb{NI}_G(\boldsymbol{A}) = \{\boldsymbol{x} \in \mathbb{R}^3 : \mathcal{C}(\boldsymbol{x}) \cdot \boldsymbol{A} = 0\}, \tag{2.26}$$

$$\text{OPNS: } \mathbb{NO}_G(\boldsymbol{X}) = \{\boldsymbol{x} \in \mathbb{R}^3 : P(\boldsymbol{x}) \wedge \boldsymbol{X} = 0\}. \tag{2.27}$$

The IPNS and OPNS representations are dual to each other. Duality in this case means multiplication with the pseudo-scalar. 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.

We show the subspaces that several geometric primitives occupy within the algebra in Figure 2.2.

Operators using Geometric Primitives

As part of the geometric algebra, the geometric primitives can be used in algebraic expressions that have geometrically meaningful interpretations. For example, the projection of a point P to another geometric primitive X is achieved by the general formula

$$P' = (P \cdot X)X^{-1}. \tag{2.28}$$

Using what is known as the *meet* operator, it is also possible to calculate intersections

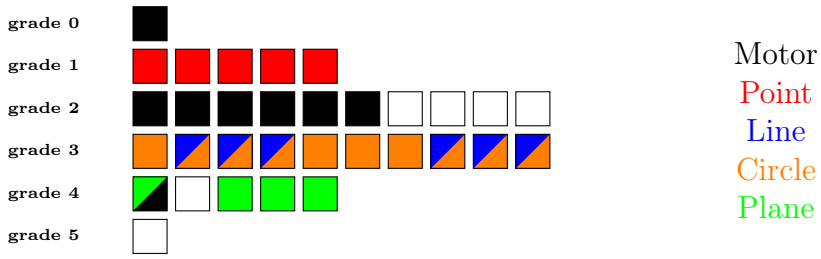


Figure 2.2.: Non-zero elements of various geometric primitives in their primal representations in conformal geometric algebra. Boxes represent basis blades and colored boxes represent the possible non-zero blades of the geometric primitive with the matching color. It can be seen that of the 32 basis blades only a sparse number is used for the representations. Note that geometric primitives are single-grade objects, while transformations are mixed-grade. The boxes correspond to the basis blades that are shown in Figure 2.1

between any two geometric primitives

$$Y = (X_1^* \wedge X_2^*)^*. \tag{2.29}$$

Here, it is not required to additionally consider edge cases. The resulting multivector retains a geometric meaning, e.g. when a line meets a sphere there are three possibilities:

the line intersects the sphere, in which case Y is a point pair;
the line is tangential to the sphere, which results in a single point;
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. Similarly, in the case of a line and a circle, it is not necessary to check whether the line is tangential, intersecting the circle twice or not all. Equation (2.29) will always return a meaningful geometric primitive that conveys the information of these different cases.

The geometric primitives can also directly be used for geometric operations such as reflections and projections, which result in rigid body motions. Here, two consecutive reflections on intersecting planes result in a rotation and on parallel planes in a translation.

2.3.2. Transformation Groups in CGA

In general, since the orthogonal group $O(n+1,1)$ is isomorphic to the conformal group $C(n)$, i.e. the group of angle-preserving transformations, CGA contains conformal transformations via the group $Pin(4,1)$. For the purpose of this thesis, we neglect pure reflections and special conformal transformations. As the relevant transformation groups,

we only consider subgroups that are formed by an even number of unit vectors, i.e. subgroups of $Spin(4,1)$. In particular, we only introduce the subgroups formed by rotations, translations, and uniform scaling.

Rotation Group

The group of rotations in three-dimensional Euclidean space is usually represented by the special orthogonal group $\boldsymbol{SO}(3)$, i.e. a matrix Lie group. The group $Spin(3)$ is its double-cover and can be represented as unit quaternions. In CGA, it is the rotors that form an isomorphic group to unit quaternions and we denote them here as $\mathcal{R}$. Their Lie algebra is the bivector algebra $\mathbb{B}_R = \text{span}\{\boldsymbol{e}_{12}, \boldsymbol{e}_{13}, \boldsymbol{e}_{23}\}$. Given the elements $R \in \mathcal{R}$ and $B_R \in \mathbb{B}_R$, the exponential map $\exp_{\mathcal{R}} : \mathbb{B}_R \to \mathcal{R}$ and its inverse the logarithmic map $\log_{\mathcal{R}} : \mathcal{R} \to \mathbb{B}_R$ are

$$R = \exp(B_R) = \cos\left(\frac{1}{2}\|B_R\|\right) - \sin\left(\frac{1}{2}\|B_R\|\right)\|B_R\|^{-1}B_R, \tag{2.30}$$

and

$$B_R = \log(R) = \frac{-2\cos^{-1}\left(\langle R\rangle_0\right)}{\sin\left(\cos^{-1}\left(\langle R\rangle_0\right)\right)}\langle R\rangle_2 . \tag{2.31}$$

Translation Group

The translation group of $\mathbb{R}^3$ is the Euclidean space itself under the addition operation, i.e. $(\mathbb{R}^3, +)$, which is often shortened to simply $\mathbb{R}^3$. In CGA, this group can be represented in versor form. Here, we denote the translation group containing all translation versors in CGA as $\mathcal{T}$. Note that, unlike the rotation group in CGA, the group $\mathcal{T}$ is not a double-cover of $(\mathbb{R}^3, +)$, since $(\mathbb{R}^3, +)$ is already a simply-connected group. The Lie algebra of the group $\mathcal{T}$ is the bivector algebra $\mathbb{B}_T = \text{span}\{\boldsymbol{e}_{1\infty}, \boldsymbol{e}_{2\infty}, \boldsymbol{e}_{3\infty}\}$. Given the elements $T \in \mathcal{T}$ and $B_T \in \mathbb{B}_T$, the exponential map $\exp_{\mathcal{T}} : \mathbb{B}_T \to \mathcal{T}$ and its inverse the logarithmic map $\log_{\mathcal{T}} : \mathcal{T} \to \mathbb{B}_T$ are

$$T = \exp(B_T) = 1 - \frac{1}{2}B_T, \tag{2.32}$$

and

$$B_T = \log(T) = -2\langle T\rangle_2 . \tag{2.33}$$

In general, the translation bivector B_T can be found from a Euclidean vector $\boldsymbol{t} \in \mathbb{R}^3$ as

$$B_T = \boldsymbol{t} \wedge \boldsymbol{e}_\infty. \tag{2.34}$$

Uniform Scaling Group

Uniform scaling is a transformation that preserves geometric similarity, i.e. the shape, proportions, angles and orientation as well as parallelism and collinearity are preserved while distances are changed by an isotropic scaling factor. Here, we restrict uniform scaling to positive scalars $\mathbb{R}^+$ to preserve the handedness as well. In CGA, the versor achieving this is called a dilator D and consequently the set of all dilators forms the dilation group $\mathcal{D}$, with its corresponding bivector Lie algebra $\mathbb{B}_D$. Given the elements $D \in \mathcal{D}$ and $B_D \in \mathbb{B}_D = \text{span}\{\boldsymbol{e}_{0\infty}\}$, the exponential map $\exp_{\mathcal{D}} : \mathbb{B}_D \to \mathcal{D}$ and its inverse the logarithmic map $\log_{\mathcal{D}} : \mathcal{D} \to \mathbb{B}_D$ are

$$D = \exp(B_D) = \cosh\left(\frac{1}{2}\|B_D\|\right) - \sinh\left(\frac{1}{2}\|B_D\|\right)\boldsymbol{e}_{0\infty}, \tag{2.35}$$

and

$$B_D = \log(D) = 2\cosh^{-1}\left(\langle D\rangle_0\right)\boldsymbol{e}_{0\infty}, \tag{2.36}$$

where the bivector B_D then relates to the scaling factor $d \in \mathbb{R}^+$ via

$$B_D = \log(d)\boldsymbol{e}_{0\infty}. \tag{2.37}$$

Note that the scaling is always with respect to the origin.

Rigid Transformation Group

The group of rigid transformations in Euclidean space is the most commonly used group in robotics. Traditionally, it is represented by the matrix Lie group $\boldsymbol{SE}(3)$ called the special Euclidean group. Alternative representations, such as dual quaternions, are representations of $Spin(3) \ltimes \mathbb{R}^3$, which is the double-cover of $\boldsymbol{SE}(3)$. Here, we denote this group as $\mathcal{M}$ and usually call its elements motors M. The group is found as $\mathcal{M} = \mathcal{R} \ltimes \mathcal{T}$, and we define the canonical decomposition of an element $M \in \mathcal{M}$ as

$$M = TR. \tag{2.38}$$

The Lie algebra of $\mathcal{M}$ is the bivector algebra $\mathbb{B}_M = \text{span}\{\boldsymbol{e}_{12}, \boldsymbol{e}_{13}, \boldsymbol{e}_{23}, \boldsymbol{e}_{1\infty}, \boldsymbol{e}_{2\infty}, \boldsymbol{e}_{3\infty}\}$ and an element $B_M \in \mathbb{B}_M$ is decomposed as

$$B_M = B_T + B_R. \tag{2.39}$$

Consequently, given the elements $M \in \mathcal{M}$ and $B_M \in \mathbb{B}_M$, the exponential map $\exp_{\mathcal{M}} : \mathbb{B}_M \to \mathcal{M}$ and its inverse the logarithmic map $\log_{\mathcal{M}} : \mathcal{M} \to \mathbb{B}_M$ are

$$M = \exp(B_M) = \exp(B_T)\exp(B_R), \tag{2.40}$$

and

$$B_M = \log(M) = \log(T) + \log(R). \tag{2.41}$$

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. Some visual examples of how motors are transforming geometric primitives are shown in Figure 2.3.

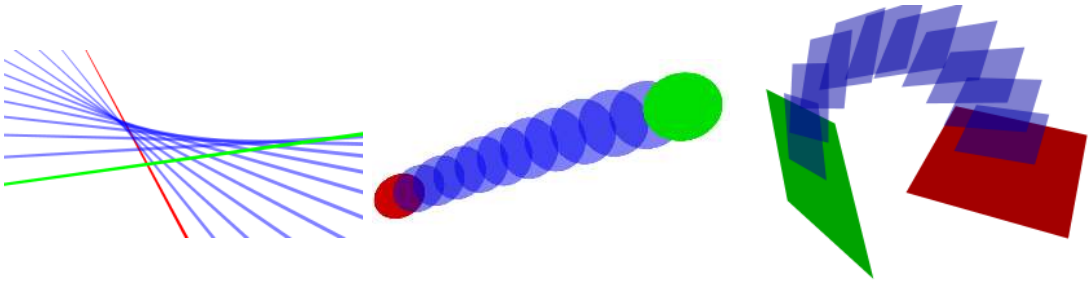


Figure 2.3.: Rigid body transformations of various geometric primitives. Red marks the initial primitive and green the final one, the primitives resulting from the trajectory of interpolated motors are shown in blue.

It has been shown in [21] 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

$$M(t) = \exp\left(\sum_{j=1}^{n} w_j(t) \log(M_j)\right). \tag{2.42}$$

The weights need to fulfill $\sum_{j=1}^{n} w_j = 1$ for each timestep, but can otherwise be chosen arbitrarily. In [28] this is exploited for mesh deformation.

2.3.3. Twists and Wrenches

In the previous section, we have identified a bivector $B_M \in \mathbb{B}_M$ as the screw axis of the motion defined by its exponential. Continuing to utilize the terminology of screw theory, the screws that carry velocity and force information are called twists and wrenches, respectively. Hence, twists are identified with the time derivatives of the bivectors that generate rigid body motions, i.e. their space is found as

$$\mathcal{V} \in \mathbb{B}_M = \text{span}\{\boldsymbol{e}_{23}, \boldsymbol{e}_{13}, \boldsymbol{e}_{12}, \boldsymbol{e}_{1\infty}, \boldsymbol{e}_{2\infty}, \boldsymbol{e}_{3\infty}\}, \tag{2.43}$$

where the six objects forming a twist are bivectors. The space of twists algebraically corresponds to the bivector dual lines that form the screw axes of motors.

Wrenches, on the other hand, are usually called co-screws, meaning that there is a certain duality relationship between twists and wrenches. In matrix Lie algebra, however, this duality is not directly visible, since both twists and wrenches are simply 6-dimensional vectors. In conformal geometric algebra, this duality is explicitly found via multiplication with the conjugate pseudo-scalar $I_c = I\boldsymbol{e}_0$ [100]. Multiplication of I_c with a twist yields the space of wrenches as

$$\mathcal{W} \in \text{span}\{\boldsymbol{e}_{23}, \boldsymbol{e}_{13}, \boldsymbol{e}_{12}, \boldsymbol{e}_{01}, \boldsymbol{e}_{02}, \boldsymbol{e}_{03}\}. \tag{2.44}$$

Using these definitions, the inner product of a twist $\mathcal{V}$ and a wrench $\mathcal{W}$ reduces to the scalar product and calculates the power of the motion, i.e.

$$p = -\mathcal{V} \cdot \mathcal{W}. \tag{2.45}$$

In Lie theory, twists are elements of the Lie algebra and wrenches are elements of the dual Lie algebra. As per the above definitions, this duality relationship can be directly seen from the different bivector blades in the respective spaces. The elements also allow for a direct geometrical interpretation: the linear velocity part of twists (i.e. $\boldsymbol{e}_{1\infty}, \boldsymbol{e}_{2\infty}, \boldsymbol{e}_{3\infty}$) corresponds to a direction bivector, whereas the force part of wrenches (i.e. $\boldsymbol{e}_{01}, \boldsymbol{e}_{02}, \boldsymbol{e}_{03}$) corresponds to a tangent bivector. This geometric interpretation further clarifies why twists and wrenches transform differently under rigid body transformations, i.e. why in matrix Lie algebra the adjoint matrix $\boldsymbol{Ad}$ transforms twists and the dual adjoint matrix $\boldsymbol{Ad}^*$ transforms wrenches. Using conformal geometric algebra, this distinction is not necessary, since, by definition of the algebra, direction and tangent bivectors (i.e. linear velocities and forces) multiply differently using the geometric product. Hence, a motor can be used to transform both twists and wrenches according to Equation (2.17), which means it simultaneously represents the adjoint and the dual adjoint operation.

Similarly, a unified expression for the Lie bracket can be found in conformal geometric algebra. The Lie bracket is a linear mapping between elements of the Lie algebra. For twists acting on twists or wrenches, this mapping can be found as

$$\mathcal{V}' = \mathcal{V}_1 \times \mathcal{V}_2 \quad \text{and} \quad \mathcal{W}' = \mathcal{V} \times \mathcal{W}, \tag{2.46}$$

where $\times$ is the commutator product that is defined in Equation (2.8). The implications of these definitions are very interesting, since CGA simultaneously clarifies the duality relationship of twists and wrenches, by removing the ambiguity of what a 6-dimensional vector represents through an algebraically determined difference. Furthermore, it also unifies their treatment by having the same adjoint operations.

2.4. Discussion

2.4.1. Computational Properties of Motors

When comparing the computational properties of motors in CGA to matrices, there are several important quantities to consider, i.e. the required memory, the number of floating-point operations for transformations and for composition. First of all, in terms of memory, it is easy to see that motors requiring the storage of 8 floats are a more compact representation than matrices that use at least 12 floats, since the bottom row is constant and can be treated separately. The scenario in which matrices are more efficient than motors is if a large number of vectors needs to be transformed. In this case, matrices generally require less floating-point operations. For composition, on the other hand, i.e. chaining multiple transformations, motors and dual quaternions are more efficient [62]. So depending on the scenario, one or the other representation might be preferred. Converting motors to matrices, however, is easy and relatively cheap, so in general it could be beneficial to store transformations as motors and to chain them in this form for, e.g. the computation of the forward kinematics, and to only convert them to matrices if a large number of points needs to be transformed. More analysis needs to be performed on this, since also it might end up being worth to do slightly more computation if it means to move less data around. A study on the use of the different representations for robot kinematics has been performed in [62] and a similiar one should be made for the dynamics.

Other advantages that motors and dual quaternions share over transformation matrices is that they are simpler type invariants, i.e. violations of the unit constraint are much easier to detect and corrections are much cheaper to enforce, i.e. re-normalizing a motor or a unit dual quaternion is a lot easier than re-orthonormalizing a matrix. Furthermore, in some applications, it is necessary to interpolate between transformations, which is also much easier to achieve with motors than with matrices.

Due to the widespread usage of matrices, modern systems use highly optimized algorithms for matrix operations and the architecture around graphics processing units also helps to parallelize these operations. Hence, an argument could be made for using matrices in the implementation. For this, we want to point out that in general, for every geometric algebra as a real, associative algebra, one can find an isomorphic matrix algebra. In the case of CGA $\mathbb{G}_{4,1}$ it is the algebra of 4×4 complex matrices, i.e. $\mathbb{G}_{4,1} = \mathbb{C}^{4\times4}$. This means that the presented algorithms could also be implemented entirely using (sparse) matrices as opposed to the bitset-based multivector implementation that we utilized in this thesis.

2.4.2. Comparison to Spatial Vector Algebra

Spatial vector algebra is a framework that is designed to treat screws as unified 6-dimensional vectors. It requires twelve basis vectors to form two vector spaces, one for motions, i.e. twists, and one for forces, i.e. wrenches. The bases of the two vector spaces are made dual to each other by using a dual coordinate system known as Plücker coordinates [74]. Instead of an inner product, the algebra defines a scalar product between the two spaces that, like the inner product of Equation (2.45), yields the power of the motion. The equivalent of the Lie bracket that we introduced for CGA in Equations (2.46) and (5.14) is implemented in spatial vector algebra using the definition of two separate cross products, in order to treat the two vector spaces of motions and forces differently. Spatial vector algebra uses mathematical constructions and definition that are sometimes not seen in the implementation, i.e. the matrices that are used in the implementation do not encode the distinction between screws and coscrews.

3. Robot Kinematics and Dynamics

This chapter introduces robot kinematics and dynamics from the perspective of conformal geometric algebras.

Publication Note

The material presented in this chapter is adapted from the following publications:

Tobias Löw and Sylvain Calinon. "Geometric Algebra for Optimal Control With Applications in Manipulation Tasks". In: *IEEE Transactions on Robotics* 39.5 (2023), pp. 3586–3600. DOI: 10.1109/TRO.2023.3277282

Tobias Löw and Sylvain Calinon. "Recursive Forward Dynamics of Serial Kinematic Chains Using Conformal Geometric Algebra". In: *Workshop on Algorithmic Foundations of Robotics (WAFR)*. 2024

3.1. Introduction

This chapter describes the computation of the robot kinematics and dynamics in conformal geometric algebra. Since these are important concepts in robotics, naturally, there have been many algorithms and implementations. There are many off-the-shelf robot simulators available using classical formulations, such as Raisim [106], Isaac Sim [139], Gazebo [127], Mujoco [210], Bullet [58] or Coppelia Sim [212]. Furthermore, there are several software libraries that, while not being full simulation engines, allow for the efficient computation of the robot dynamics, e.g. Pinocchio [50], KDL [200] and RBDL [77].

Researchers have realized before that mathematical tools from geometry can lead to simplified and unified treatments for robot modeling. Featherstone's seminal work [76] used spatial algebra for computing the dynamics recursively. By treating $\boldsymbol{SE}(3)$ as a Lie group and adopting basic ideas from Riemannian geometry, a geometric variant of the robot dynamics was derived in [176]. That work showed that by looking at the problem through the lens of geometry, various impractical notational conventions can be avoided and that the connections to differential geometry lead to easily factorizable and differentiable equations, making it very attractive for optimization and optimal control. These ideas of exploiting Lie groups and Lie algebras were extended further to show the natural emergence of a matrix factorization of the mass matrix and its inverse [182], where a recursive algorithm is embedded within its structure. Furthermore, the inherent invariance w.r.t the reference frame allows for arbitrary frames in the kinematic modeling [164]. Our formulation in CGA not only retains this coordinate invariance, but actually uses a double covering group of $\boldsymbol{SE}(3)$ for its computations and that the algebra contains more classes of transformations including non-rigid ones.

The theory of dual quaternion algebra presents another paradigm to approaching problems in robotics from a geometric perspective. There are works that are describing the robot dynamics using dual quaternion algebra, currently in the form of Newton-Euler type dynamics, which are less efficient for the computation of the accelerations [6]. A notable insight from that work is that by combining the geometric perspective from screw theory, the thoroughness of Lie algebra and a simple algebraic framework in the form of dual quaternion algebra leads to simplified and more general expressions. This insight seamlessly translates to geometric algebra due to their common roots in Clifford algebra. Hence, there are also works that utilize geometric algebra to derive a Newton-Euler type algorithm for robot inverse dynamics and control of manipulators [23] and multicopters [11]. In contrast to those works, we are presenting a recursive forward dynamics algorithm following Featherstone's formalism of the articulated body algorithm. In order to give a complete overview and to integrate our inertia tensor formulation into the Newton-Euler algorithm, we are also showing the inverse dynamics. Previous work in geometric algebra also looked at the inertia tensor and showed that it was fully contained within the geometric algebra $\mathbb{G}_{0,6,2}$ [188]. Since this algebra is comparatively

large, that work asked the question regarding the minimal algebra containing the adjoint operation of $\boldsymbol{SE}(3)$. While the presented inertia tensor and transformations of it require multiple operations, the proposed approach is fully contained within the algebra.

Hence, our contributions are as follows:

we define an inertia tensor using elements from conformal geometric algebra,
we formulate the articulated body inertia in conformal geometric algebra,
we formulate the recursive forward dynamics in conformal geometric algebra,
we implement the presented algorithms in an open-source library *gafro*[1].

The rest of the chapter is organized as follows: Section 3.2 introduces the mathematical background, in Section 3.4 we show our formulation of the inertia tensor in CGA, in Section 3.5 we then introduce the robot dynamics in CGA, and in Section 3.6 we show numerical results.

3.2. Background

3.2.1. Robot Dynamics

The manipulator equation using classical linear algebra for computing the dynamics of the system is

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau} - \boldsymbol{\tau}_{\text{ext}}, \tag{3.1}$$

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 $\boldsymbol{\tau}_{ext}$ are the external torques. The problem of forward dynamics then arises when solving Equation (3.1) for the joint accelerations $\ddot{\boldsymbol{q}}$, i.e.

$$\ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}(\boldsymbol{q})\Big(\boldsymbol{\tau} - \boldsymbol{\tau}_{\text{ext}} - \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \boldsymbol{g}(\boldsymbol{q})\Big). \tag{3.2}$$

This matrix algebra approach to solving the forward dynamics therefore requires finding the inverse of the generalized mass matrix $\boldsymbol{M}^{-1}(\boldsymbol{q})$, which can be computationally demanding. Hence, the previous work on deriving recursive algorithms for the forward dynamics showed how the inverse mass matrix could be efficiently factorized, alleviating the need of matrix inversion. This concept translates to the formulation of the forward dynamics in CGA, since we also don't compute the mass matrix or its inverse explicitly.

3.2.2. Torque Control

Standard practice for controlling robot manipulators is using torque commands. Often, this is achieved by converting acceleration commands found from planning methods to

[1] `https://gitlab.com/gafro`

torques. This can be realized using an inverse dynamics controller, the required control command can thus be found as

$$\boldsymbol{u}_\tau = \boldsymbol{\tau}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}_d) + \boldsymbol{K}_p(\boldsymbol{q}_d - \boldsymbol{q}) + \boldsymbol{K}_d(\dot{\boldsymbol{q}}_d - \dot{\boldsymbol{q}}), \tag{3.3}$$

where the torque vector $\boldsymbol{\tau}$ is computed using inverse dynamics. $\boldsymbol{K}_p$ and $\boldsymbol{K}_d$ are the stiffness and damping gains, respectively.

3.3. Kinematics of Serial Manipulators

In this section, we present how geometric algebra 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.

3.3.1. Forward Kinematics of Serial Manipulators

The forward kinematics of a kinematic chain of n joints can easily be defined using motors [22]. Assuming that we only have revolute joints, the forward motor $M(\boldsymbol{q})$, given the configuration $\boldsymbol{q}$, can be computed with

$$M(\boldsymbol{q}) = \prod_{i=1}^{n} M_i(q_i) = \prod_{i=1}^{n} M_{F,i} R_i(q_i). \tag{3.4}$$

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

$$R_i(q_i) = \exp\left(-\frac{1}{2} q_i B_i\right), \tag{3.5}$$

where the bivectors B_i essentially represent the rotation planes of the joints. These quantities can easily be found using e.g. DH-parameters [195].

3.3.2. 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 geometric algebra.

Using an arbitrary representation of the end-effector forward kinematics

$$\boldsymbol{\xi} = \boldsymbol{f}(\boldsymbol{q}), \tag{3.6}$$

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.

$$\boldsymbol{J}^A(\boldsymbol{q}) = \frac{\partial \boldsymbol{f}(\boldsymbol{q})}{\partial \boldsymbol{q}}, \tag{3.7}$$

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

$$\dot{\boldsymbol{\xi}} = \boldsymbol{J}^A(\boldsymbol{q})\dot{\boldsymbol{q}}. \tag{3.8}$$

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

$$\begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix} = \boldsymbol{J}^G(\boldsymbol{q})\dot{\boldsymbol{q}}. \tag{3.9}$$

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

$$\boldsymbol{J}^G(\boldsymbol{q}) = \boldsymbol{J}^M(\boldsymbol{\xi})\boldsymbol{J}^A(\boldsymbol{q}). \tag{3.10}$$

In geometric algebra the analytic Jacobian can be found as the derivative of the forward kinematics motor defined in Equation (3.4), i.e.

$$\boldsymbol{\mathcal{J}}^A(\boldsymbol{q}) = \frac{\partial M(\boldsymbol{q})}{\partial \boldsymbol{q}} = \left[\frac{\partial M(\boldsymbol{q})}{\partial q_1} \dots \frac{\partial M(\boldsymbol{q})}{\partial q_N} \right]. \tag{3.11}$$

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 i-th joint angle is

$$\frac{\partial M(\boldsymbol{q})}{\partial q_i} = M_1(q_1) \dots M_{F,i} \left(-\frac{1}{2} B_i \right) R_i(q_i) \dots M_N(q_N). \tag{3.12}$$

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

$$\boldsymbol{\mathcal{J}}^G(\boldsymbol{q}) = \begin{bmatrix} B_1' & \dots & B_n' \end{bmatrix}, \tag{3.13}$$

with the rotation bivectors

$$B_i' = M_1 \dots M_{i-1} M_{F,i} B_i \widetilde{M}_{F,i} \widetilde{M}_{i-1} \dots \widetilde{M}_1. \tag{3.14}$$

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 Equations (3.12) and (3.14) the relationship between the analytic and geometric

Jacobians in geometric algebra can easily be derived as

$$\boldsymbol{\mathcal{J}}^G_{kj}(\boldsymbol{q}) = -2\boldsymbol{\mathcal{J}}^A_{kj}(\boldsymbol{q})\widetilde{M}(\boldsymbol{q}). \tag{3.15}$$

The time derivative of the geometric Jacobian can be found using the time derivatives of the rotation bivectors $\dot{B}'_j$ as

$$\dot{\boldsymbol{\mathcal{J}}}^G(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \begin{bmatrix} \dot{B}'_1 & \cdots & \dot{B}'_N \end{bmatrix} = \left(\boldsymbol{\mathcal{J}}^{\times}(\boldsymbol{q})\dot{\boldsymbol{q}}\right)^{\top}, \tag{3.16}$$

where

$$\mathcal{J}^{\times}_{ij}(\boldsymbol{q}) = B'_i \times B'_j = \frac{1}{2}(B'_i B'_j - B'_j B'_i), \tag{3.17}$$

where the operator $\times$ is the commutator product from Equation (2.8), i.e. the Lie bracket.

Since the analytic Jacobian can be used to derive the time derivative of the end-effector motor

$$\dot{M}_{\boldsymbol{q}} = \boldsymbol{\mathcal{J}}^A_{\boldsymbol{q}}\dot{\boldsymbol{q}}, \tag{3.18}$$

and the geometric Jacobian for finding the end-effector twist $\mathcal{V}_{\boldsymbol{q},\dot{\boldsymbol{q}}}$

$$\mathcal{V}_{\boldsymbol{q},\dot{\boldsymbol{q}}} = \boldsymbol{\mathcal{J}}^G_{\boldsymbol{q}}\dot{\boldsymbol{q}}, \tag{3.19}$$

it is straightforward to relate the bivector time derivative of the end-effector motor $\dot{B}_{\boldsymbol{q}}$ to the end-effector twist $\mathcal{V}_{\boldsymbol{q},\dot{\boldsymbol{q}}}$

$$\mathcal{V}_{\boldsymbol{q},\dot{\boldsymbol{q}}} = -2\dot{B}_{\boldsymbol{q}} = -2\widetilde{M}_{\boldsymbol{q}}\dot{M}_{\boldsymbol{q}}. \tag{3.20}$$

3.3.3. Inverse Kinematics of Serial Manipulators

Using the expressions derived in Section 3.3.2, 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

$$\boldsymbol{q}^* = \arg\min_{\boldsymbol{q}} \left\| \log\left(\widetilde{M}_{\text{target}} M(\boldsymbol{q})\right) \right\|_2^2. \tag{3.21}$$

The forward kinematics motor $M(\boldsymbol{q})$ can be found using Equation (3.4). 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

$$\boldsymbol{J}_{\mathbb{B}}(\boldsymbol{q}) = \mathcal{E}^{\mathbb{B}\to\mathbb{R}^6}\left[\frac{\partial}{\partial q_i}\log\left(\widetilde{M}_{\text{target}}M(\boldsymbol{q})\right)\right]. \tag{3.22}$$

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

$$\boldsymbol{J}_{\mathbb{B}}(\boldsymbol{q}) = \boldsymbol{J}_{\mathcal{M}\to\mathbb{B}}(\boldsymbol{q})\boldsymbol{J}_{\mathcal{M}}(\boldsymbol{q}). \tag{3.23}$$

$\boldsymbol{J}_{\mathcal{M}}(\boldsymbol{q}) \in \mathbb{R}^{8\times N}$ is the embedding of the analytic Jacobian of Equation (3.11) multiplied by the target motor, i.e.

$$\boldsymbol{J}_{\mathcal{M}}(\boldsymbol{q}) = \mathcal{E}^{\mathcal{M}\to\mathbb{R}^8}\left[\widetilde{M}_{\text{target}}\boldsymbol{\mathcal{J}}^A(\boldsymbol{q})\right]. \tag{3.24}$$

$\boldsymbol{J}_{\mathcal{M}\to\mathbb{B}}(\boldsymbol{q}) \in \mathbb{R}^{6\times 8}$ 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 A.1.

With Equation (3.22) the Gauss-Newton step becomes

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k - \alpha\left(\boldsymbol{J}_{\mathbb{B}}(\boldsymbol{q})^\top\boldsymbol{J}_{\mathbb{B}}(\boldsymbol{q})\right)^{-1}\boldsymbol{J}_{\mathbb{B}}(\boldsymbol{q})^\top\boldsymbol{f}(\boldsymbol{q}_k), \tag{3.25}$$

where α is the line-search parameter. This shows how geometric algebra 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 geometric algebra.

3.4. Inertia Tensor

In this section, we are presenting our formulation of the general inertia tensor in CGA. This formulation is a direct extension of the definition of the rotational inertia tensor in dual quaternion algebra [6]. The rotational part can be identified in conformal geometric algebra with the bivector blades $\boldsymbol{e}_{23}, \boldsymbol{e}_{13}, \boldsymbol{e}_{12}$. We add the bivector blades $\boldsymbol{e}_{01}, \boldsymbol{e}_{02}, \boldsymbol{e}_{03}$, such that a general inertia element becomes

$$\mathcal{I} \in \text{span}\{\boldsymbol{e}_{23}, \boldsymbol{e}_{13}, \boldsymbol{e}_{12}, \boldsymbol{e}_{01}, \boldsymbol{e}_{02}, \boldsymbol{e}_{03}\}. \tag{3.26}$$

Note that an inertia element uses the same basis blades as a wrench that was defined in Equation (2.44). This follows from the fact that we want the inner product of an inertia element and twists to be the scalar product in order to compute the different

components of the resulting wrenches.

Hence, the inertia tensor $\boldsymbol{\mathcal{I}}$ consists of six bivector-valued inertia elements, one for each component of the resulting wrench. We define the geometric algebra inertia tensor as the union of all elements belonging to the blade index list $\mathbb{I}_{\mathcal{I}}$, i.e.

$$\boldsymbol{\mathcal{I}} = \{\mathcal{I}^{\boldsymbol{e}_k}\}_{\boldsymbol{e}_k \in \mathbb{I}_{\mathcal{I}}}, \tag{3.27}$$

where the upper blade index indicates which component of the resulting wrench it is responsible for. In accordance with the definition of a wrench, these blade indices therefore are

$$\mathbb{I}_{\mathcal{I}} = \{\boldsymbol{e}_{23}, \boldsymbol{e}_{13}, \boldsymbol{e}_{12}, \boldsymbol{e}_{01}, \boldsymbol{e}_{02}, \boldsymbol{e}_{03}\}. \tag{3.28}$$

3.4.1. Linear Mapping from Twist to Wrench

The inertia tensor is a linear operator that acts on twists and produces wrenches. This linear map can be found as the sum of the inner products of each inertia element with the given twist, i.e.

$$\mathcal{W} = \boldsymbol{\mathcal{I}}[\mathcal{V}] = -\sum_{\boldsymbol{e}_k \in \mathbb{I}_{\mathcal{I}}} (\mathcal{I}^{\boldsymbol{e}_k} \cdot \mathcal{V})\boldsymbol{e}_k. \tag{3.29}$$

At this point we would like to mention that we generally call the mathematical objects twists and wrenches according to the definion of the spaces in Equations (2.43) and (2.44), respectively. The physical interpretation, however, can of course be a mapping either from velocity to momentum or acceleration to force. In both cases, the mathematical bivector spaces are the same, hence we treat it using a unified terminology.

3.4.2. Rigid Body Transformation of Inertia Tensor

Often, it is required to view the inertia tensor from a frame that is different to the inertial frame. Hence, it is necessary to have a formulation of how an inertia tensor behaves under rigid body transformations. These are expressed using motors in CGA, hence, given a motor M, we define the transformation of an inertia tensor and denote this operation using the symbol $*$, i.e.

$$M * \boldsymbol{\mathcal{I}} = \left\{ M \left(\sum_{\boldsymbol{e}_k \in \mathbb{I}_{\mathcal{I}}} \langle M \mathcal{I}^{\boldsymbol{e}_k} \widetilde{M} \rangle_{\boldsymbol{e}_j} \boldsymbol{e}_k \right) \widetilde{M} \right\}_{\boldsymbol{e}_j \in \mathbb{I}_{\mathcal{I}}}, \tag{3.30}$$

where the operator $\langle \cdot \rangle_{\boldsymbol{e}_j}$ extracts the $\boldsymbol{e}_j$ blade component of the enclosed expression.

With equations (3.29) and (3.30), we can find the equivariance relationship

$$M\boldsymbol{\mathcal{I}}[\mathcal{V}]\widetilde{M} = (M * \boldsymbol{\mathcal{I}}) \left[M \mathcal{V} \widetilde{M} \right]. \tag{3.31}$$

Hence, transforming a wrench produced by applying an inertia tensor to a twist, gives the same wrench as first transforming the inertia and the twist individually and then applying the transformed inertia tensor to the transformed twist.

3.4.3. Spatial Inertia

Traditionally, the spatial inertia is a 6×6 symmetric matrix that contains the information about the rotational inertia, the mass and the center of mass. We show in this section how to obtain it in our CGA formalism, in order to give a more detailed explanation of the involved operations.

Given an arbitrary rigid body, we have its inertial parameters as the mass m, the motor describing the location of the center of mass M^{CoM} and the inertia matrix $\boldsymbol{I}$, i.e. a symmetric positive-definite matrix with the six independent components $i_{xx}, i_{yy}, i_{zz}, i_{xy}, i_{xz}, i_{yz}$. Thus, we find the general inertia tensor $\boldsymbol{\mathcal{I}}$ as a function of m and $\boldsymbol{I}$, i.e. the elements of $\boldsymbol{\mathcal{I}}$ are

$$\begin{aligned} \mathcal{I}^{\boldsymbol{e}_{23}} &= \quad i_{xx}\boldsymbol{e}_{23} - i_{xy}\boldsymbol{e}_{13} + i_{xz}\boldsymbol{e}_{12}, & \mathcal{I}^{\boldsymbol{e}_{01}} &= m\boldsymbol{e}_{01}, \\ \mathcal{I}^{\boldsymbol{e}_{13}} &= -i_{xy}\boldsymbol{e}_{23} + i_{yy}\boldsymbol{e}_{13} - i_{yz}\boldsymbol{e}_{12}, & \mathcal{I}^{\boldsymbol{e}_{02}} &= m\boldsymbol{e}_{02}, \\ \mathcal{I}^{\boldsymbol{e}_{12}} &= \quad i_{xz}\boldsymbol{e}_{23} - i_{yz}\boldsymbol{e}_{13} + i_{zz}\boldsymbol{e}_{12}, & \mathcal{I}^{\boldsymbol{e}_{03}} &= m\boldsymbol{e}_{03}. \end{aligned} \tag{3.32}$$

Note the sign change in certain elements of the inertia, that is due to the metric tensor of CGA, which led to our choice of basis bivectors to closely match and facilitate the implementation. Then, we find the spatial inertia tensor $\boldsymbol{\mathcal{I}}^{CoM}$ by transforming the inertia tensor $\boldsymbol{\mathcal{I}}$ using the motor describing the location of the center of mass M^{CoM}

$$\boldsymbol{\mathcal{I}}^{CoM} = M^{CoM} * \boldsymbol{\mathcal{I}}. \tag{3.33}$$

3.4.4. Manipulator Inertia

As mentioned, we are not assuming the inertia to be constant in this work, 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.

$$\boldsymbol{\mathcal{I}}(\boldsymbol{q}) = \sum_{i=0}^{N} \boldsymbol{\mathcal{B}}_i^{\top} \mathcal{I}(\boldsymbol{\mathcal{B}}_i). \tag{3.34}$$

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 j-th element of $\boldsymbol{\mathcal{B}}_i$ can thus be found as

$$B_{i,j} = \widetilde{R}_i(\boldsymbol{q}) \log^R \left(\boldsymbol{\mathcal{J}}^G_{ij}(\boldsymbol{q}) \right) R_i(\boldsymbol{q}). \tag{3.35}$$

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

The time derivative of the inertia matrix thus follows as

$$\dot{\boldsymbol{\mathcal{I}}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \sum_{i=0}^{N} \boldsymbol{\mathcal{B}}_i^\top \left(\mathcal{I}(\dot{\boldsymbol{\mathcal{B}}}_i) + \widetilde{R}_i(\boldsymbol{q}) \hat{B}_i^w R_i(\boldsymbol{q}) \right). \tag{3.36}$$

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

$$\dot{B}_{i,j} = \widetilde{R}_i(\boldsymbol{q}) \log^R \left(\dot{\boldsymbol{\mathcal{J}}}^G_{ij}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \right) R_i(\boldsymbol{q}). \tag{3.37}$$

The variable $\hat{B}_i^w$ from Equation (3.36) can be found as

$$\hat{B}_i^w = (I_3 B_i^w) \wedge \left(R_i(\boldsymbol{q}) \mathcal{I} \left(\widetilde{R}_i(\boldsymbol{q}) I_3 B_i^w R_i(\boldsymbol{q}) \right) \widetilde{R}_i(\boldsymbol{q}) \right). \tag{3.38}$$

Note that in this case I_3 stands for $\boldsymbol{e}_{123}$, which is the pseudoscalar of the Euclidean geometric algebra $\mathbb{G}_3$, which is a sub-algebra 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 non-zero in $\boldsymbol{e}_1$,$\boldsymbol{e}_2$ and $\boldsymbol{e}_3$. The outer product in Equation (3.38) 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 non-zero.

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 [68].

3.4.5. Articulated Body Inertia

One of the key concepts that was introduced by Featherstone for computing the dynamics of multibody systems is the Articulated Body Inertia (ABI). The ABI is the inertia that a body appears to have if it is part of a multi-body system [73]. It is an integral part for the efficient computation of the forward dynamics. Although the ABI is structurally and mathematically identical to the spatial inertia, they physically differ in that the spatial inertia converts from velocity to momentum and the ABI from acceleration to force [75].

The computation of the ABI is a recursive algorithm that iteratively adds the influence of child bodies to parent bodies. The recursion starts with the outermost body, where the ABI is identical to the spatial inertia tensor of the body. We present the CGA formulation of this computation in Algorithm 1. Here, $\boldsymbol{q}$ denotes the current joint positions, $\boldsymbol{\mathcal{I}}_j$ the link inertias, M_j the current joint motors and B_j the screw axes (as bivector) of the joints. The computed articulated body inertia of the links is $\hat{\boldsymbol{\mathcal{I}}}_j$.

Algorithm 1: Articulated Body Inertia

Input: $\boldsymbol{q}, \boldsymbol{\mathcal{I}}_j, M_j, B_j$
Output: $\hat{\boldsymbol{\mathcal{I}}}_j$
$\hat{\boldsymbol{\mathcal{I}}}_n \leftarrow M_n^{CoM} * \boldsymbol{\mathcal{I}}_n$
for $j = n-1$ **to** 1 **do**
  $\mathcal{W}_{B_{j+1}} = -\hat{\boldsymbol{\mathcal{I}}}_{j+1}[B_{j+1}](B_{j+1} \cdot \hat{\boldsymbol{\mathcal{I}}}_{j+1}[B_{j+1}])^{-1}$
  $\bar{\boldsymbol{\mathcal{I}}}_j = \left\{ \hat{\mathcal{I}}^{\boldsymbol{e}_k}_{j+1} + (B_{j+1} \cdot \hat{\mathcal{I}}^{\boldsymbol{e}_k}_{j+1})\mathcal{W}_{B_{j+1}} \right\}_{\boldsymbol{e}_k \in \mathbb{I}_{\mathcal{I}}}$
  $\hat{\boldsymbol{\mathcal{I}}}_j = M_j^{CoM} * \boldsymbol{\mathcal{I}}_j + M_{j+1} * \bar{\boldsymbol{\mathcal{I}}}_j$
end

3.5. Dynamics of Serial Manipulators

In geometric algebra, Equation (3.1) can be transformed to a simplified version, which was shown in [19]. 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 geometric algebra reformulation of Equation (3.1) that was derived in [19] and then present our extension with the joint position dependent inertia tensor. The elements of Equation (3.1) can be expressed as multivector matrices, where the generalized mass matrix becomes

$$\boldsymbol{\mathcal{M}}(\boldsymbol{q}) = \boldsymbol{\mathcal{I}}(\boldsymbol{q}) + \boldsymbol{\mathcal{V}}^{\top}(\boldsymbol{q})\boldsymbol{m}\boldsymbol{\mathcal{V}}(\boldsymbol{q}). \tag{3.39}$$

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

$$\boldsymbol{\mathcal{C}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} = \dot{\boldsymbol{\mathcal{I}}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{\mathcal{V}}^{\top}(\boldsymbol{q})\boldsymbol{m}\dot{\boldsymbol{\mathcal{V}}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}, \tag{3.40}$$

and the gravitational forces are

$$\boldsymbol{\mathcal{G}}(\boldsymbol{q}) = \boldsymbol{\mathcal{V}}^{\top}(\boldsymbol{q})\boldsymbol{m}\boldsymbol{\mathcal{G}}. \tag{3.41}$$

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 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

$$\boldsymbol{\mathcal{V}}_{j,k}(\boldsymbol{q}) = X_j^{CoM}(\boldsymbol{q}) \cdot B_k'. \tag{3.42}$$

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

$$\dot{\boldsymbol{\mathcal{V}}}_{j,k}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \Big(\boldsymbol{I}\boldsymbol{\mathcal{V}}(\boldsymbol{q})\dot{\boldsymbol{q}}\Big)\boldsymbol{\mathcal{J}}^G(\boldsymbol{q}) + \left(\boldsymbol{I}\boldsymbol{\mathcal{X}}^{CoM}(\boldsymbol{q})\right)\dot{\boldsymbol{\mathcal{J}}}^G(\boldsymbol{q}, \dot{\boldsymbol{q}}). \tag{3.43}$$

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 $\boldsymbol{I}\boldsymbol{\mathcal{X}}^{CoM}(\boldsymbol{q})$, where the matrix $\boldsymbol{\mathcal{X}}^{CoM}(\boldsymbol{q}) \in \mathbb{G}_{4,1}^{N\times 1}$ contains all centers of masses of the links.

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

$$\boldsymbol{\tau}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) = \boldsymbol{\tau}_{ext} + \boldsymbol{\mathcal{I}}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{\mathcal{I}}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{\mathcal{V}}^\top(\boldsymbol{q})\boldsymbol{m}\Big(\boldsymbol{\mathcal{V}}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{\mathcal{V}}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{\mathcal{G}}\Big), \tag{3.44}$$

and consequently the forward dynamics of a serial manipulator can be expressed as

$$\ddot{\boldsymbol{q}}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau}) = \Big(\boldsymbol{\mathcal{I}}(\boldsymbol{q}) + \boldsymbol{\mathcal{V}}^\top(\boldsymbol{q})\boldsymbol{m}\boldsymbol{\mathcal{V}}(\boldsymbol{q})\Big)^{-1}\Big(\boldsymbol{\tau} - \boldsymbol{\tau}_{ext} - \boldsymbol{\mathcal{V}}^\top(\boldsymbol{q})\boldsymbol{m}\Big(\dot{\boldsymbol{\mathcal{V}}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{\mathcal{G}}\Big) - \dot{\boldsymbol{\mathcal{I}}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\Big). \tag{3.45}$$

3.5.1. Recursive Inverse Dynamics

The recursive computation of the inverse dynamics of serial kinematic chains has already been presented in [20], where the authors used the variant known as motor algebra. We show the recursive algorithm here for the sake of completeness and consistency in the notation.

The problem of inverse dynamics is defined as calculating the joint torques $\boldsymbol{\tau}$ given the joint positions $\boldsymbol{q}$, velocities $\dot{\boldsymbol{q}}$ and accelerations $\ddot{\boldsymbol{q}}$. Solving this problem is essential for controlling robots in order to achieve a desired motion, such as in inverse dynamics control. Recursive algorithms that break down the problem into smaller sub-problems and solve them iteratively are efficient and widely used in robotics due to their ability to handle complex kinematic chains and their recursive nature allows for easy implementation.

The algorithm has two stages: the forward and the backward pass. Using the joint positions q_j, velocities $\dot{q}_j$ and accelerations $\ddot{q}_j$, the relative joint motors $M_{j,j-1}$, the joint

twists $\mathcal{V}_j$ and their time derivatives $\dot{\mathcal{V}}_j$ are computed in the forward pass. Here, B_j denote the joint axes and M_j^{CoM} the center of mass of the child link of the j-th joint, relative to the joint's axis of rotation. $M_{j,j-1}$ therefore relates the centers of mass of two consecutive links. Afterwards, in the backward pass, the wrenches acting on the joints $\mathcal{W}_j$ are computed using the inertia map $\boldsymbol{\mathcal{I}}_j$ as it was defined in Equation (3.29). The joint torques τ_j are then found by the inner product of the wrench and the joint axis. The entire algorithm is given in Algorithm 2.

Algorithm 2: Recursive Inverse Dynamics

Input: $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$
Output: $\boldsymbol{\tau}$
$\mathcal{V}_0 \leftarrow 0$
for $j = 1$ **to** n **do**
  $M_{j,j-1} = \widetilde{M}_j^{CoM} \widetilde{M}_j(q_j) M_{j-1}^{CoM}$ $\mathcal{V}_j = M_{j,j-1}\mathcal{V}_{j-1}\widetilde{M}_{j,j-1} + \dot{q}_j B_j$
  $\dot{\mathcal{V}}_j = M_{j,j-1}\dot{\mathcal{V}}_{j-1}\widetilde{M}_{j,j-1} + \dot{q}_j(B_j \times \mathcal{V}_j) + \ddot{q}_j B_j$
end
$\dot{\mathcal{V}}_{n+1} \leftarrow -g\boldsymbol{e}_{3i}$
$\mathcal{W}_{n+1} \leftarrow \mathcal{W}_{ext}$
for $j = n$ **to** 1 **do**
  $\mathcal{W}_j = \widetilde{M}_{j+1,j}\mathcal{W}_{j+1}M_{j+1,j} + \boldsymbol{\mathcal{I}}_j\left[\dot{\mathcal{V}}_j\right] - \mathcal{V}_j \times \boldsymbol{\mathcal{I}}_j\left[\mathcal{V}_j\right]$ $\tau_j = -\mathcal{W}_j \cdot B_j$
end

3.5.2. Recursive Forward Dynamics

The problem of forward dynamics is defined as finding the accelerations $\ddot{\boldsymbol{q}}$ given the joint positions $\boldsymbol{q}$, velocities $\dot{\boldsymbol{q}}$ and joint torques $\boldsymbol{\tau}$. It is a method to simulate the motion of articulated bodies given their physical properties and constraints.

The algorithm for recursively computing the forward dynamics in CGA is given in Algorithm 3. Note that the computation of the ABI of Algorithm 1 can be fully integrated in this algorithm and does not need to be performed separately. The algorithm consists of three recursions over the joints of the system, two forward and one backward recursions. In the first forward recursion, the spatial transformations of the joints are computed and the corresponding velocities are propagated through the kinematic chain. The transformations are denoted by the motors M_j and the velocities by the twists $\mathcal{V}_j$. The velocity and gravity induced accelerations and forces are then $\dot{\mathcal{V}}_j$ and $\mathcal{W}_{b,j}$, respectively. Afterwards, in the backward pass, those forces are combined with the inertial forces $\mathcal{W}_{B_j}$ (or technically momenta) induced by the articulated body inertias $\hat{\boldsymbol{\mathcal{I}}}_j$ and are propagated to parent joints. If present, external forces $\mathcal{W}_{ext}$ can also be taken into account in this step. Using these quantities, the bias joint accelerations $\hat{\ddot{q}}_j$ and torques

$\hat{\tau}_j$ are calculated, where the currently applied torques τ_j are also considered. Here, the quantities $\bar{\mathcal{W}}_j$ and $\hat{\mathcal{W}}_j$ are helper variables to reduce the complexity of a single line in the algorithm. Lastly, in the second forward pass, the correct joint accelerations $\ddot{\boldsymbol{q}}_j$ are computed.

Algorithm 3: Recursive Forward Dynamics

Input: $\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau}$
Output: $\ddot{\boldsymbol{q}}$
$\mathcal{V}_0 \leftarrow 0$
$\dot{\mathcal{V}}_0 \leftarrow -g\boldsymbol{e}_{3i}$
for $j = 1$ **to** n **do**
$M_j = M_j(q_j)$
$\mathcal{V}_j = \widetilde{M}_j \mathcal{V}_{j-1} M_j + \dot{q}_j B_j$
$\dot{\mathcal{V}}_j = \widetilde{M}_j \dot{\mathcal{V}}_{j-1} M_j + \dot{q}_j (B_j \times \mathcal{V}_j)$
$\boldsymbol{\mathcal{I}}_j^{CoM} = M_j^{CoM} * \boldsymbol{\mathcal{I}}_j$
$\mathcal{W}_{b,j} = \boldsymbol{\mathcal{I}}_j^{CoM} \left[\dot{\mathcal{V}}_j\right] - \mathcal{V}_j \times \boldsymbol{\mathcal{I}}_j^{CoM} [\mathcal{V}_j]$
end
$\mathcal{W}_{n+1} \leftarrow \mathcal{W}_{ext}$
for $j = n$ **to** 1 **do**
$\mathcal{W}_j = M_{j+1} \mathcal{W}_{j+1} \widetilde{M}_{j+1} + \mathcal{W}_{b,j}$
$\Omega_j = -\left(B_{j+1} \cdot \hat{\boldsymbol{\mathcal{I}}}_{j+1} [B_{j+1}]\right)^{-1}$
$\mathcal{W}_{B_{j+1}} = \Omega_j \hat{\boldsymbol{\mathcal{I}}}_{j+1} [B_{j+1}]$
$\hat{\tau}_j = \tau_j + (B_j \cdot \mathcal{W}_j)$
$\bar{\mathcal{W}}_j = \hat{\mathcal{W}}_{j+1} + \left(\hat{\mathcal{W}}_{j+1} \cdot B_{j+1} + \hat{\tau}_{j+1}\right) \mathcal{W}_{B_{j+1}}$
$\hat{\mathcal{W}}_j = M_{j+1} \bar{\mathcal{W}}_j \widetilde{M}_{j+1}$
$\hat{\ddot{q}}_j = \left(\hat{\tau}_j + B_j \cdot \hat{\mathcal{W}}_j\right) \Omega_j$
end
$\hat{\dot{\mathcal{V}}}_0 \leftarrow 0$
for $j = 1$ **to** n **do**
$\bar{\dot{\mathcal{V}}}_j = \widetilde{M}_j \hat{\dot{\mathcal{V}}}_{j-1} M_j$
$\hat{\dot{\mathcal{V}}}_j = \bar{\dot{\mathcal{V}}}_j + \left(\mathcal{W}_{B_j} \cdot \bar{\dot{\mathcal{V}}}_j + \hat{\ddot{q}}_j\right) B_j$
$\ddot{q}_j = \hat{\ddot{q}}_j + \left(M_j \mathcal{W}_{B_j} \widetilde{M}_j\right) \cdot \hat{\dot{\mathcal{V}}}_{j-1}$
end

3.6. Numerical Results

3.6.1. Inverse Kinematics

In order to evaluate the numerical inverse kinematics using the motor formulation, we repeated the following experiment 10000 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 3.3.3, 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 successul solves for these statistics. Note that this inverse kinematics solution presented in this work 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 [25] 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.

3.6.2. Numerical Validation of the Recursive Dynamics Algorithms

In order to be sure that our algorithms calculate the correct values for the robot dynamics, we implemented them for CGA using C++20 making them available as an open-source library. Since this library is heavily templated, especially on the numeric type, it is possible to use it in combination with *libtorch*. This means that all the mentioned computations can be used with *PyTorch* [10], making them parallelizable on a GPU. Thus, the recursive forward dynamics can be efficiently leveraged for sampling-based model predictive control and reinforcement learning.

Here, we use this implementation to compare the resulting values with the output of equivalent functions from existing libraries. We chose to compare to *Pinocchio* and *RBDL*, since both of them are using implementations of spatial vector algebra, as well as *KDL* which is a library within the ROS framework that uses an inversion of the mass matrix in order to compute the forward dynamics. The results of this numerical comparison can be seen in Table 3.1. The results show that our algorithm correctly computes the forward dynamics of the system and closely match the results based on spatial vector algebra, but the matrix inversion in *KDL* clearly introduces numerical imprecisions.

Table 3.1.: Numerical errors of the calculated acceleration from different forward dynamics solvers. The values are the norm difference of the resulting accelerations, averaged over 1000 computations. Each time we randomly sampled different values for the position, velocity and torque.

	Pinocchio	**RBDL**	**KDL**
Inverse Dynamics	1.28015e-14	1.27072e-14	0.668193
Forward Dynamics	6.73759e-14	7.06135e-14	125.061

3.7. Discussion

3.7.1. Computational Efficiency

Since the dynamics algorithms are heavily inspired by Featherstone's original formulation of the articulated body algorithm, the computational efficiency for the geometric algebra version is also $\mathcal{O}(n)$ in theory. However, since the implementations of our recursive forward dynamics and of geometric algebra for robotics in general are still in their infancy, they are not as highly optimized as other more established libraries that are using matrix algebra. Since we are currently prioritizing the theoretical developments, we have made no attempt at optimization. For this reason, our current implementation takes roughly one order of magnitude longer to compute the forward dynamics. The main reason for this is the allocation of additional memory due to unnecessary copy operations, mainly in obtaining fundamental parameters of the robotic system and the computation of the articulated body inertia. This is an issue that we will be adressing in the future, since it mostly requires some software engineering effort. In general, we expect the recursive forward dynamics in geometric algebra to compute the accelerations more efficiently than other formulations due to the uniform treatment of the involved twists and wrenches in the forward and backward passes. This expectation is backed by the findings about the computational efficiency of the forward kinematics [148]. Here, geometric algebra alleviates the need not only of having a dual adjoint matrix, but that of having an adjoint matrix altogether. This is because twists, wrenches and all other objects, such as the screw axes, can be uniformly transformed using the motors, which reduces the amount of computation and memory needed. This extends also to the inertia tensor, since with our formulation of its elements as wrench bivectors, they are part of the algebra and can thus also be transformed using motors directly.

3.8. Conclusion

In this chapter, we presented the formulation of the recursive forward dynamics and the spatial and articulated body inertia in CGA. Although the presented results in their current stage of research are of theoretical nature, we have shown their validity in simulation experiments. Additionally, the algorithms are implemented in an open-source library, making them ready to be used in practical applications. Future work on this library now includes addressing the problem of contact modeling in geometric algbra, since this would be the next step in order to provide full simulation capabilities with geometric algebra. However, we want to point out that by including the forward dynamics, geometric algebra, in principle, is now capable of replacing traditional vector/matrix based libraries for computing the kinematics and dynamics of serial kinematic chains. It, however, offers many additional tools for geometrically modeling various problems in robotics.

The formulation of CGA offers a unified view on screws, it allows for the coordinate invariant formulations of Lie groups and has computational and representational advantages. Thus, it not only unifies several frameworks into one algebra, but also, via the direct connection of the pin group as a double cover to the orthogonal group, it facilitates the geometric interpretation of conformal mappings compared to matrix algebra and makes them computationally more efficient.

The current research is limited by its implementation and the resulting lower computational performance compared to other libraries. Future research on using CGA can therefore have several directions. First, it should be explored how the different possible implementations of geometric algebra affect the performance of the algorithms and to improve the implementation in general. And second, the mathematical background of geometric algebra offers several directions for possible extensions and applications to other areas such as deformable objects and soft robotics. For this reason, the contributions should be seen from a mathematical perspective, opening doors to new potential research directions and uncovering mathematical and geometric connections that are hidden in other frameworks.

Part II.

Modeling of Optimization Problems for Manipulation Tasks

4. Geometric Algebra for Optimal Control

This chapter introduces the capabilities of geometric algebra when applied to robot manipulation tasks. In particular, it demonstrates how the modelling of cost functions for optimal control can be done uniformly across different geometric primitives. The resulting expressions have a low symbolic complexity and maintain a geometric intuitiveness. We demonstrate the usefulness, simplicity and computational efficiency of geometric algebra in several experiments using a Franka robot.

Publication Note

The material presented in this chapter is adapted from the following publication:

Tobias Löw and Sylvain Calinon. "Geometric Algebra for Optimal Control With Applications in Manipulation Tasks". In: *IEEE Transactions on Robotics* 39.5 (2023), pp. 3586–3600. DOI: 10.1109/TRO.2023.3277282

Website

Videos and supplemental material are available at:
`https://geometric-algebra.tobiloew.ch/optimal_control/`

4.1. Introduction

GA encodes geometric primitives, such as points, lines, planes, spheres, or quadric surfaces such as ellipsoids (depicted in red in Figure 4.1), 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 Figure 4.1). 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 Figure 4.1, it means that $u = f(x_0, x_d)$ can be described uniquely for the different geometric objects represented in the figure.

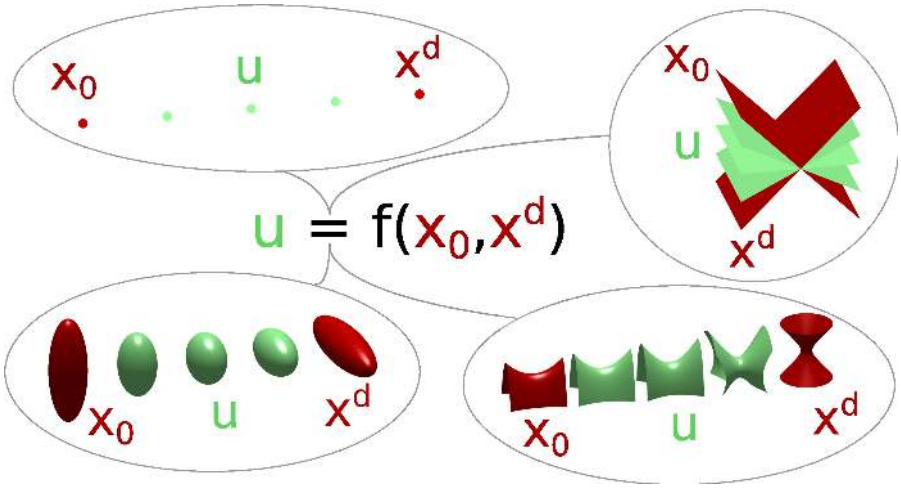


Figure 4.1.: A generic cost function in geometric algebra can consider different geometric primitives without changing its structure.

The key insight of this chapter is how the representation of geometric primitives within CGA can be used to uniformly model control objectives for robot manipulation tasks. Hence, we propose the usage of geometric algebra 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. We then demonstrate how geometric algebra formulations can be seamlessly used within existing frameworks based on linear matrix algebra by exploiting the sparsity of geometric algebra in order to facilitate its adoption.

This chapter is organized as follows: Section 4.3 introduces geometric algebra for optimal control and Section 4.4 then shows the experiments.

4.2. Background: 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. Modelling these mathematically requires special care, since they will determine the quality of the resulting solution. Furthermore optimal control can be applied as solver to a model predictive control problem, which requires fast convergence in order to achieve acceptable real time control rates. We will show that the use of geometric algebra for geometric primitives improves the clarity of equations and thus reduces computational difficulties. The modelling of the cost functions becomes easier and is done uniformly across all different primitives and is done directly in the error vector as opposed to the precision matrix, which results in a low symbolic complexity of expressions and a geometric intuitiveness, i.e. geometric meaning can directly be inferred.

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

$$\min_{\boldsymbol{u}} L(\boldsymbol{x}, \boldsymbol{u}) = l_f(\boldsymbol{x}_N) + \sum_{k=1}^{N-1} l_k(\boldsymbol{x}_k) + \|\boldsymbol{u}\|_R^2, \tag{4.1}$$

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) [207]. It solves the problem by linearizing the non-linear 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 a model predictive control (MPC) fashion in the experiments. This means that we are solving the regulation problem at each timestep and apply only the first control command to the robot.

4.3. Method

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

4.3.1. Optimal Control on the Motor Manifold

Employing homogeneous coordinates in 4D geometric algebra effectively allows us to linearize rigid body motions in 3D 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-dimensional 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

$$\boldsymbol{x}_{t+1} = \begin{bmatrix} \boldsymbol{b}_{t+1} \\ \dot{\boldsymbol{b}}_{t+1} \end{bmatrix} = \boldsymbol{A} \begin{bmatrix} \boldsymbol{b}_t \\ \dot{\boldsymbol{b}}_t \end{bmatrix} + \boldsymbol{C}\boldsymbol{u}_t, \tag{4.2}$$

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

Using the optimal control formulation that was presented in Equation (4.1), we now need to define appropriate state costs based on geometric algebra. Naturally the inverse kinematics cost function that was presented in Section 3.3.3 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.

$$M_t = \exp(B(\boldsymbol{b}_t)). \tag{4.3}$$

The corresponding cost function therefore becomes

$$l(\boldsymbol{x}_t) = \left\| \boldsymbol{e}(\boldsymbol{x}_t) \right\|_2^2 = \left\| \log\left(\widetilde{M}_{\text{target}} M_t \right) \right\|_2^2. \tag{4.4}$$

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-dimensional as well. Since the motors include orientation as well as position, it essentially is an oriented pointmass system.

Apart from defining target poses using the motor manifold, geometric algebra additionally offers the possibility to define targets using its geometric primitives and the accompanying incidence relationships. We exploit the nullspace 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

$$E(\boldsymbol{q}) = X_d \wedge M_t X \widetilde{M}_t, \tag{4.5}$$

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.

4.3.2. Optimal Control for Serial Manipulators

In this section, we are presenting the formulation of objective functions for optimal control problems with serial manipulators based on geometric algebra. 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 Equation (4.5) 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

$$\boldsymbol{\mathcal{J}}^E(\boldsymbol{q}) = X_d \wedge \left(\boldsymbol{\mathcal{J}}^A(\boldsymbol{q})X\widetilde{M}(\boldsymbol{q}) + M(\boldsymbol{q})X\widetilde{\boldsymbol{\mathcal{J}}^A}(\boldsymbol{q})\right), \tag{4.6}$$

where the $\boldsymbol{\mathcal{J}}^A(\boldsymbol{q})$ is the analytic Jacobian that was presented in Equation (3.11) and the reverse of a multivector matrix is defined as the element-wise 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 non-trivial blades it holds. It is, however, known a priori what the resulting non-trivial blades are. From this it follows that the embedding function $\mathcal{E}$ actually becomes dependent on X and X_d, i.e.

$$\boldsymbol{J}^E(\boldsymbol{q}) = \mathcal{E}(X, X_d)\left[\boldsymbol{\mathcal{J}}^E(\boldsymbol{q})\right]. \tag{4.7}$$

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 conformal geometric algebra allows for the equations to be coded exactly as they are presented in this chapter.

4.4. Experiments

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

4.4.1. Pointmass System

In Sections 3.3.3 and 4.3.1 we first presented the cost function to minimize the difference between two motors and then optimal control for an oriented pointmass 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 Figure 4.2.

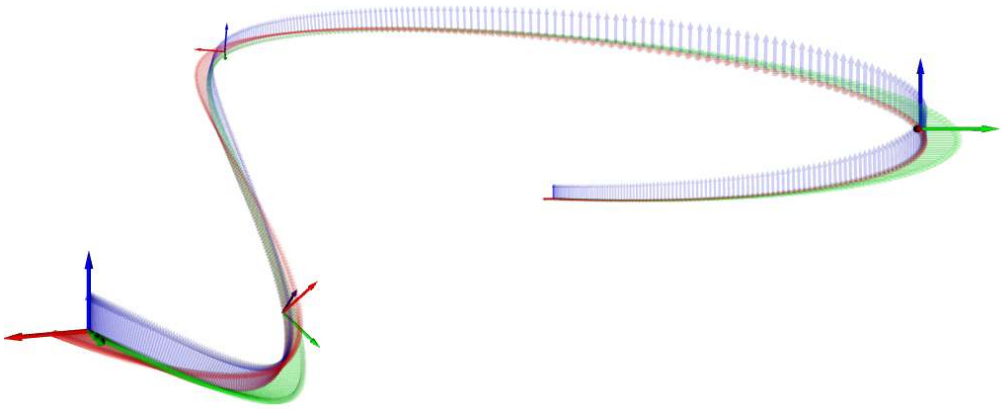


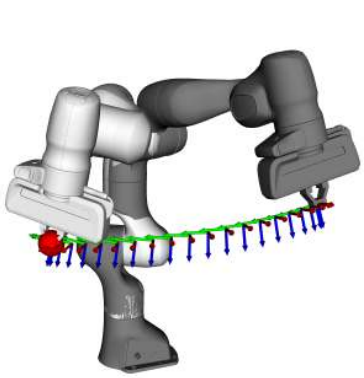
Figure 4.2.: Optimal trajectory for an oriented pointmass system. We placed four target motors along the trajectory at $T/4$, $T/2$, $3T/4$ and T, respectively. The target motors are highlighted along the trajectory. The optimal trajectory was then found using the system defined in Equation (4.2) and the cost function from Equation (3.21).

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.

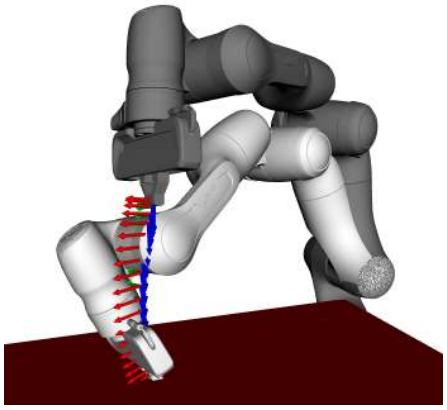
4.4.2. Reaching Tasks

Using the cost function formulation of geometric algebra that was presented in Equation (4.5) various reaching tasks can be defined. In general, for a reaching task, the end-effector should reach a certain position. This can be modeled by using a point for X. Then the desired multivector X_d can be any other geometric primitive, which in turn means that instead of only reaching a point, we can also reach lines, planes, circles and spheres. Higher order quadrics are possible as well, this however remains the subject of further investigations. We present optimal trajectories that were computed using the iterative linear quadratic regulator to explain how different geometric primitives can be reached using the same structure of the cost function, which is shown in Figure 4.3. In the experiments using the real Franka Emika we are then using 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 chapter are the modeling aspects. Since we are using an MPC

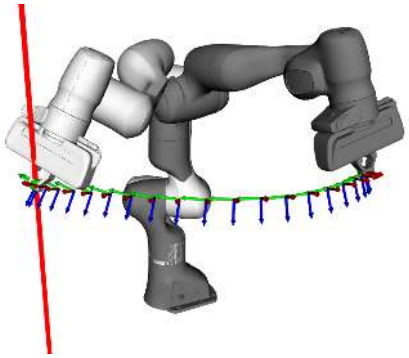
framework we do not need to use fixed points for the reaching but can have movable targets. In practice we use Aruco markers to track the target online or we are disturbing the robot while it is moving.



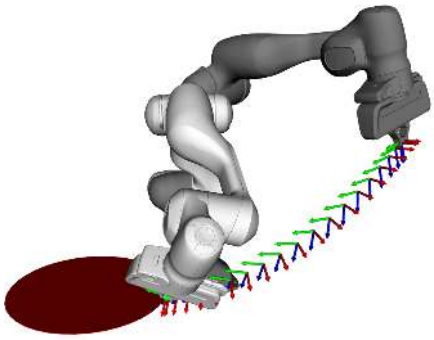
(a) Reaching a point.



(b) Reaching a plane.



(c) Reaching a line.



(d) Reaching a circle.

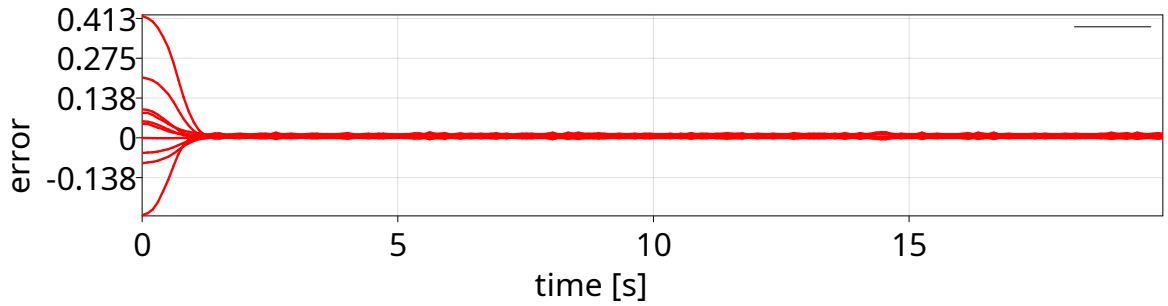
Figure 4.3.: Examples of optimal trajectories for reaching tasks using different geometric primitives. The initial configuration is always shown in gray and the final one in white. The target geometric primitive is shown in red. And the trajectory is depicted as the frames corresponding to the end-effector.

Reaching a Point

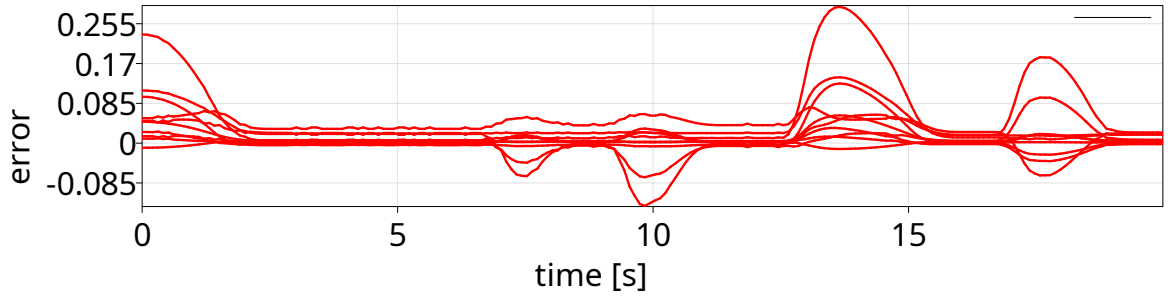
Reaching a point means that the desired geometric primitive is a point, i.e. $X_d = P$ in Equation (4.5). The reference primitive X_r is a point as well and represents the tip of the end-effector. Figure 4.3a shows an example trajectory for reaching a fixed point from a random initial configuration.

In the real robot experiments we used a single Aruco marker for reaching a movable point. For safety reasons the target point was set 10cm 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 Figure 4.4. In both plots that are presented we show the values of the 10-dimensional 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 nullspace 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. Figure 4.4a 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 Figure 4.4b. It can be seen that the MPC controller is fast and reactive, reaching the targets in a stable manner.



(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.

Figure 4.4.: Error of reaching a fixed target point in MPC for a total duration of 20s. The individual lines show the elements of resulting 10-dimensional error vector that results from the outer product of two points.

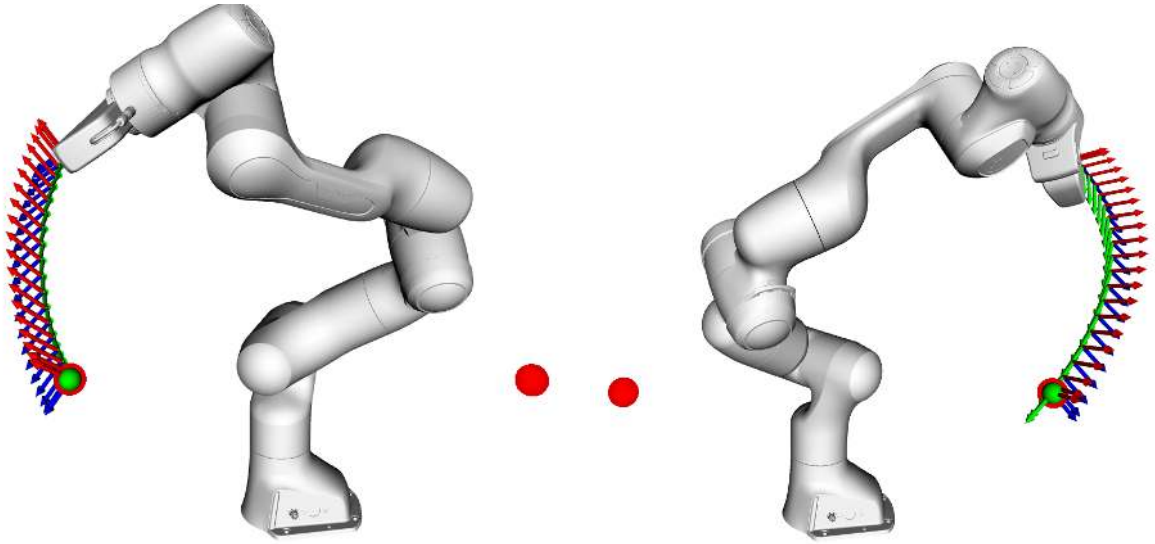
Reaching a point is in this context a trivial example, since it can be easily done using classical methods as well, but it serves to show that reaching problems can be solved for all geometric primitives in the same way as they are solved for a point using geometric algebra.

Reaching a Pointpair

Using a pointpair as the target presents a special opportunity to model a control problem with options. A pointpair is the result of the outer product of two points. From this outer product nullspace representation, it follows that the outer product of the 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 pointpair.

The two possibilities are shown in Figure 4.5, where Figure 4.5a shows the robot reaching the first point and Figure 4.5b the 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 Equation (4.6), 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 pointpair primitive due to its power to model a binary target. We also want to point out here that the pointpair primitive would be very suitable for modeling dual-arm manipulation tasks. In this scenario the pointpair would represent the end-effector positions of the two manipulators.



(a) Left (b) Right

Figure 4.5.: Depending on the initial configuration, either the left or the right point of the pointpair is reached.

Reaching a Plane

A plane in geometric algebra is represented by three individual points and $\boldsymbol{e}_\infty$ using the outer product nullspace, 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 (4.5) will therefore minimize the reaching motion to the plane from any random initial configuration as shown in Figure 4.3b.

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 Figure 4.3c. 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 10cm above it. The target line therefore always is perpendicular to the $y-z-$plane. In Figure 4.6 we show the results of the experiment. The corresponding error vector has 6 components and is geometrically equivalent to a circle.

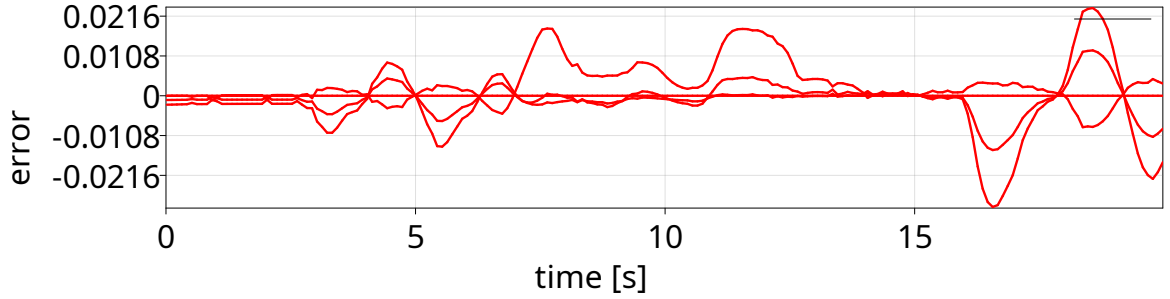


Figure 4.6.: Components of the error vector resulting from the outer product of a line and a point. Error of reaching a fixed target point in MPC for a total duration of 20s. The individual lines show the resulting error vector that results from the outer product of a line and a point.

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 Figure 4.3d). The target is here only the boundary of circle (not the full disc).

4.4.3. Pointing Task

The modelling of a pointing task only requires the usage of a line instead of a point for X in Equation (4.5). A possible scenario where this task would be applied is tracking an object with a robot arm endowed with a camera. The line can in this case be interpreted as the line of sight of the camera. Again different geometric primitives can be used as the target, since the intersection of a line with any other primitive can be calculated in closed form without exceptions. The setup is shown in Figure 4.7. Figure 4.8 shows the moving target.

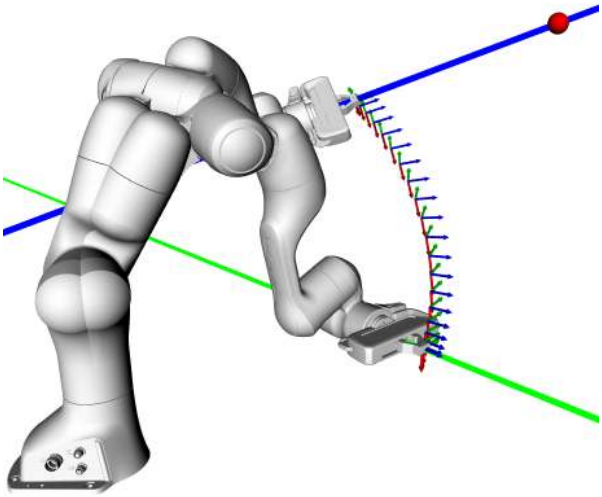


Figure 4.7.: optimal trajectory example for a pointing task. The target is shown as the red point. The pointing line is defined to be collinear to the z-axis of the end-effector frame. It is shown in green for the initial configuration and in blue for the final configuration.

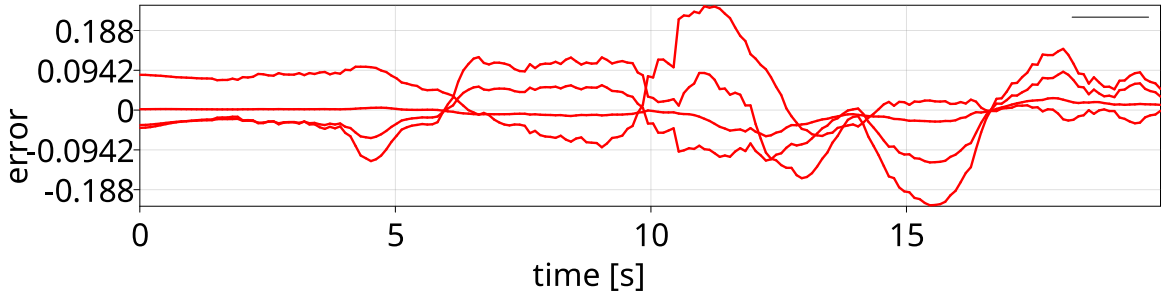


Figure 4.8.: Experimental results of the pointing task showing the components of the error vector during 20s. The Aruco marker representing the target point was constantly moved around, which is the reason why the error vector is more jittery than in other experiments.

4.4.4. Circular Object Grasping Task

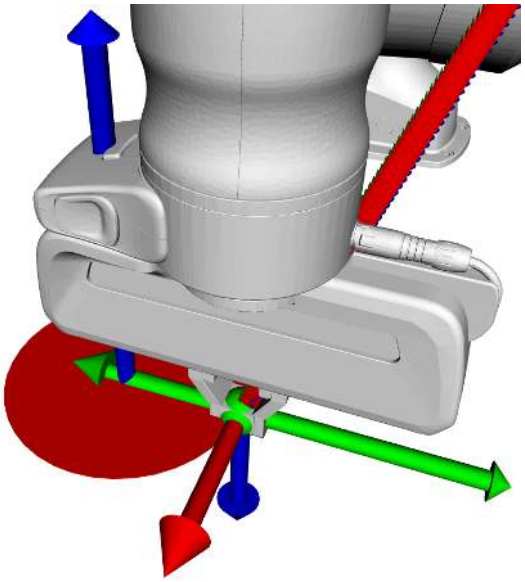
In this task the goal is to give an object with a round opening to the robot. The setup is depicted in Figure 4.9. The opening is modeled as a circle, i.e. the robot can grasp the object all around its opening. However, two additional constraints on the orientation are necessary to model this task. These constraints are shown in Figure 4.9a. The end-effector is required to be perpendicular to the plane that the circle lies in. A plane in geometric algebra can be obtained from a circle by a multiplying the circle with $\boldsymbol{e}_\infty$ and

the normal vector is then obtained as

$$\boldsymbol{n}_E = E^* - 0.5(E^* \cdot \boldsymbol{e}_0)\boldsymbol{e}_\infty. \tag{4.8}$$

The second constraint is that the direction that the gripper is actuated in needs to be perpendicular to the circle, which expressed 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

$$P' = (E \cdot P)E^{-1}. \tag{4.9}$$



(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, 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.

Figure 4.9.: Experiment setup of the circular object grasping task.

In Figure 4.9 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

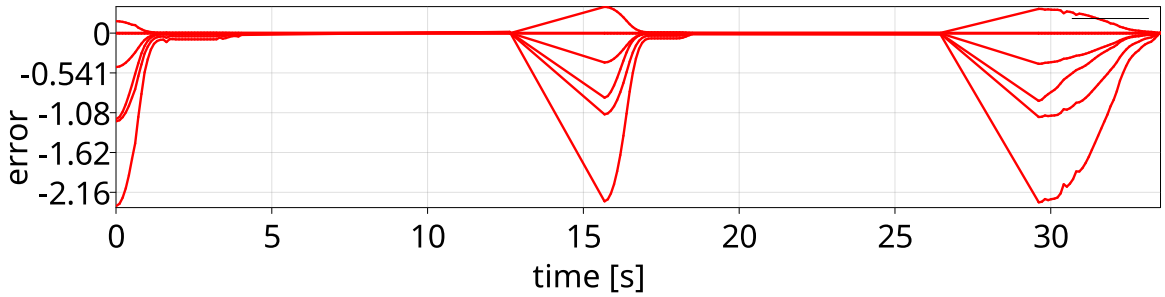


Figure 4.10.: Experimental results of the circular object grasping task. In this figure three repetitions are depicted, the location of the target box was changed in-between.

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 Figure 4.10.

4.5. Conclusion

We presented in this chapter the usage of geometric algebra for the modelling of optimal control tasks.

Higher order quadric surfaces such as cones and paraboloids in $\mathbb{G}_{6,3}$ [220] or ellipsoids and hyperboloids in $\mathbb{G}_{9,6}$ [37] 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 chapter, since the properties of the different geometric algebras such as the outer product nullspace, which we rely on, remain the same. It is therefore the topic of future work to investigate the integration of these algebras into our formulation. The benefit of this would be a more versatile and generic modeling of surfaces that can be exploited for various manipulation tasks.

A possible extension and application of this work would be grasping and in-hand manipulation. Using the geometric representations and the corresponding optimization functions presented in this we can actually very easily derive a model for grasping in geometric algebra. The three contact types 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.

5. Cooperative Dual-Task Space

In this chapter, we present an extension of the cooperative dual-task space (CDTS) in CGA. The CDTS was first defined using dual quaternion algebra and is a well established framework for the simplified definition of tasks using two manipulators. By integrating CGA, we aim to further enhance the geometric expressiveness and thus simplify the modeling of various tasks. We show this formulation by first presenting the CDTS and then its extension that is based around a cooperative pointpair. This extension keeps all the benefits of the formulation based on dual quaternions, but adds more tools for geometric modeling of the dual-arm tasks. We also present how this CGA-CDTS can be seamlessly integrated with the optimal control framework from the previous chapter. In the experiments, we demonstrate how to model different objectives and constraints using the CGA-CDTS. Using a setup of two Franka robots we then show the effectiveness of this approach using model predictive control in real world experiments.

Publication Note

The material presented in this chapter is adapted from the following publication:

> Tobias Löw and Sylvain Calinon. "Extending the Cooperative Dual-Task Space in Conformal Geometric Algebra". In: *IEEE International Conference on Robotics and Automation*. 2024

Website

Videos and supplemental material are available at:
`https://geometric-algebra.tobiloew.ch/cdts/`

5.1. Introduction

With the increasing desire to deploy robots in human environments, the need for robots to have human-like manipulation capabilities arises. One inherent ability that humans have is to manipulate objects using both their hands and arms, which is needed for example when objects that are either too large or too heavy need to be manipulated. In order to match these capabilities and to be able to mimic them, robotic systems also need to be able to cooperatively control two arms in order to perform tasks in human environments.

Apart from dual-arm systems being more human-like in terms of form factor, they also have some technical advantages. One can have the stiffness and strength of parallel manipulators combined with the flexibility and dexterity of serial manipulators [134]. Furthermore, since they increase the redundancy in the task-space due to their high number of degrees of freedom, they are better suited for intricate tasks that require a high manipulability such as screw assembly [142] and dishwashing [171]. Other applications of bimanual systems are manipulating articulated objects [8] or cables [224]. More advantages and examples are listed in [199].

Since many problems in robotics boil down to optimization problems that can be solved efficiently with various state-of-the-art solvers, it is of great interest to facilitate the modeling and increase the expressiveness of the formulations. Choosing the correct representation can make a huge difference in terms of how much prior knowledge we can embed into the formulation of those optimization problems. These are in robotics often very geometric, hence it is very beneficial to choose representations that intuitively allow to incorporate the geometry of the problem. Such cooperative structures for bimanual manipulation were proposed in [177]. Another formulation, the cooperative dual-task space (CDTS) [4], uses dual quaternion algebra (DQA), which not only unifies the treatment of position and orientation, it also offers a singularity-free representation and efficient computation [185]. Furthermore, it also allows the representation of various geometric primitives, which can then be used to simplify the modeling of the tasks [158].

Dual quaternions have a strong connection to geometric algebra, especially the variants known as projective (PGA) and conformal (CGA) geometric algebra [91], since dual quaternions are isomorphically embedded in their sub-algebras [92]. The geometric algebras, however, are richer algebras that offer more geometric primitives and, more importantly, they offer the geometric construction of primitives based on operations such as intersections [92]. This leads to new possibilities when formulating objectives and constraints based on the geometric primitives in the CGA-CDTS compared to the DQ-CDTS, which we will show in the experiments.

Based on the compact representation of the CDTS, various control strategies have been proposed. In [80], a coupled task-space admittance controller was presented, that allowed for a geometrically consistent stiffness term. A reactive control strategy was

developed in [129] that leveraged geometric primitives for task relaxations and priorities. Task priorities for control in the CDTS were also proposed in [63]. In order to exploit human demonstrations that allow the teaching of cooperative motions, motion primitives for bimanual systems were presented based on the CDTS [216].

Note that the results of the research that is based on the DQ-CDTS can also be used with the CGA-CDTS, albeit with mathematical changes due to the different algebra. Furthermore, the mentioned advantages of DQA also apply to CGA, since dual quaternions and the corresponding subalgebra in CGA, i.e. the motors, are isomorphic [22].

In this chapter, we formulate the CDTS in CGA and show how this formulation naturally extends the DQ-CDTS. The resulting CGA-CDTS retains the same properties of a compact representation of two arm system as the DQ-CDTS, while adding a useful geometric primitive, the cooperative pointpair, that represents both end-effector positions simultaneously. Furthermore, we demonstrate how the CGA-CDTS can be used in the optimal control formulation with geometric primitives for manipulators that we presented in Chapter 4. Hence, this chapter aims to explain the basic mathematical formulations of various optimization problems using the CGA-CDTS.

5.2. Method

The CDTS was proposed as a compact and singularity-free representation of a two-arm system [4]. It is defined using two poses that depend on the end-effector poses of the two manipulators, one is the relative and the other one is the absolute pose. All poses are represented using unit dual quaternions. Here, we present the reformulation of the CDTS in CGA, which uses motors instead of dual quaternions. We then show its extension by using the additional geometric primitives available in CGA. Lastly, we present how the CGA-CDTS can be used within the optimal control framework using geometric algebra that we presented in Chapter 4. We show the CGA-CDTS in Figure 5.1.

5.2.1. Conformal Geometric Algebra Cooperative Dual-Task Space

The geometric algebra equivalent of the relative and absolute dual quaternions of the DQ-CDTS are defined using motors. Given the joint configurations of the two manipulators, $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$, respectively, we can easily find their end-effector motors $M_1(\boldsymbol{q}_1)$ and $M_2(\boldsymbol{q}_2)$ using the forward kinematics in CGA. From this it is straightforward to formulate the relative motor as

$$M_r(\boldsymbol{q}_1, \boldsymbol{q}_2) = \widetilde{M}_2(\boldsymbol{q}_2) M_1(\boldsymbol{q}_1), \tag{5.1}$$

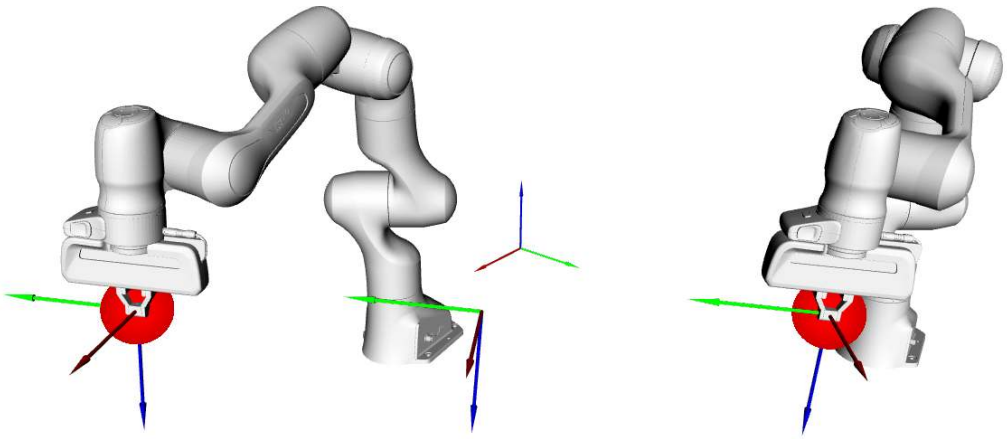


Figure 5.1.: Cooperative dual-task space using conformal geometric algebra. The figure shows the two manipulators as well as their individual, relative and absolute motors. Additionally, it shows the cooperative pointpair.

while its Jacobian, i.e. the relative analytic Jacobian, can be found as

$$\boldsymbol{\mathcal{J}}_r^A(\boldsymbol{q}_1, \boldsymbol{q}_2) = \begin{bmatrix} \widetilde{M}_2(\boldsymbol{q}_2)\boldsymbol{\mathcal{J}}_1^A(\boldsymbol{q}_1) & \widetilde{\boldsymbol{\mathcal{J}}}_2^A(\boldsymbol{q}_2)M_1(\boldsymbol{q}_1) \end{bmatrix}. \tag{5.2}$$

Similarly, the absolute motor can be found as

$$\begin{aligned} M_a(\boldsymbol{q}_1, \boldsymbol{q}_2) &= M_2(\boldsymbol{q}_2)M_{r/2}(\boldsymbol{q}_1, \boldsymbol{q}_2) \\ &= M_2(\boldsymbol{q}_2)\exp\left(\frac{1}{2}\log\left(M_r(\boldsymbol{q}_1, \boldsymbol{q}_2)\right)\right), \end{aligned} \tag{5.3}$$

with its corresponding absolute analytic Jacobian

$$\begin{aligned} \boldsymbol{\mathcal{J}}_a^A(\boldsymbol{q}_1, \boldsymbol{q}_2) =& M_2(\boldsymbol{q}_2)\boldsymbol{\mathcal{J}}_{M_{r/2}}^A(\boldsymbol{q}_1, \boldsymbol{q}_2) \\ &+ \begin{bmatrix} \boldsymbol{0} & \boldsymbol{\mathcal{J}}_2^A(\boldsymbol{q}_2)M_{r/2}(\boldsymbol{q}_1, \boldsymbol{q}_2) \end{bmatrix}, \end{aligned} \tag{5.4}$$

where $\boldsymbol{\mathcal{J}}_{M_{r/2}}^A(\boldsymbol{q}_1, \boldsymbol{q}_2)$ is the analytic Jacobian of the motor $M_{r/2}(\boldsymbol{q}_1, \boldsymbol{q}_2)$. It can be found as

$$\boldsymbol{\mathcal{J}}_{M_{r/2}}^A(\boldsymbol{q}_1, \boldsymbol{q}_2) = \boldsymbol{J}_{\mathbb{B}\to\mathcal{M}}(B_{r/2})\boldsymbol{J}_{\mathcal{M}\to\mathbb{B}}(M_r)\boldsymbol{J}_r^A(\boldsymbol{q}_1, \boldsymbol{q}_2). \tag{5.5}$$

Here, $B_{r/2}$ is the logarithm of the motor $M_{r/2}$. The matrices $\boldsymbol{J}_{\mathbb{B}\to\mathcal{M}}$ and $\boldsymbol{J}_{\mathcal{M}\to\mathbb{B}}$ are the Jacobians of the exponential and logarithmic mapping respectively. We already showed the derivation of the Jacobian of the logarithmic mapping in the appendix of [148]. The Jacobian of the exponential mapping can be found in Appendix A.

Both the relative and the absolute analytic Jacobians are $1 \times 2N$ multivector matrices that contain motors as their elements. Hence, when expanding it to normal matrix

algebra they become $8 \times 2N$ matrices.

Since motors in CGA can be used to transform any geometric primitive that is part of the algebra in a uniform way, it is easy to find cooperative geometric primitives. Their definition can be trivially found using Equation (2.17), where M is either the relative $M_r(\boldsymbol{q}_1, \boldsymbol{q}_2)$ or absolute $M_a(\boldsymbol{q}_1, \boldsymbol{q}_2)$ motor and X can be any geometric primitive. The corresponding Jacobians are then found using the respective Jacobians $\boldsymbol{\mathcal{J}}_r^A(\boldsymbol{q}_1, \boldsymbol{q}_2)$ and $\boldsymbol{\mathcal{J}}_a^A(\boldsymbol{q}_1, \boldsymbol{q}_2)$.

5.2.2. Cooperative Pointpair

In extension to the CDTS that was defined using dual quaternion algebra, the CDTS presented here using CGA also allows a geometric primitive that corresponds to both end-effector positions simultaneously. This cooperative pointpair is defined as the outer product of the two end-effector points, i.e.

$$P_{cdts} = M_1(\boldsymbol{q}_1)\boldsymbol{e}_0\widetilde{M}_1(\boldsymbol{q}_1) \wedge M_2(\boldsymbol{q}_2)\boldsymbol{e}_0\widetilde{M}_2(\boldsymbol{q}_2). \tag{5.6}$$

The Jacobian of the cooperative pointpair can be found as

$$\boldsymbol{\mathcal{J}}_{P_{cdts}} = \begin{bmatrix} \boldsymbol{\mathcal{J}}_{P_{cdts},1} & \boldsymbol{\mathcal{J}}_{P_{cdts},2} \end{bmatrix}, \tag{5.7}$$

where

$$\begin{aligned} \boldsymbol{\mathcal{J}}_{P_{cdts},1} =& \boldsymbol{\mathcal{J}}_1^A(\boldsymbol{q}_1)\boldsymbol{e}_0\widetilde{M}_1(\boldsymbol{q}_1) \wedge M_2(\boldsymbol{q}_2)\boldsymbol{e}_0\widetilde{M}_2(\boldsymbol{q}_2) \\ &+ M_1(\boldsymbol{q}_1)\boldsymbol{e}_0\widetilde{\boldsymbol{\mathcal{J}}}_1^A(\boldsymbol{q}_1) \wedge M_2(\boldsymbol{q}_2)\boldsymbol{e}_0\widetilde{M}_2(\boldsymbol{q}_2), \end{aligned} \tag{5.8}$$

and

$$\begin{aligned} \boldsymbol{\mathcal{J}}_{P_{cdts},2} =& M_1(\boldsymbol{q}_1)\boldsymbol{e}_0\widetilde{M}_1(\boldsymbol{q}_1) \wedge \boldsymbol{\mathcal{J}}_2^A(\boldsymbol{q}_2)\boldsymbol{e}_0\widetilde{M}_2(\boldsymbol{q}_2) \\ &+ M_1(\boldsymbol{q}_1)\boldsymbol{e}_0\widetilde{M}_1(\boldsymbol{q}_1) \wedge M_2(\boldsymbol{q}_2)\boldsymbol{e}_0\widetilde{\boldsymbol{\mathcal{J}}}_2^A(\boldsymbol{q}_2). \end{aligned} \tag{5.9}$$

Note that the cooperative pointpair is a direct representation of both points and is not the same as stacking the two points. Therefore the Jacobian matrix is also different, which will lead to different solutions of optimization problems. An example of this is shown in Figure 5.2, where two Franka Emika robots are tasked to reach a plane, once individually (i.e. by stacking their end-effector points) and once cooperatively (i.e. by using the cooperative pointpair that is presented here). It can be seen that the corresponding solution configurations are not the same, which shows that the cooperative pointpair representation lets the two robots influence each other.

Evidently, this cooperative pointpair Jacobian has a singularity in the case when both end-effector positions are equal. This will cause the outer product, by definition, to be

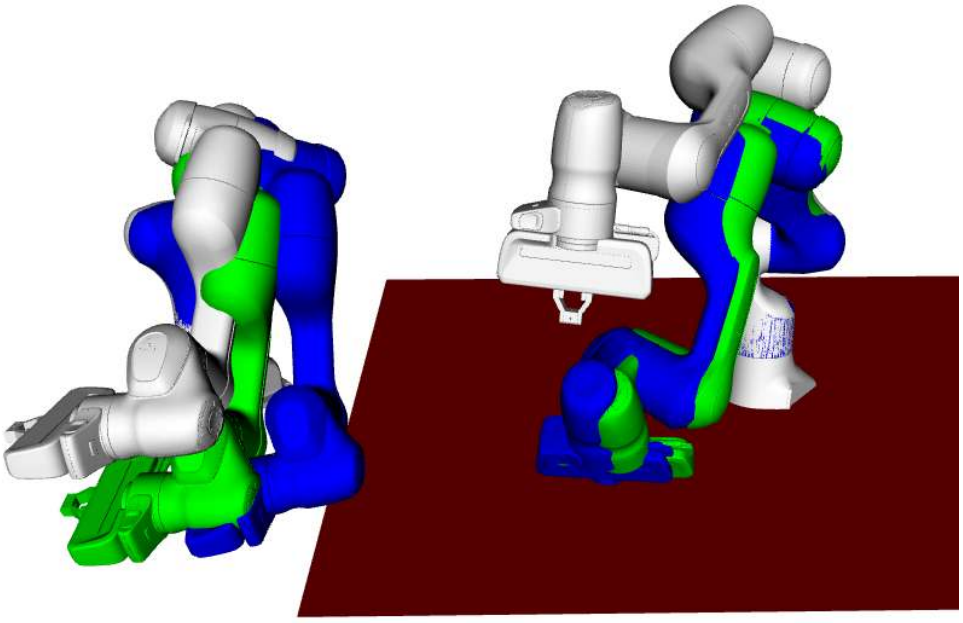


Figure 5.2.: Dual-Arm manipulator reaching a plane. The target plane is shown in red. The white Franka Emika robots show the initial configurations, the green ones are the result of individually reaching the plane, and the blue ones cooperatively.

zero. In practice, this can be avoided by posing a constraint on the distance between end-effectors, which usually is required anyways.

5.2.3. Using the CGA-CDTS for Optimal Control

Integrating the CGA-CDTS into the optimal control framework that we presented in [148] can be achieved by replacing the single end-effector motor with the relative and absolute motors, respectively. The objective of reaching a target motor is then formulated as

$$E_M(\boldsymbol{q}_1, \boldsymbol{q}_2) = \log\left(\widetilde{M}_{target} M(\boldsymbol{q}_1, \boldsymbol{q}_2)\right), \tag{5.10}$$

where $M(\boldsymbol{q}_1, \boldsymbol{q}_2)$ can be either $M_r(\boldsymbol{q}_1, \boldsymbol{q}_2)$ or $M_a(\boldsymbol{q}_1, \boldsymbol{q}_2)$.

Alternatively, we can define a residual multivector for reaching a geometric primitive X_d as

$$E_{X_d}(\boldsymbol{q}_1, \boldsymbol{q}_2) = X_d \wedge M(\boldsymbol{q}_1, \boldsymbol{q}_2) X \widetilde{M}(\boldsymbol{q}_1, \boldsymbol{q}_2), \tag{5.11}$$

again using either the relative or absolute motor. Deriving the respective Jacobians is straightforward using the definitions of the relative and absolute analytic Jacobians in Equations (5.2) and (5.4). With this, these residual multivectors of the CGA-CDTS can

then directly be used to define objectives or constraints for optimal control problems, which would mean for example that a relative or absolute geometric primitive should be reached.

The cooperative pointpair can also be used in order to define tasks as optimization problems. The first way to do so is using the outer product in a similar way to the above relative and absolute residual multivectors, i.e.

$$E_{cdts}(\boldsymbol{q}_1, \boldsymbol{q}_2) = P_{target} \wedge P_{cdts}(\boldsymbol{q}_1, \boldsymbol{q}_2), \tag{5.12}$$

with the Jacobian

$$\boldsymbol{\mathcal{J}}_{E_{cdts}}(\boldsymbol{q}_1, \boldsymbol{q}_2) = P_{target} \wedge \boldsymbol{\mathcal{J}}_{P_{cdts}}(\boldsymbol{q}_1, \boldsymbol{q}_2), \tag{5.13}$$

This objective means that the two manipulators should cooperatively reach a single point. The implications and results of this objective are further detailed in the experiment section.

Another common use-case are containment relationships for the cooperative pointpair with respect to other geometric primitives. These can then be used to define tasks where the dual-arm system should cooperatively reach a target. The mathematical formulation is using a product called the commutator product $\times$, i.e.

$$\begin{aligned} E_{cdts}(\boldsymbol{q}_1, \boldsymbol{q}_2) &= X_d \times P_{cdts}(\boldsymbol{q}_1, \boldsymbol{q}_2) \\ &= \frac{1}{2}(X_d P_{cdts}(\boldsymbol{q}_1, \boldsymbol{q}_2) - P_{cdts}(\boldsymbol{q}_1, \boldsymbol{q}_2) X_d). \end{aligned} \tag{5.14}$$

The Jacobian of this containment relationship can be found as

$$\boldsymbol{\mathcal{J}}_{E_{cdts}}(\boldsymbol{q}_1, \boldsymbol{q}_2) = X_d \times \boldsymbol{\mathcal{J}}_{E_{cdts}}(\boldsymbol{q}_1, \boldsymbol{q}_2). \tag{5.15}$$

The interpretation of the residual multivector $E_{cdts}(\boldsymbol{q}_1, \boldsymbol{q}_2)$ is that it should be reduced to zero if the cooperative pointpair is contained within the desired geometric primitive X_d, e.g. if both end-effector points lie on a circle.

The distance between the two end-effector can be constrained using the inner product between the two end-effector points, i.e.

$$E_d(\boldsymbol{q}_1, \boldsymbol{q}_2) = -2M_1(\boldsymbol{q}_1)\boldsymbol{e}_0\widetilde{M}_1(\boldsymbol{q}_1) \cdot M_2(\boldsymbol{q}_2)\boldsymbol{e}_0\widetilde{M}_2(\boldsymbol{q}_2) - d^2, \tag{5.16}$$

where d is the desired distance.

5.3. Experiments

In this section, we are presenting various cooperative tasks that are defined in the CGA-CDTS. For each task we are providing the mathematical definition of the optimization

problem. We are then solving those optimization problems using standard solvers such as Gauss-Newton. For simplicity, the problems in simulation are formulated essentially as inverse kinematics problems, the same objectives and constraints can, however, be used in optimal control problems that are for example then used for model predictive control. We are demonstrating this in the real world experiments, where the problems are then solved by using a variant of the iterative linear quadratic regulator (iLQR) [207]. Both the simulation and the real-world experiments use the same setup of two table-top mounted Franka Emika robots. Additional material for the experiments as well as the videos of the real world experiments can be found on the accompanying website [1]. All geometric algebra computations are done using our open-source library *gafro*[2].

5.3.1. Cooperatively Reaching a Point

As mentioned before, the cooperative pointpair models both end-effector positions simultaneously. Hence, when formulating a simple optimization problem such as reaching a single point, the system automatically chooses which manipulator should perform the task. Since the information of both manipulators is encoded in one geometric primitive, it alleviates the need to construct the problem using conditional formulations. The problem can be expressed compactly using the outer product, i.e.

$$\boldsymbol{q}^* = \min_{\boldsymbol{q}} \left\| P_{target} \wedge P_{cdts}(\boldsymbol{q}_1, \boldsymbol{q}_2) \right\|_2^2. \tag{5.17}$$

An example of this task is shown in Figure 5.3. Notice how in both cases the manipulator that is closer to the point performs the reaching task while the other remains in its initial configuration.

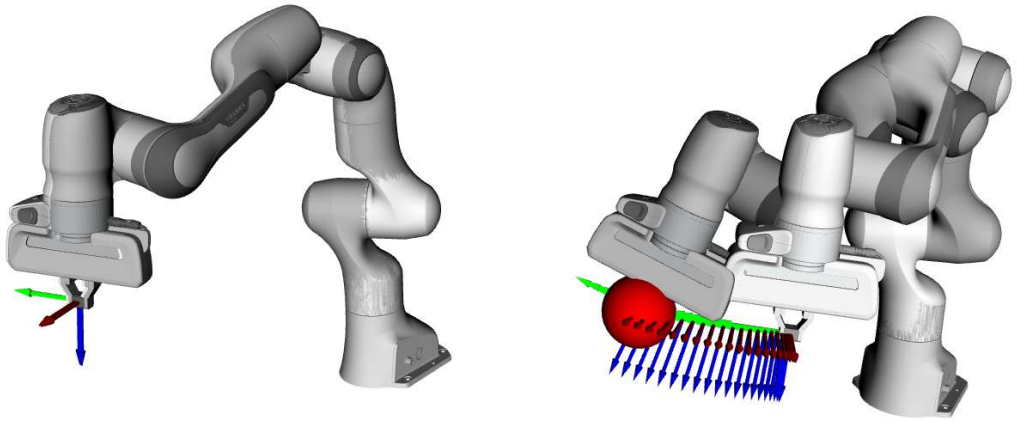
We also show this experiment in using the real-world setup and the accompanying video can be found on our website[3].

5.3.2. Cooperatively Reaching a Circle

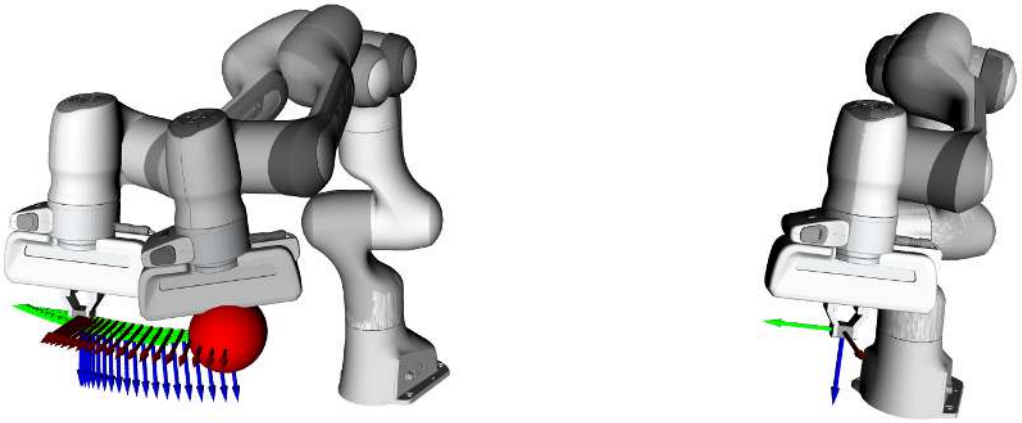
One of the geometric primitives that is available in CGA but not DQA is a circle. Hence, we can use the CGA-CDTS to define a task where a dual arm system should cooperatively reach a circle. An example application of reaching a circle would be holding a filled bucket with two manipulators. Mathematically, a circle is obtained by the outer product of three points.

The problem of cooperatively reaching a circle is formulated as a constrained opti-

[1] https://geometric-algebra.tobiloew.ch/cdts/
[2] https://gitlab.com/gafro/gafro
[3] https://geometric-algebra.tobiloew.ch/cdts/



(a) Left arm reaching for the point.



(b) Right arm reaching for the point.

Figure 5.3.: Two Franka Emika robots reaching for a single point in the cooperative dual-task space. The initial configurations are shown in white and the final ones in gray. The target point is shown in red.

mization problem using the cooperative pointpair

$$\begin{aligned} \min_{\boldsymbol{q}} & \left\| \log\left(\widetilde{M}_r(\boldsymbol{q}_0) M_r(\boldsymbol{q})\right) \right\|_2^2 \\ \text{s.t. } & C \times P_{cdts}(\boldsymbol{q}_1, \boldsymbol{q}_2) = 0. \end{aligned} \tag{5.18}$$

This optimization problem formulates the task of reaching a circle, while trying to maintain the initial relative motor. The result of this problem can be seen in Figure 5.4. Formulating the task in this manner circumvents the mentioned singularity issue in the Jacobian of the cooperative pointpair, since the system will reach circle while keeping as close as possible to their initial relative poses.

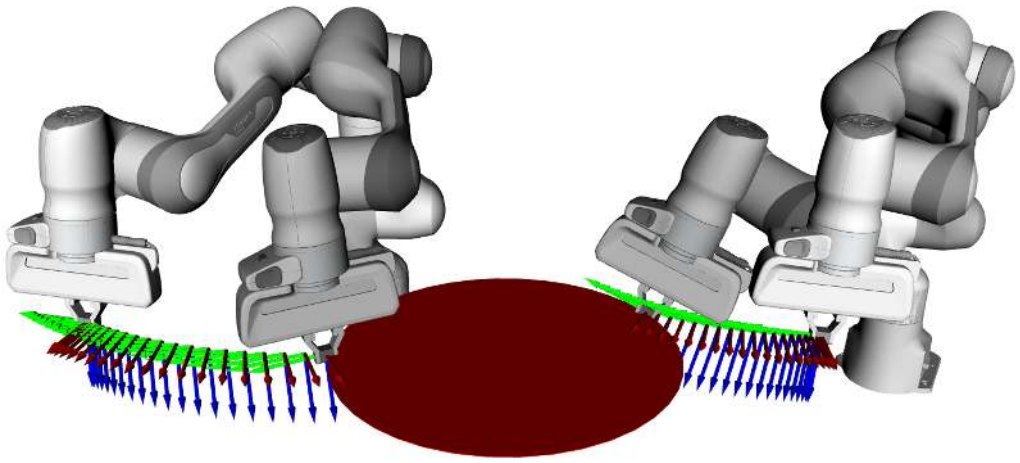


Figure 5.4.: Two Franka Emika robots reaching a circle in the cooperative dual-task space while trying to maintain the same relative motor.

5.3.3. Aligning Orientation Axis

When two manipulators are cooperatively manipulating an object, often it is necessary to partially constrain their relative orientation, which is necessary in nearly all dual-arm grasping scenarios of rigid objects. One way to do so is enforcing collinear lines in the desired direction at the end-effector level. This is again achieved by using the commutator product, i.e.

$$\min_{\boldsymbol{q}} \left\| M_1(\boldsymbol{q}_1) L_1 \widetilde{M}_1(\boldsymbol{q}_1) \times M_2(\boldsymbol{q}_2) L_2 \widetilde{M}_2(\boldsymbol{q}_2) \right\|. \tag{5.19}$$

In Figure 5.5, we show the results of minimizing Equation (5.19), where $L_1 = L_2 = \boldsymbol{e}_0 \wedge (\boldsymbol{e}_0 + \boldsymbol{e}_1 + \frac{1}{2}\boldsymbol{e}_\infty) \wedge \boldsymbol{e}_\infty$, which corresponds to aligning two lines that pass through the x-axes of the frames at the end-effectors of the two manipulators.

There are, of course, other ways to achieve this behavior, e.g. by constraining the orientation part of the relative motor. We chose to demonstrate this method of aligning orientation, however, because it provides a lot of flexibility, since the two lines L_1 and L_2 can be chosen arbitrarily.

A similar instance of this problem can be defined using the absolute motor of the CGA-CDTS. By using the absolute translator $T_a(\boldsymbol{q}_1, \boldsymbol{q}_2)$ we can move a desired line to the absolute position and require it to be identical to the line moved by the absolute motor. This defines a desired orientation with respect to the arbitrary axis that is defined by the line. The formulation hence is

$$\min_{\boldsymbol{q}} \left\| T_a(\boldsymbol{q}_1, \boldsymbol{q}_2) L \widetilde{T}_a(\boldsymbol{q}_1, \boldsymbol{q}_2) \times M_a(\boldsymbol{q}_1, \boldsymbol{q}_2) L \widetilde{M}_a(\boldsymbol{q}_1, \boldsymbol{q}_2) \right\|. \tag{5.20}$$

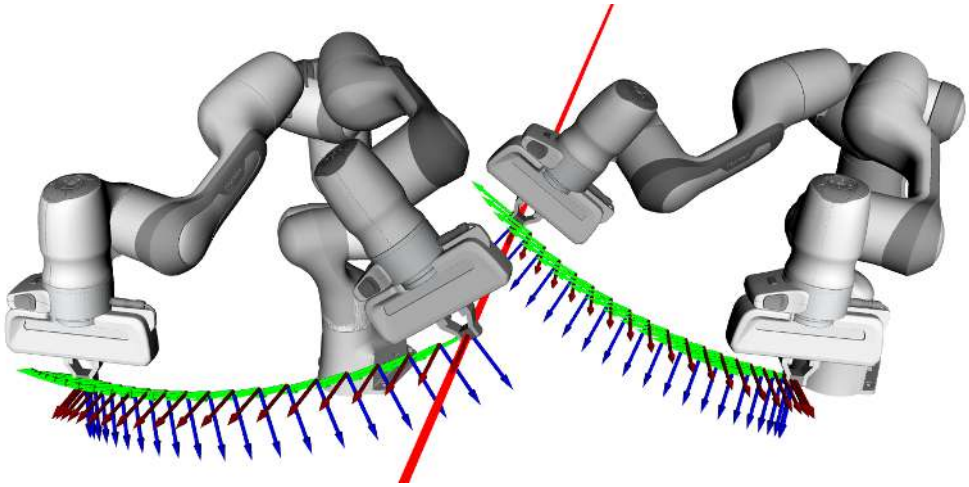


Figure 5.5.: Aligning the x-Axis.

5.3.4. Balancing a Plate

In this real-world experiment we are combining several of the previous constraints in order to implement the task of balancing a plate. First, the robot lifts the plate to a height of 20cm above the table, then it will try to keep it in that position. This is formulated as constrained optimization problem, where the objective is to stay close to the initial configuration and the constraint is to keep the z-axis of absolute motor perpendicular to the xy-plane.

We formulate this task as an optimal control problem

$$\begin{aligned} \boldsymbol{u}^* = &\min_{\boldsymbol{u}} \left\| E_N(\boldsymbol{x}_N) \right\|_2^2 + \sum_{k=0}^{N-1} \left\| E_k(\boldsymbol{x}_k) \right\|_2^2 + \|\boldsymbol{u}_k\|_R^2 \\ &\text{s.t. } \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k) \end{aligned} \tag{5.21}$$

where $\boldsymbol{x} = \left[\boldsymbol{q}_1^\top, \boldsymbol{q}_2^\top, \dot{\boldsymbol{q}}_1^\top, \dot{\boldsymbol{q}}_2^\top\right]^\top$ and $\boldsymbol{u} = \left[\ddot{\boldsymbol{q}}_1^\top, \ddot{\boldsymbol{q}}_2^\top\right]^\top$. As the state dependent cost we choose the residual multivectors of Equations (5.19) and (5.20), where the line L is chosen to be the z-axis, i.e. $L = \boldsymbol{e}_0 \wedge (\boldsymbol{e}_0 + \boldsymbol{e}_3 + \frac{1}{2}\boldsymbol{e}_\infty) \wedge \boldsymbol{e}_\infty$. This means that the two manipulators should simultaneously keep their relative grasping positions on the plate and to keep the plate horizontal.

This formulation is given to a model predictive controller that uses second order system dynamics to compute desired accelerations. Using inverse dynamics we then compute torque commands for the control of the two manipulators. We chose a short horizon of 10 timesteps with $\Delta t = 0.01$, in order to achieve a very reactive behavior of the controller. The model predictive controller is then run at 100Hz and the inverse dynamics controller at 1000Hz.

Since there are only relative constraints in this task, the robots have a compliant

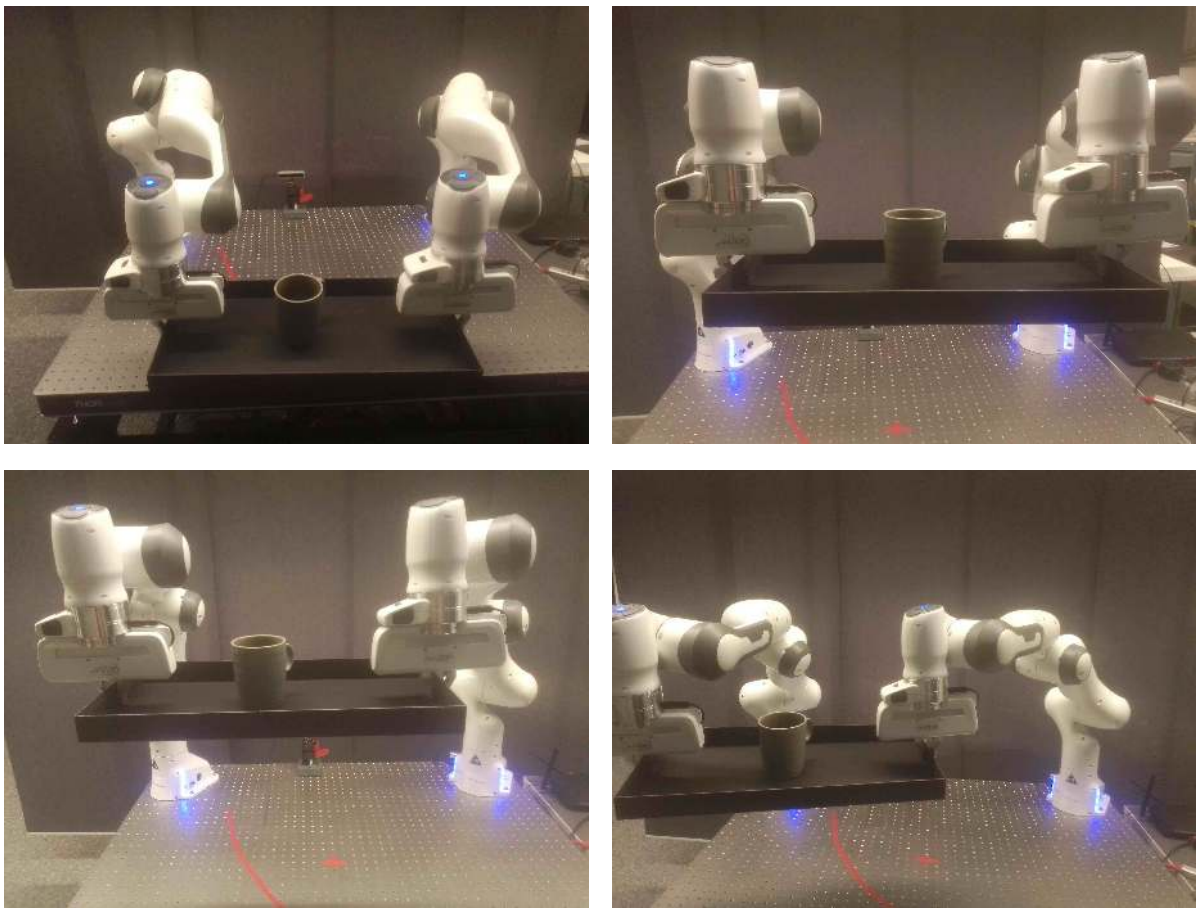


Figure 5.6.: Results of balancing a plate in different configurations.

behavior when one of them or the plate is moved by hand. The other one then adapts its configuration accordingly. We show several different configurations in Figure 5.6. If no external forces are applied, the robots stay in their current configuration. This experiment can also be seen in the accompanying video on our website.

5.4. Conclusion

In this chapter we presented an extension of the cooperative dual-task space in conformal geometric algebra, namely the CGA-CDTS. This extension keeps all the benefits of the original formulation that is based on dual quaternions, but adds more tools for geometric modeling of the dual-arm tasks. After reformulating the CDTS, we showed how the cooperative pointpair can be used to simultaneously represent both end-effector positions and how that can be exploited for cooperative reaching tasks and how the additional geometric primitives facilitate the modeling of dual-arm tasks. We then demonstrated the integration of the CGA-CDTS into an existing framework for optimal control using

geometric algebra.

6. Cooperative Manipulation Control

Many tasks in human environments require collaborative behavior between multiple kinematic chains, either to provide additional support for carrying big and bulky objects or to enable the dexterity that is required for in-hand manipulation. Since these complex systems often have a very high number of degrees of freedom coordinating their movements is notoriously difficult to model. In this chapter, we present the derivation of the theoretical foundations for cooperative task spaces of multi-arm robotic systems based on geometric primitives defined using conformal geometric algebra. Based on the similarity transformations of these cooperative geometric primitives, we derive an abstraction of complex robotic systems that enables representing these systems in a way that directly corresponds to single-arm systems. By deriving the associated analytic and geometric Jacobian matrices, we then show the straightforward integration of our approach into classical control techniques rooted in operational space control. We demonstrate this using bimanual manipulators, humanoids and multi-fingered hands in optimal control experiments for reaching desired geometric primitives and in teleoperation experiments using differential kinematics control. We then discuss how the geometric primitives naturally embed nullspace structures into the controllers that can be exploited for introducing secondary control objectives. This work, represents the theoretical foundations of this cooperative manipulation control framework, and thus the experiments are presented in an abstract way, while giving pointers towards potential future applications.

Publication Note

The material presented in this chapter is currently under review:

Tobias Löw, Cem Bilaloglu, and Sylvain Calinon. "Cooperative Geometric Primitives for Multi-Arm Manipulation Control". In: *under review* (2025)

Website

Videos and supplemental material are available at:
`https://geometric-algebra.tobiloew.ch/cooperative_/geometric_primitives`

6.1. Introduction

Humans excel at dexterous manipulation using the redundancy of their arms and hands to achieve complex tasks where cooperative behaviors between the different kinematic chains is required. Many environments are built around these advanced manipulation capabilities, and robots that are increasingly operating within them. Yet, it still remains challenging for robots to achieve dexterous manipulation that is on par with humans [116]. Nevertheless, robots are now expected to handle various tasks that require them to interact with objects in a way that goes beyond traditional parallel jaw grippers or suction cups that are often featured on single arm manipulators. These types of end-effectors are limiting the robot manipulation capabilities, since many objects are often more complex and require different strategies. For instance, big and bulky objects may call for dual-arm manipulators for picking and placing. This kind of coordination between two arms is vital when grasping, lifting, and transferring objects of varying shape, weight, and fragility. This is similarly true for intricate tasks such as in-hand manipulating and multi-fingered grasping using robotics hands. The common ground here is coordinating the movements of two or more parallel kinematic chains to interact with the environment during manipulation tasks.

However, many manipulation tasks do not require exploiting the full redundancy of the system. This is because the kinematic chains are not acting independently but instead are coordinated to achieve a shared objective. As a result, the effective behavior of the system often lies on a lower-dimensional manifold embedded within the full configuration space. We refer to these manifolds exhibiting specific geometric structures as cooperative task spaces. Recognizing and leveraging this structure not only reduces the dimensionality of the control or planning problems but also improves interpretability. Moreover, these geometric structures naturally give rise to geometric nullspaces, which can be exploited for secondary objectives, such as regulating contact forces, without interfering with the primary task objective, as we previously demonstrated in [31].

Existing work using dual quaternions [4] and geometric algebra [147] has begun to explore cooperative task spaces, particularly in the context of dual-arm manipulation. However, the notion of cooperation extends well beyond dual-arm settings. Many real-world scenarios involving multiple kinematic chains, such as torso-arm-hand coordination or multi-fingered in-hand manipulation, can benefit from a unified geometric framework that generalizes cooperative control. To illustrate this, we provide a non-exhaustive collection of robotic systems and manipulation challenges that can be abstracted through cooperative subspaces corresponding to geometric primitives, as shown in Figures 6.1-6.5. The geometric primitives only depend on the number of kinematic chains that are involved in the task. Hence, they model the cooperative behavior, and are not chosen based on an approximation of the object. Our overarching goal is to generalize operational space control [123], which was originally formulated for single manipulators,

to systems with multiple kinematic chains by representing cooperative task spaces as geometric primitives within the control architecture.

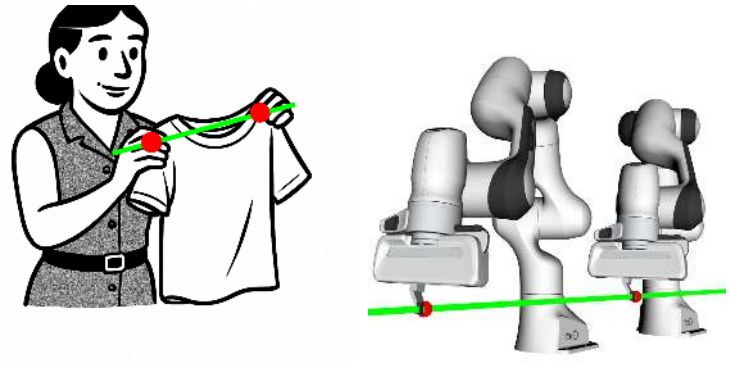


Figure 6.1.: Clothes folding task represented by a cooperative line using two reference points.

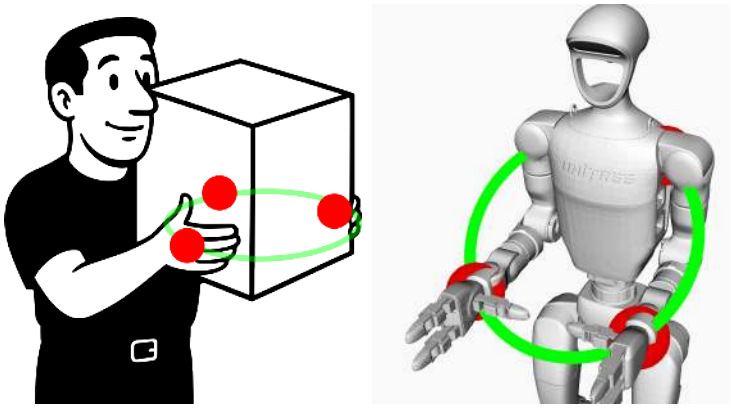


Figure 6.2.: Task of carrying a big box represented by a cooperative circle using three reference points.

In this chapter, we address the challenge of cooperative manipulation control for robotic systems involving multiple parallel kinematic chains. We do so by combining the various ideas around cooperative control, grasping nullspaces and geometric primitives, into a single, mathematically coherent framework by introducing cooperative geometric primitives and integrating them into geometrically consistent control schemes. For this, we leverage conformal geometric algebra (CGA) to find algebraic objects that represent geometric primitives that go beyond keypoints. In prior work, we have shown how these primitives define geometric nullspaces and how they can be used in optimal control [148]. In the same optimal control framework, we have then extended the cooperative dual-task space using CGA for bimanual robots [147]. Here, we further extend this idea, by exploring how cooperative geometric primitives enable decentralized control strategies that harmonize task execution, disturbance rejection, and inter-agent coordination. We show

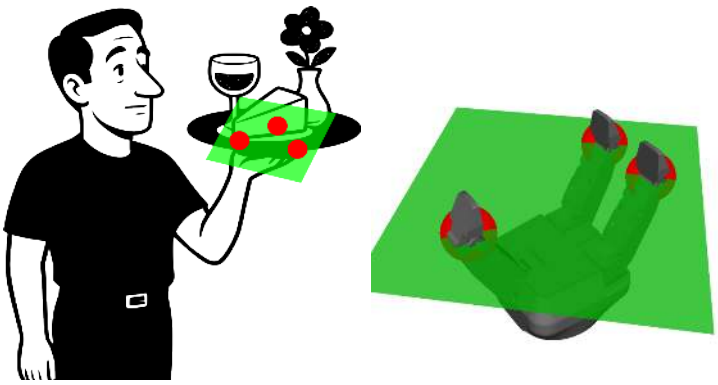


Figure 6.3.: Task of carrying a plate represented by a cooperative plane using three reference points.

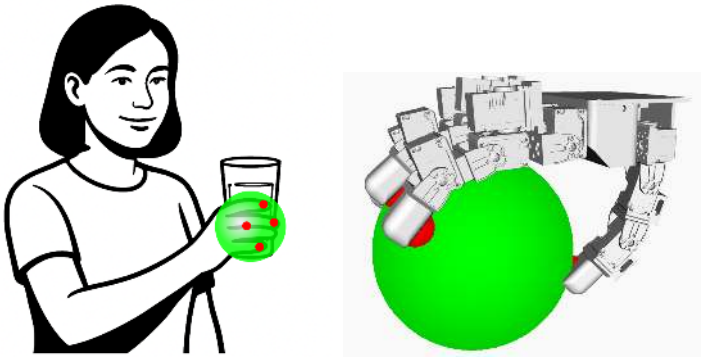


Figure 6.4.: Task of grasping a glass represented by a cooperative sphere using four reference points.

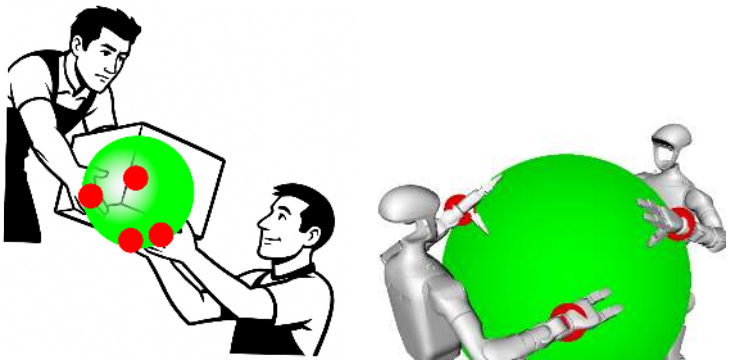


Figure 6.5.: Task of handing over big box represented by a cooperative sphere using four reference points.

that by exploiting CGA, different systems can be modelled in a uniform manner that allows informed collaboration strategies based on the geometric primitives and conformal transformations that can be expressed in this algebra. Our contributions therefore are

A geometric representation for cooperative behaviors of multiple parallel kinematic

chains using geometric primitives, enabling intuitive modeling of coordinated manipulation tasks;
Introduction of similarity transformations of geometric primitives, extending classical rigid body transformations from single end-effector control to multi-chain cooperative manipulation;
A unified mathematical framework for cooperative control based on similarity transformations, that can be used with optimal control or differential kinematics formulations;
Exploitation of geometric nullspaces inherent in the cooperative task space for achieving secondary control objectives, such as contact force regulation or redundancy resolution, without affecting primary task execution;
Demonstration of the versatility and scalability of the proposed approach on complex robotic systems, including humanoid platforms and multi-fingered hands, through tasks involving cooperative reaching and teleoperation.

6.1.1. Related Work

A common example for highlighting the challenges of cooperative manipulation is the collective transport of big and bulky objects that cannot be handled by a single robot. This task is very challenging, since multiple robots interact with the same object via local control actions [71], which requires them to cooperate in order to achieve a common, global goal. Traditional approaches often use a leader-follower strategy, where the different roles are predefined, and the agents communicate explicitly. However, these strategies often fail due to uncertainties stemming from the task parameters, environmental dynamics, or the team composition [72]. Solving these issues requires strategies that do not rely on centralized coordination, but instead use the manipulated object as a physical channel for implicit coordination. By sensing the exerted forces on the shared payload, the robots can decompose the motion into an allowed task space motion and a nullspace motion that is used rejecting perturbations [48]. This strategy minimizes force conflicts, and enables cooperative manipulation even under strict communication constraints. Recent advances in cooperative control favor frameworks that further emphasize distributed optimization and adaptability. They optimize the individual performance of the robots in the system that suffers reduced operation capabilities due to the physical connection through grasping [96]. Furthermore, the optimization can include objectives for enhancing the manipulability, avoiding obstacles, and reducing internal forces caused by robot motion errors due to coupling [97]. Hence, there is an inherent trade-off between physical coupling (e.g. rigid grasps) and operational flexibility, which makes it important to deal with disturbances to avoid unexpected behaviors [7]. Learning-based approaches try to simplify the problem of coordinating multiple agents by introducing actions through shared latent spaces, which reduces the sample complexity of high-dimensional tasks like

dual-arm manipulation [**aljaloutCLASCoordinatingMultiRobot**]. Alternatively, bi-manual end-effector poses are learned from demonstrations using task-parameterized dynamical systems [198].

Some approaches are inspired by the formation control that is often used in the area of autonomous ground vehicles [203], and are extended to manipulation, where the agents are treated as part of a cohesive geometric formation [196]. In this setting, one of the agents can be a human that is guiding the robots, which in turn try to maintain desired geometry w.r.t. each other, while reducing internal stresses [197]. It can then be shown that, the internal interactions forces in cooperative manipulation that arise from a formation of robots, lie in the null space of the grasp matrix, i.e. the dynamics of the object are not affected by them [64]. This insight also bridges the connection to literature on grasping and in-hand manipulation, where the problem setting is similar, in the sense that multiple parallel kinematic chains are tasks to cooperatively manipulate a shared object. Both scenarios have a high kinematic redundancy can be exploited [219]. We can therefore also mirror control strategies for robotic grip adjustment, to enable adaptive force modulation for cooperative robotic manipulation in response to environmental feedback. Here, compliant control behaviors are achieved through impedance control schemes on the object level [181]. Dexterity in contact-rich manipulation is often limited, because in contrast to humans, typically grasping models for robotic hands only use single contact points at the fingertips, instead of the whole surface [33]. Smart gripper design can alleviate this issue to a certain extent, where the geometry is optimized to achieve more dexterous manipulation capabilities, such as fixing the movement of the object's pose along sphere [178]. This highlights the importance of geometric considerations for encoding spatial relationships and constraints and the role of geometric primitives.

Geometric primitives also play an important role in recent advances on general manipulation control, currently mainly in the form of object-centric keypoints. The goal here is to reduce complex tasks across a category of objects, where shape variations are allowed, to geometric constraints on keypoints [87]. When used in closed-loop manipulation control for achieving contact-rich tasks, this keypoint-based representation leads to automatic generalization to a category of objects by incorporating an object-centric action representation [153]. The simplicity of the representation via keypoints has also been proven to have advantages for visual imitation learning. Here, task representations are extracted from demonstrations and decomposed into keypoint-based geometric constraints [86]. This approach has been extended to bimanual manipulation and was shown to generalize to novel scenes in-category objects, due to the object-centric, embodiment-independent, and viewpoint-invariant representation [85]. Expressing task constraints via geometric constraints on keypoints also facilitates the connection to vision-language models, such that tasks can be defined using language instructions [105].

6.2. Method

Geometric algebra enables the direct representation of geometric primitives within the algebra. Here, we show how to include these geometric primitives in the control objective by deriving control strategies based on similarity transformations.

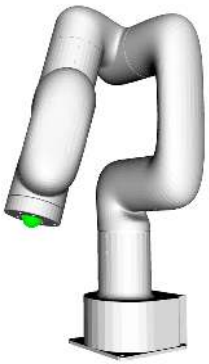
6.2.1. Cooperative Geometric Primitives

In this chapter, we focus on modeling different control objectives for establishing coordination strategies between multiple parallel kinematic chains using geometric primitives. Hence, in the following, we assume that the task-space modeling of the robotic system is done via one of the geometric primitives of CGA. We call this primitive the cooperative geometric primitive and denote it as $X_c(\boldsymbol{q})$, where $\boldsymbol{q}$ is the joint configuration of the robotic system. In order to formulize the cooperative geometric primitives, we extend the ideas around the cooperative pointpair that was shown in Equation (5.6) to n parallel kinematic chains. We use the properties of the outer product of points to find the cooperative geometric primitive $X_c(\boldsymbol{q})$ formed by the end points of those kinematic chains

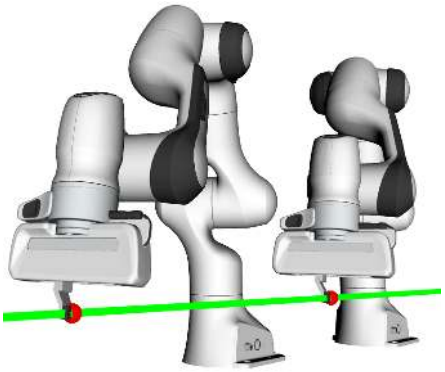
$$X_c(\boldsymbol{q}) = \bigwedge_{i=1}^{n} P_i(\boldsymbol{q}) = \bigwedge_{i=1}^{n} M_i(\boldsymbol{q}_i)\boldsymbol{e}_0\widetilde{M}_i(\boldsymbol{q}_i), \tag{6.1}$$

where the conformal points P_i correspond to the end points of the kinematic chains. The trivial case of one kinematic chain is a point. Two kinematic chains give a pointpair, three a circle, and four a sphere. Examples of the cooperative geometric primitives for varying numbers of kinematic chains can be seen in Figure 6.6. Here, we use three manipulators as a proxy for a three-fingered hand to highlight the mathematical equivalence between multiple robot manipulators and robotic hands with multiple fingers. Note that, instead of a pointpair, we could also use a line and instead of a circle, a plane. Both would only require multiplication with the point at infinity and would lead to a relaxed control law that allowed more degrees of freedom. In general, this construction of cooperative geometric primitives holds for geometric algebras of different dimensions, i.e. for any number of n. In our case, using $\mathbb{G}_{4,1}$, which is a 5-dimensional algebra, the highest dimensional geometric primitive is a sphere. A sphere is constructed from $n = 4$ points, i.e. we limit our view in this section on a maximum of four parallel kinematic chains. In theory, however, using higher dimensional geometric algebras, the derived formulas could be extended to $4 + n$ parallel kinematic chains. We will go into more detail on this point in the discussion in Section 6.4.6.

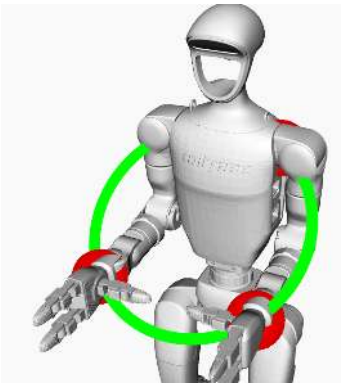
The objective is then to reach a desired geometric primitive of the same type, which is denoted as X_d. We base the formulation of the control objective for cooperative manipulation control on similarity transformations. Suppose that, for any pair of geometric primitives $X_c(\boldsymbol{q})$ and X_d, there is at least one similarity transformation Z that trans-



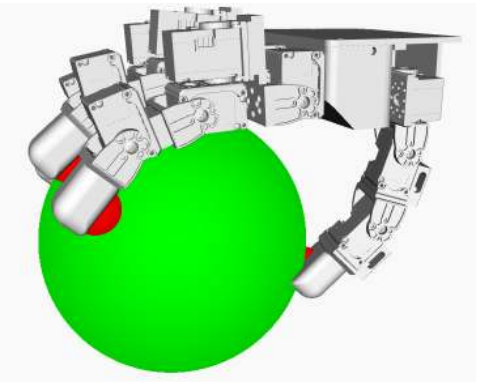
(a) End-effector point of a Ufactory Lite6 robot.



(b) The cooperative dual task space for two Franka robots.



(c) The cooperative circle for a humanoid formed by its two wrists and a point on the torso.



(d) The cooperative sphere of the four fingers of the Leap Hand.

Figure 6.6.: Cooperative geometric primitives for multiple kinematic chains. The red points indicate the reference points on the robots that are considered for the cooperative geometric primitive, which is then shown in green.

forms $X_c(\boldsymbol{q})$ into X_d. Mathematically, the control objective can then be expressed as an optimization problem

$$\min \left\| \log(Z(X_c(\boldsymbol{q}), X_d)) \right\|, \tag{6.2}$$

where we use the logarithmic map of the versor representation, as explained in Section 2.3.2.

Here, we do not propose any particular solver for this optimization problem, but focus on the general modeling. In practice, this optimization problem can be solved using state-of-the-art methods from the literature around optimal control, impedance/admittance control or simply using differential kinematics. To account for this generality of the problem formulation, we give all the mathematical details that are necessary for these different control paradigms. In particular, we derive the gradients, Hessians and Jacobians. In the following, we explain each of these aspects more concretely.

The analytic Jacobian for the cooperative geometric primitive $\boldsymbol{\mathcal{J}}_c^A(\boldsymbol{q})$ can be found as the derivative of $X_c(\boldsymbol{q})$ w.r.t. the joint angles, i.e.

$$\boldsymbol{\mathcal{J}}_c^A(\boldsymbol{q}) = \frac{\partial}{\partial \boldsymbol{q}} X_c(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{\mathcal{J}}_{c,1}^A(\boldsymbol{q}) & \dots & \boldsymbol{\mathcal{J}}_{c,n}^A(\boldsymbol{q}) \end{bmatrix}, \tag{6.3}$$

where

$$\boldsymbol{\mathcal{J}}_{c,i}^A(\boldsymbol{q}) = P_1 \wedge \boldsymbol{\mathcal{J}}_{P,i} \wedge \ldots \wedge P_n, \tag{6.4}$$

and

$$\boldsymbol{\mathcal{J}}_{P,i} = \boldsymbol{\mathcal{J}}_i^A \boldsymbol{e}_0 \widetilde{M_i} + M_i \boldsymbol{e}_0 \widetilde{\boldsymbol{\mathcal{J}}_i^A}. \tag{6.5}$$

We could then use this Jacobian to define a control law similar to differential kinematics, where we take into account the difference between the current and the target geometric primitive

$$\boldsymbol{u} = \big(\boldsymbol{\mathcal{J}}_c^A\big)^{-1}\big(X_d - X_c(\boldsymbol{q})\big), \tag{6.6}$$

however, this control law does not take into account the manifold of the motion between the geometric primitives and is also hard to tune, since the parameters would lie in the subspace of the geometric primitives, which can be hard to interpret. Instead, we now present the control formulation that is based on the similarity transformations between the geometric primitives.

6.2.2. Similarity Transformations

Similarity transformations form a Lie group $\mathcal{Z}$ that is a seven dimensional manifold. We define the canonical decomposition of a general similarity transform versor to be

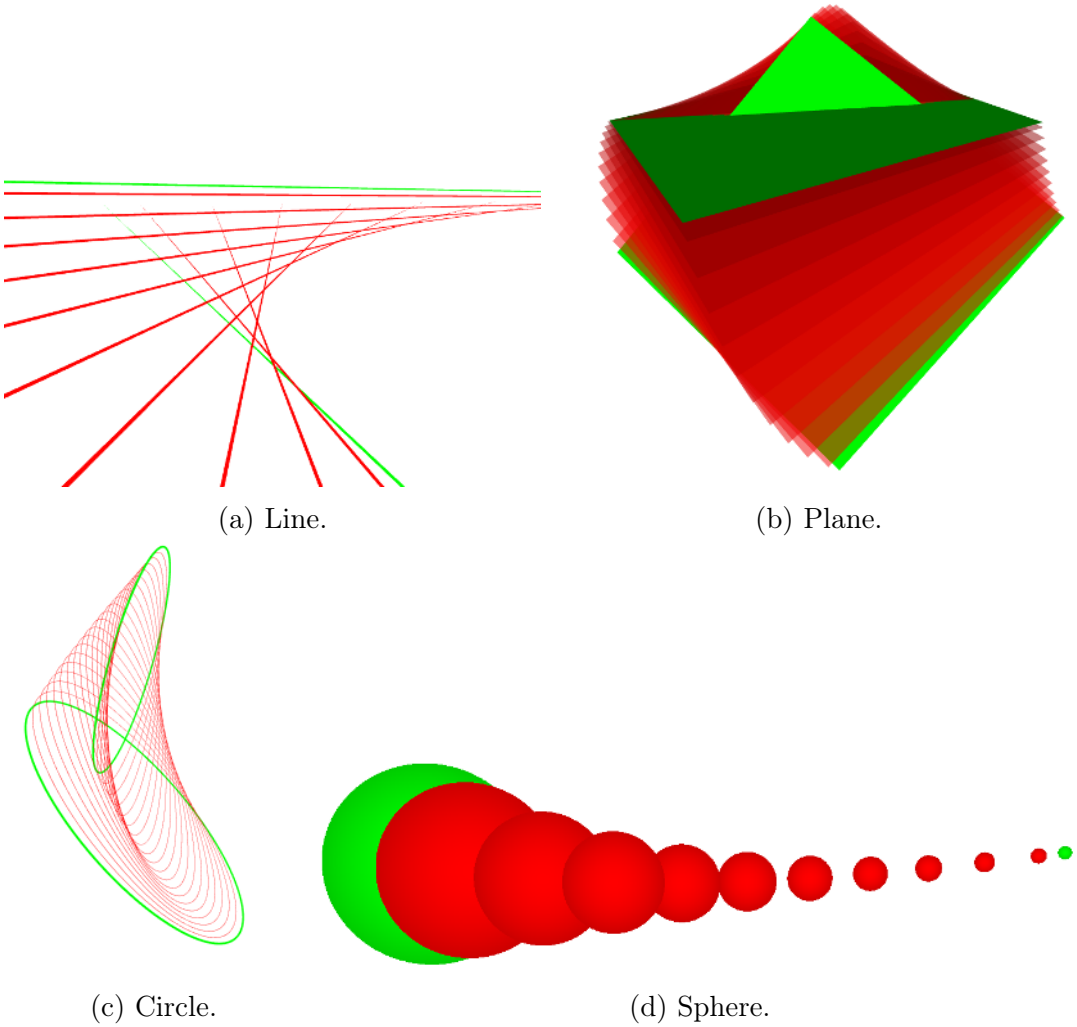
$$Z = TRD. \tag{6.7}$$

Elements of the associated Lie algebra $\mathbb{B}_Z$ can be found via the logarithmic map

$$\begin{aligned} B_Z &= \log(Z) \\ &= \log(T) + \log(R) + \log(D). \end{aligned} \tag{6.8}$$

More details on the transformation groups and their logarithmic maps can be found in Section 2.3.2. In the same way the Lie algebra of quaternions has been used for interpolation between group elements in applications such as spherical linear interpolation, we can use elements $B_Z \in \mathbb{B}_Z$ for interpolating a similarity transformation between two geometric primitives. We show this interpolation in Figure 6.7.

Given the two geometric primitives, X_1 and X_2, we now derive the similarity trans-



(a) Line. (b) Plane.

(c) Circle. (d) Sphere.

Figure 6.7.: Interpolation of different geometric primitives using similarity transformations.

formation $Z(X_1, X_2)$ that fulfills

$$X_2 = Z(X_1, X_2) X_1 \widetilde{Z}(X_1, X_2). \tag{6.9}$$

Its bivector $B_Z(X_1, X_2)$ found from the logarithmic map can then be used in the optimization problem given in Equation (6.2). A general formula to find a versor $V_C(X_1, X_2)$ that transforms X_1 to X_2 was derived in [130]

$$V_C(X_1, X_2) = c^{-1} \left(1 + X_2 X_1\right), \tag{6.10}$$

where c is a normalization constant that depends on the geometric primitives. However, using this general formula, $V_C(X_1, X_2) \in \mathcal{C}$, i.e. the resulting versor will be an element of

the conformal transformation group $\mathcal{C}$, not of the similarity transformation group $\mathcal{Z}$. In order to restrict the versor to $\mathcal{Z}$, we present the necessary derivations that give a similarity transformation between pairs of geometric primitives according to Equation (6.9). Note that this clearly shows that the transformation between primitives is not unique, as was already stated in [130].

Point to Point

The simplest case is transforming one point to another. Since points neither have an orientation nor a size, the rotor R and dilator D simplify to the identity transformation, i.e. $R = D = 1$. Regarding the translation versor, given the two points P_1 and P_2, it is found as

$$\begin{aligned} Z(P_1, P_2) =& T(P_1, P_2) \\ =& \exp\big((P_2 - P_1) \wedge \boldsymbol{e}_\infty\big). \end{aligned} \tag{6.11}$$

Line to Line

Lines have a flat geometry, i.e. they are part of Euclidean geometry, which means a transformation between two lines can be described purely by rotations and translations. Hence, the versor resulting from Equation (6.10) will only contain a translation and a rotation, i.e. $Z(L_1, L_2) \in \mathcal{M}$.

Plane to Plane

Using the same argumentation as for the case of line to line transformations, the transformation from one plane to another can be found using Equation (6.10). The resulting versor will only contain a translation and a rotation, i.e. $Z(E_1, E_2) \in \mathcal{M}$.

Circle to Circle

Circles require all three versors in the similarity transformation: translations, rotations and dilations. The dilation versor we can directly compute based on the radii r_1 and r_2 of X_1 and X_2

$$D(X_1, X_2) = \exp\big(\log(r_2 r_1^{-1}) \boldsymbol{e}_{0\infty}\big), \tag{6.12}$$

where the radius of a circle can be computed as

$$r = \left\| (X \cdot \boldsymbol{e}_\infty)\, X^{-1} \right\|. \tag{6.13}$$

Note that the inner expression takes the form of the general subspace projection that we showed in Equation (2.28). It can thus be interpreted as projecting infinity to the subspace of the circle and then evaluating the resulting norm.

Next, we compute the rotation versor between the circles by using the normals N_1 and N_2 of the planes that the circles lie. These planes can be found via $E = C \wedge \boldsymbol{e}_\infty$ and the normal of plane E is computed as

$$N = E^* - \frac{1}{2}(E^* \cdot \boldsymbol{e}_0)\boldsymbol{e}_\infty. \tag{6.14}$$

We then insert the normals in Equation (6.10), which yields

$$\begin{aligned} R(X_1, X_2) &= c^{-1}(1 + N_1 N_2) \\ &= c^{-1}\Big(1 + (N_2 \cdot N_1) - (N_2 \wedge N_1)\Big). \end{aligned} \tag{6.15}$$

Note that this is the geometric algebra variant of the standard expression for computing a unit quaternion between unit vectors.

Lastly, we compute the translation versor via the center points of the circles. The center point of circles can be found as

$$P(X) = X\boldsymbol{e}_\infty X, \tag{6.16}$$

which in this case amounts to reflecting infinity in the circle or sphere. Given the center points, the translation versor can then be computed as was indicated for points in Equation (6.11). Note that, for computing the center of the cooperative circle, we first apply the transformation given by $R(X_1, X_2)D(X_1, X_2)$, because this transformation changes the center point of the circle.

Sphere to Sphere

For spheres, the rotation versor is the identity, because spheres do not have an intrinsic orientation. The dilation and translation versors can then be calculated using the same strategy as for the circles. The computation of the dilation versor follows Equation (6.12), where the radius of a sphere is also computed in the same way as for the circle, i.e. using Equation (6.13). We then use Equation (6.16) to compute their center points and afterwards Equation (6.11) to obtain the translation versor.

6.2.3. Cooperative Similarity Control

In Equation (6.2), we formulated the problem of cooperative manipulation control as minimizing the similarity transformation between the cooperative and desired geometric primitives, $X_c(\boldsymbol{q})$ and X_d, via its logarithmic map. After having derived the similarity transformations between geometric primitives in Section 6.2.2, we can now formulate the control strategies based on them.

First, we define the cooperative similarity transformation $V_{Z,c}(\boldsymbol{q})$. Similarly to how an end-effector pose represents the transformation of the coordinate system at the origin to the end-effector, it is expressed as the similarity transformation w.r.t. a unit geometric primitive at the origin. We denote this primitive as X_u. In section 6.2.2, we have already presented how to similarity transformations between two geometric primitives of the same kind. Accordingly, the versor $V_{Z,c}(\boldsymbol{q})$ can be found following the explanations around Equation (6.7) as

$$V_{Z,c}(\boldsymbol{q}) = Z\left(X_u, X_c(\boldsymbol{q})\right). \tag{6.17}$$

This versor will then fulfill $X_c(\boldsymbol{q}) = V_{Z,c}(\boldsymbol{q}) X_u \widetilde{V}_{Z,c}(\boldsymbol{q})$. It is a direct equivalent to the end-effector motor found by the forward kinematics in Equation (3.4), albeit it represents a cooperation between multiple kinematic chains and not just a single one. Using the unit geometric primitive, we also define a desired similarity transformation

$$V_{Z,d} = Z\left(X_u, X_d\right). \tag{6.18}$$

Thus, the versor from Equation (6.2) can be written as

$$Z(X_c(\boldsymbol{q}), X_d) = V_{Z,cd}(\boldsymbol{q}) = \widetilde{V}_{S,c}(\boldsymbol{q}) V_{S,d}, \tag{6.19}$$

which is equivalent to the formulation of inverse kinematics for a single kinematic chain based on the end-effector motor. Note that, instead of using Equation (6.19), the versor $Z(X_c(\boldsymbol{q}), X_d)$ can also directly be found following the explanations in Section 6.2.2. From Equation (6.19), we can then find the bivector $B_{Z,cd}(\boldsymbol{q})$ via the logarithmic map that represents the task-space control signal, similar to a twist,

$$B_{Z,cd}(\boldsymbol{q}) = \log\left(V_{Z,cd}(\boldsymbol{q})\right). \tag{6.20}$$

Note that, in general, this bivector can be seven-dimensional, since the similarity transformation group is a seven-dimensional manifold.

Most commonly, optimization solvers and control algorithms make use of one of various Jacobians that can be found for the cooperative similarity versor $V_{Z,c}(\boldsymbol{q})$ and its bivector $B_{Z,c}(\boldsymbol{q})$. Hence, we now present the bivector similarity Jacobian $\boldsymbol{\mathcal{J}}_Z^B(\boldsymbol{q})$, the analytic similarity Jacobian $\boldsymbol{\mathcal{J}}_Z^A(\boldsymbol{q})$, and the geometric similarity Jacobian $\boldsymbol{\mathcal{J}}_Z^G(\boldsymbol{q})$. These Jacobians can then be used to find the gradient, approximate the Hessian, or compute a control signal, depending on the chosen control approach.

As per the usual definition, the analytic similarity Jacobian can be found from the

partial derivatives of the cooperative similarity versor w.r.t. $\boldsymbol{q}$, i.e.

$$\begin{aligned}\boldsymbol{\mathcal{J}}^{A}_{Z,c}(\boldsymbol{q}) =& \frac{\partial}{\partial \boldsymbol{q}} V_{Z,c}(\boldsymbol{q}) \\ =& \boldsymbol{\mathcal{J}}^{A}_{T,c}(\boldsymbol{q}) V_{R,c}(\boldsymbol{q}) V_{D,c}(\boldsymbol{q}) \\ &+ V_{T,c}(\boldsymbol{q}) \boldsymbol{\mathcal{J}}^{A}_{R,c}(\boldsymbol{q}) V_{D,c}(\boldsymbol{q}) \\ &+ V_{T,c}(\boldsymbol{q}) V_{R,c}(\boldsymbol{q}) \boldsymbol{\mathcal{J}}^{A}_{D,c}(\boldsymbol{q}),\end{aligned} \tag{6.21}$$

where $\boldsymbol{\mathcal{J}}^{A}_{T,c}(\boldsymbol{q}), \boldsymbol{\mathcal{J}}^{A}_{R,c}(\boldsymbol{q})$, and $\boldsymbol{\mathcal{J}}^{A}_{D,c}(\boldsymbol{q})$ are the Jacobians of the translation, rotation and dilation versors, respectively. We omit the full derivation here for conciseness, which can be found in Appendix A.

In the same way that the usual kinematic modeling of robots makes a distinction between the analytic and the geometric Jacobian, via the group constraint and from the relationship between the end-effector Jacobians shown in Equation (3.15), it is easy to see that the following relationship holds:

$$\boldsymbol{\mathcal{J}}^{G}_{Z,c}(\boldsymbol{q}) = -2\widetilde{V}_{Z,c}(\boldsymbol{q}) \boldsymbol{\mathcal{J}}^{A}_{Z,c}(\boldsymbol{q}). \tag{6.22}$$

The geometric similarity Jacobian can be used to formulate control laws such as differential kinematics or impedance/admittance control. The traditional control methods then require a twist as a task-space control command. Here, the cooperative similarity bivector $B_{Z,cd}(\boldsymbol{q})$ from Equation (6.20) can be used to define a corresponding similarity twist $\mathcal{V}_Z$.

Lastly, the bivector similarity Jacobian $\boldsymbol{\mathcal{J}}^{B}_{Z}(\boldsymbol{q})$ is found as the partial derivates of the cooperative similarity bivector $B_{Z,cd}(\boldsymbol{q})$ w.r.t. $\boldsymbol{q}$

$$\boldsymbol{\mathcal{J}}_{B,c}(\boldsymbol{q}) = \boldsymbol{\mathcal{J}}_{\mathcal{Z}\to\mathbb{B}_Z}(\boldsymbol{q}) \boldsymbol{\mathcal{J}}^{A}_{Z,c}(\boldsymbol{q}), \tag{6.23}$$

where $\boldsymbol{\mathcal{J}}_{\mathcal{Z}\to\mathbb{B}_Z}(\boldsymbol{q})$ is the Jacobian of the logarithmic map of the similarity transformation group. The bivector similarity Jacobian can then be used in applications such as optimization-based inverse kinematics.

6.2.4. Manipulability Analysis

For a more fundamental understanding of the proposed cooperative similarity control of these complex robotic systems, we define the manipulability of the system. The cooperative similarity transformation fulfills the same role as the rigid body transformation at the end-effector, represented as a motor, for robotic arms. For traditional manipulability analysis, the geometric Jacobian related to this end-effector motor is used. Accordingly,

we define the similarity manipulability ellipsoid as

$$\boldsymbol{M}_Z(\boldsymbol{q}) = \boldsymbol{\mathcal{J}}_Z^G(\boldsymbol{q})\big(\boldsymbol{\mathcal{J}}_Z^G(\boldsymbol{q})\big)^\top, \tag{6.24}$$

which further highlights the parallels of the system modeling between single kinematic chains using motors and multiple kinematic chains using the cooperative similarity transformation.

The similarity manipulability ellipsoid $\boldsymbol{M}_Z(\boldsymbol{q}) \in \mathbb{R}^{7\times7}$ can be understood as a direct equivalent to the traditional velocity manipulability ellipsoid. Hence, it informs about the system's capability to move the cooperative geometric primitive, which is the same as the traditional manipulability's information about the robot's ability to move its end-effector. In addition, the cooperative similarity manipulability also contains a dimension that expresses the ability to dilate the cooperative geometric primitive. This dimension becomes of particular interest, when we look at the inverse manipulability $\boldsymbol{M}_Z(\boldsymbol{q})$. This inverse represents the force manipulability and thus expresses the system's ability to enact forces in different directions. For the dilation, this expresses the ability to apply force that increase or decrease the size the of the cooperative geometric primitive. For example, in the case of three manipulators, this would give a measure of the radial force for changing the radius of the cooperative circle. This is important information for tasks such as carrying big and bulky objects, where the three manipulators are required to apply force to an object while carrying it. The cooperative force manipulability can then be used to optimize for the ideal configuration to enable the maximum force on the dilation. Similarly, this manipulability ellipsoid could then also be used to implement standard techniques for avoiding the singularities discussed in Section 6.4.3.

6.3. Results

We have presented a purely geometric framework for the cooperative control of multiple kinematic chains. Accordingly, the experiments are designed to illustrate the underlying mathematical concepts developed in this work through kinematic simulations. We show the cooperative geometric primitives both in optimal control and in teleoperation scenarios. To maintain generality, the examples are intentionally presented in an abstract form, without specific applications. Instead, we only refer to potential applications. The presented results are implemented using our open-source software framework *gafro*[1]. Additional material and videos of the teleoperation experiments can be found on our website[2].

[1] `https://gitlab.com/gafro`
[2] `https://geometric-algebra.tobiloew.ch/cooperative_geometric_primitives/`

6.3.1. Optimal Control Experiments

In this section, we show reaching cooperative geometric primitives using multiple kinematic chains formulated as optimal control problems. Apart from a control cost for regularization, we only use a final cost to reach a desired geometric primitive. Hence, we minimize the transformation between the current cooperative geometric primitive and a desired one, which amounts to minimizing the parameters of the corresponding bivector, i.e. as shown in Equation (6.2). A bivector of zero corresponds to the identity transformation. The experiments only show the simplest case of reaching a single geometric primitive. For real applications, however, a combination of various different primitive constraints can be employed.

Line Reaching

In our first experiment, we examine the simplest case: two points representing the cooperative task space of two independent kinematic chains, such as a bi-manual platform or the arms of a humanoid. Within CGA, we can introduce a third point at infinity to define an infinite line passing through the two original points. By using a line instead of a point pair to model the cooperative behavior of the two arms, we remove constraints on the motion and allow for the arms to move freely along the line, which essentially defines a geometric nullspace. Here, we illustrate a reaching motion from one line primitive to another in Figure 6.8 with a humanoid model. In Figure 10.1, we have shown the example of cloth folding as a potential application for a dual arm system modeled by a cooperative line.

Circle Reaching

In the second experiment, we consider three points that define the cooperative task space of three independent kinematic chains. In CGA, three non-collinear points uniquely define a circle primitive. Like a line, a circle introduces a null space, allowing the three points to move freely along it. We demonstrate transitions from one circle to another using three different setups: a trio of manipulators, the torso and arms of a humanoid robot, and a collaborative reaching scenario involving both a manipulator and a humanoid.

The first scenario of three manipulators, which is shown in Figure 6.9, is the most generic one. It can be seen as a general proxy for tasks involving the lifting and manipulation of big and bulky objects.

We show the second scenario in Figure 6.10. Here, we use a single humanoid and choose a point on the torso and its two arms as the three points that define the circle. This example illustrates how a cooperative circle can be used to model a single humanoid carrying a big and bulky object in arms, while pressing the object against its torso

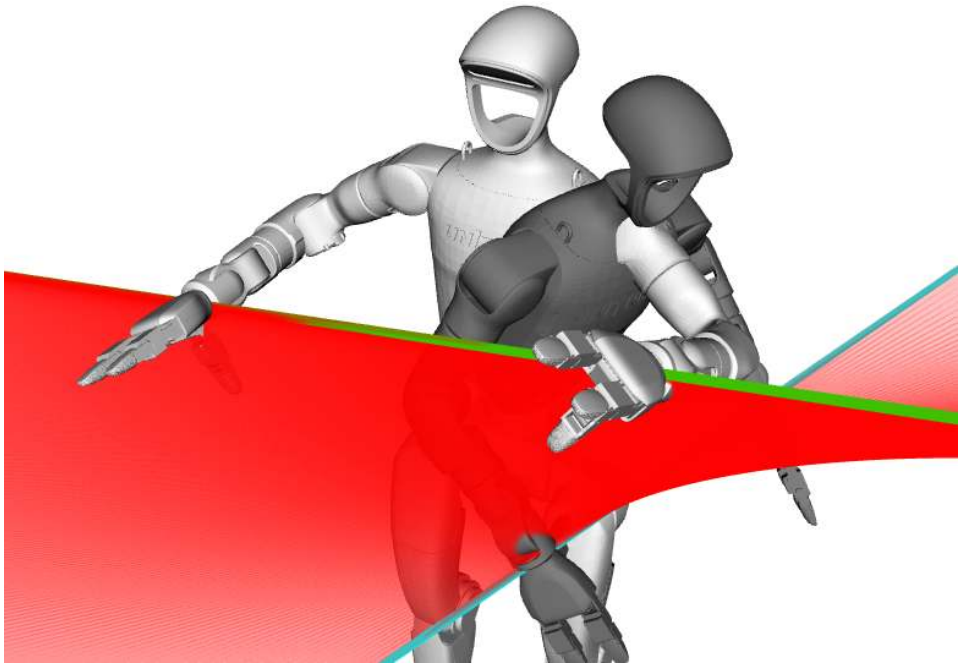


Figure 6.8.: Humanoid reaching for a desired cooperative line. The points used for modeling the cooperative line are located at the wrists. Including the joints at the waist, this system has 17 degrees of freedom. The initial configuration is shown in white and the final one in gray. The green line is the initial cooperative line, the turquoise one the final, and in red we show its trajectory in between.

to provide additional support. We show an illustration of this task for a human in Figure 10.1.

The last scenario for the cooperative circle features a manipulator and a humanoid that are collaborating for the reaching of the desired circle, as shown in Figure 6.11. Although, we use a humanoid in this example, it could be easily replaced by a human in order to model a human-robot collaboration scenario. Since this extension provides an interesting opportunity for future work, we discuss it further in Section 6.4.5.

Plane Reaching

Similar to the line, an infinite plane in CGA can be constructed by combining three points with a point at infinity. As with other primitives in CGA, a plane represents a geometric null space. Therefore, moving the three points within the same plane results in an equivalent control objective. We demonstrate this behavior using a three-fingered robotic hand, as shown in Figure 6.12. In Figure 10.1, we have used the example of a human carrying a plate as a potential application for this scenario.

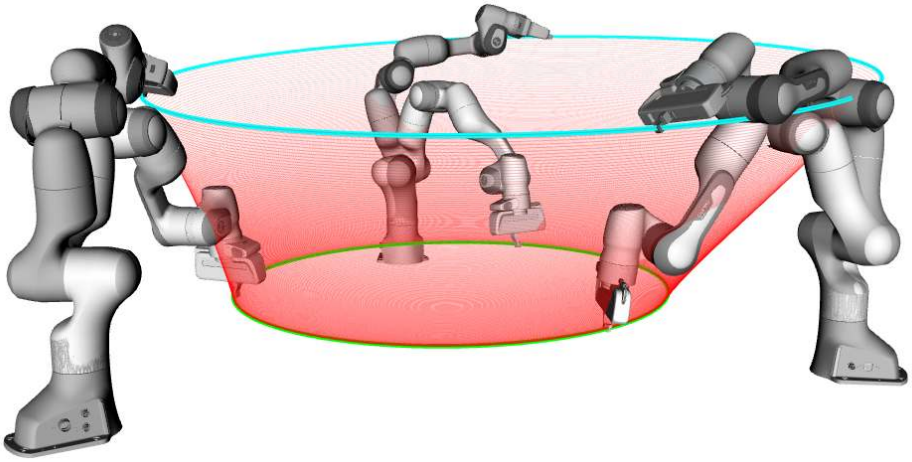


Figure 6.9.: Three Franka robots reaching for a cooperative circle. Since each robot has seven degrees of freedom, the complete system has 21 degrees of freedom. The initial configuration is shown in white and the final one in gray. The green circle is the initial cooperative circle, the turquoise one the final, and in red we show its trajectory in between.

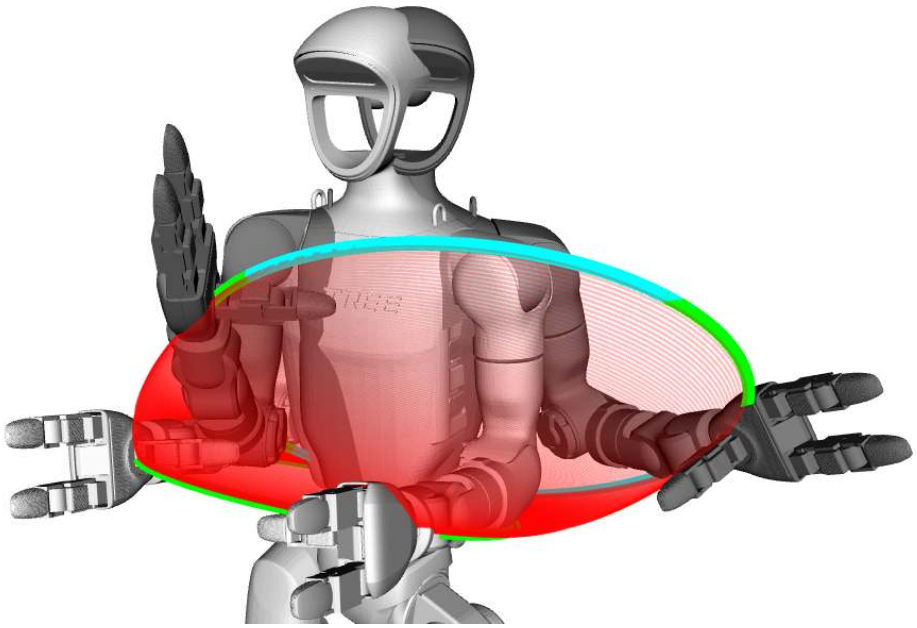


Figure 6.10.: Humanoid reaching for a cooperative circle by using a point on its torso. The initial configuration is shown in white and the final one in gray. The green circle is the initial cooperative circle, the turquoise one the final, and in red we show its trajectory in between.

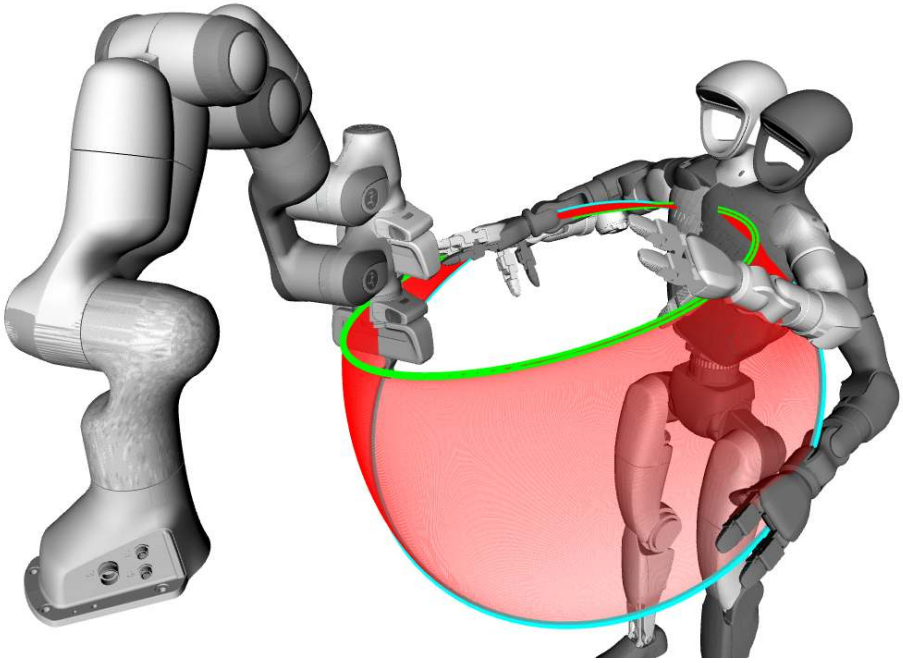


Figure 6.11.: Collaboration between a manipulator and a humanoid modeled by a cooperative circle. The initial configuration is shown in white and the final one in gray. The green circle is the initial cooperative circle, the turquoise one the final, and in red we show its trajectory in between.

Sphere Reaching

As the last experiment, we considered the sphere primitive constructed using four points, which is the limit that we can reach with CGA. We demonstrate reaching from one sphere to another using a four-fingered hand in Figure 6.13, and a collaborative scenario involving two humanoids in Figure 6.14. The cooperative sphere of the four-fingered hand is a general example for grasping using a robotic hand and can thus be applied to a wide range of applications. We also show an example of this in Figure 10.1. Similarly, in the scenario with the humanoids, one of the humanoids could be replaced by the user in order to model human-robot collaboration. Generally, this example illustrates the collective transport of big and bulky objects using two independent humanoids, as depicted in Figure 10.1.

6.3.2. Teleoperation Experiment

In this section, we present teleoperation for controlling complex robotic systems with a high number of degrees of freedom. In particular, we consider

three manipulators: 21 DoF,
two humanoids: 34 DoF,

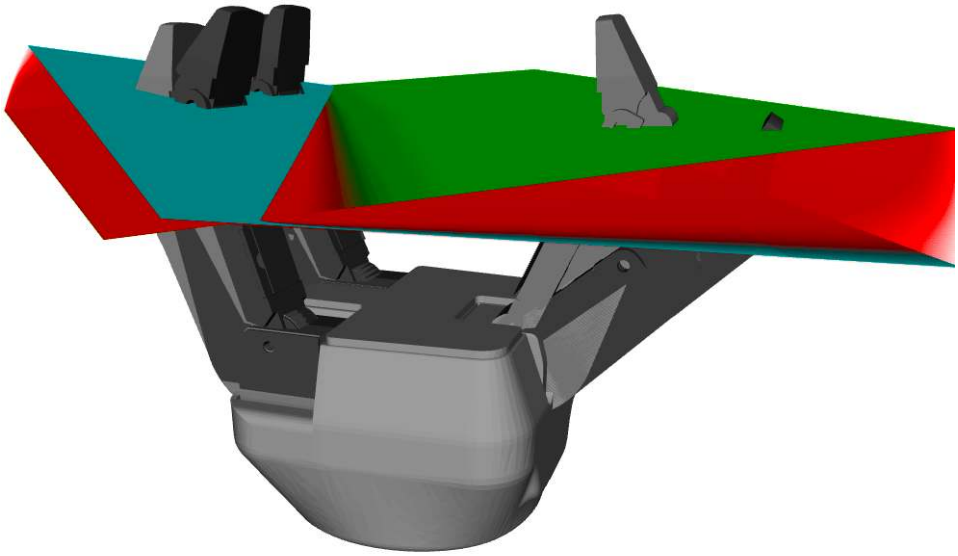


Figure 6.12.: Three-fingered hand reaching for a cooperative plane. The initial configuration is shown in white and the final one in gray. The green plane is the initial cooperative plane, the turquoise one the final, and in red we show its trajectory in between.

robotic hand: 16 DoF.

The purpose of the experiment is to demonstrate the simplicity of the cooperative geometric primitives, and how using the cooperative geometric primitives for modeling the cooperative task space of multiple parallel kinematic chains constrains the degrees of freedom and thus greatly facilitates the control of these complex systems.

In all cases, we use a readily available six degree of freedom device that is commonly used in CAD applications as a single input device to obtain the commands for the teleoperation. We then map the six axes of the input device to a Lie algebra element of the similarity transformation group, i.e. a bivector, that then represents a task space command. The corresponding joint-space command we then obtain by using differential kinematics, i.e. we invert the geometric similarity Jacobian

$$\dot{\boldsymbol{u}} = \left(\boldsymbol{\mathcal{J}}_Z^G(\boldsymbol{q})\right)^{-1} \boldsymbol{b}_{Z,i}, \tag{6.25}$$

where $\boldsymbol{b}_{Z,i} = \boldsymbol{\mathcal{B}}_{Z,i}$ is the parameter vector of the input similarity bivector. The control scheme is formulated w.r.t. to the cooperative similarity transformation, i.e. the local frame. Note that this control formulation is identical to the usual way of doing differential kinematics. The only difference is that a cooperative task space of multiple kinematic chains is modeled, as opposed to a single one.

We show in Figure 6.15 the mapping of the input device's axes to the similarity

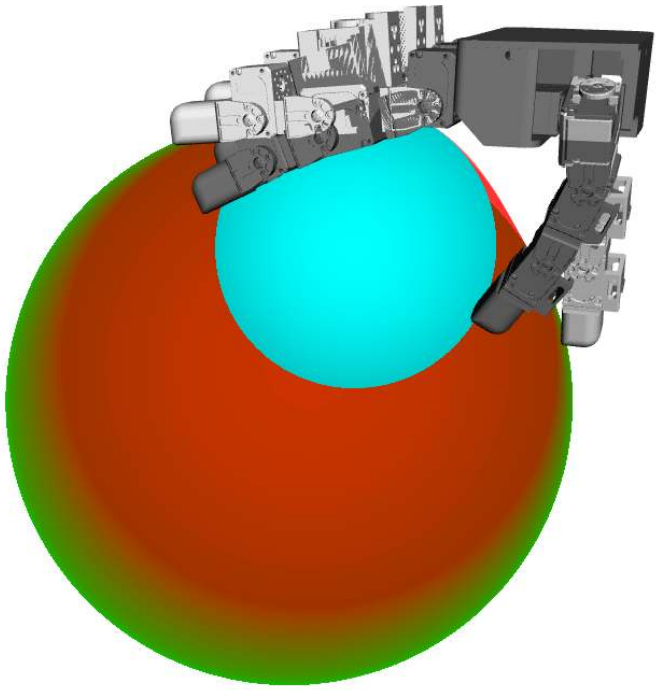


Figure 6.13.: Four-fingered hand reaching for a cooperative sphere. The hand has 16 degrees of freedom. The initial configuration is shown in white and the final one in gray. The green sphere is the initial cooperative sphere, the turquoise one the final, and in red we show its trajectory in between.

transformations, and the corresponding motion of the robotic hand. We want to point out that the sphere in that Figure is only visualizing the current cooperative sphere. It is, however, not used for control, in the sense that the sphere is moved and then tracked by the fingers. Instead, the controller from Equation (6.25) causes a joint movement that then results in the sphere changing accordingly.

6.4. Discussion

Our unified control framework based on cooperative similarity transformations, generalizes the rigid body transformation of a single robot's end-effector to the cooperative control of multiple kinematic chains. The experimental results demonstrated this approach across a variety of robotic systems, ranging from a bi-manual setup to scenarios involving collaborating humanoids. Given that, the framework is purely geometric, and considering the variety of systems explored, the kinematic simulations show both the strengths and limitations of our method, which are further discussed in the following subsections.

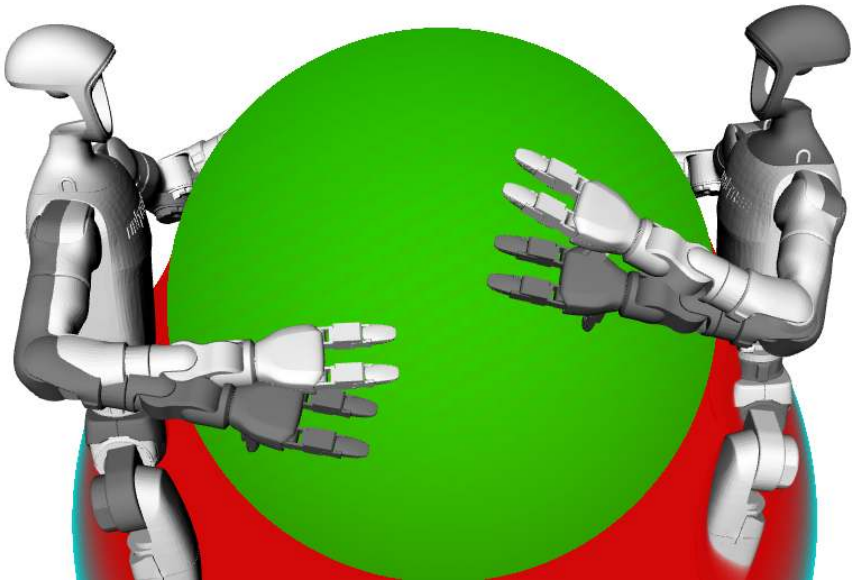


Figure 6.14.: Two humanoids reaching for a cooperative sphere. The combined system has 34 degrees of freedom, including both humanoids' waist joints. The initial configuration is shown in white and the final one in gray. The green sphere is the initial cooperative sphere, the turquoise one the final, and in red we show its trajectory in between.

6.4.1. Connection to Traditional Single-Arm Control

Throughout this chapter, we have mentioned several times that the cooperative similarity transformation and its Jacobians fulfill the same role as the rigid body transformation to the robot end-effector that is used in single-arm systems. Since similarity transformations form a seven-dimensional manifold, as opposed to the six-dimensional one of rigid body transformations, the modeling using the cooperative similarity transformations only introduces one additional dimension to achieve cooperative control behaviors of highly complex robotic systems. Although the involved mathematics that we derived in this chapter might seem unfamiliar, all the findings in the literature for traditional control methods of single-arm systems remain valid and are directly applicable to the scenarios shown in this chapter. As we have shown for example when using differential kinematics control for the teleoperation experiments in Section 6.3.2. This further highlights how the cooperative modeling greatly facilitates the control of complex robotic systems.

6.4.2. Geometric Nullspace

The systems considered in our experiments range from a bimanual setup with 14 degrees of freedom (DoF) (see Figure 6.16a) to collaborating humanoids with 34 DoF in

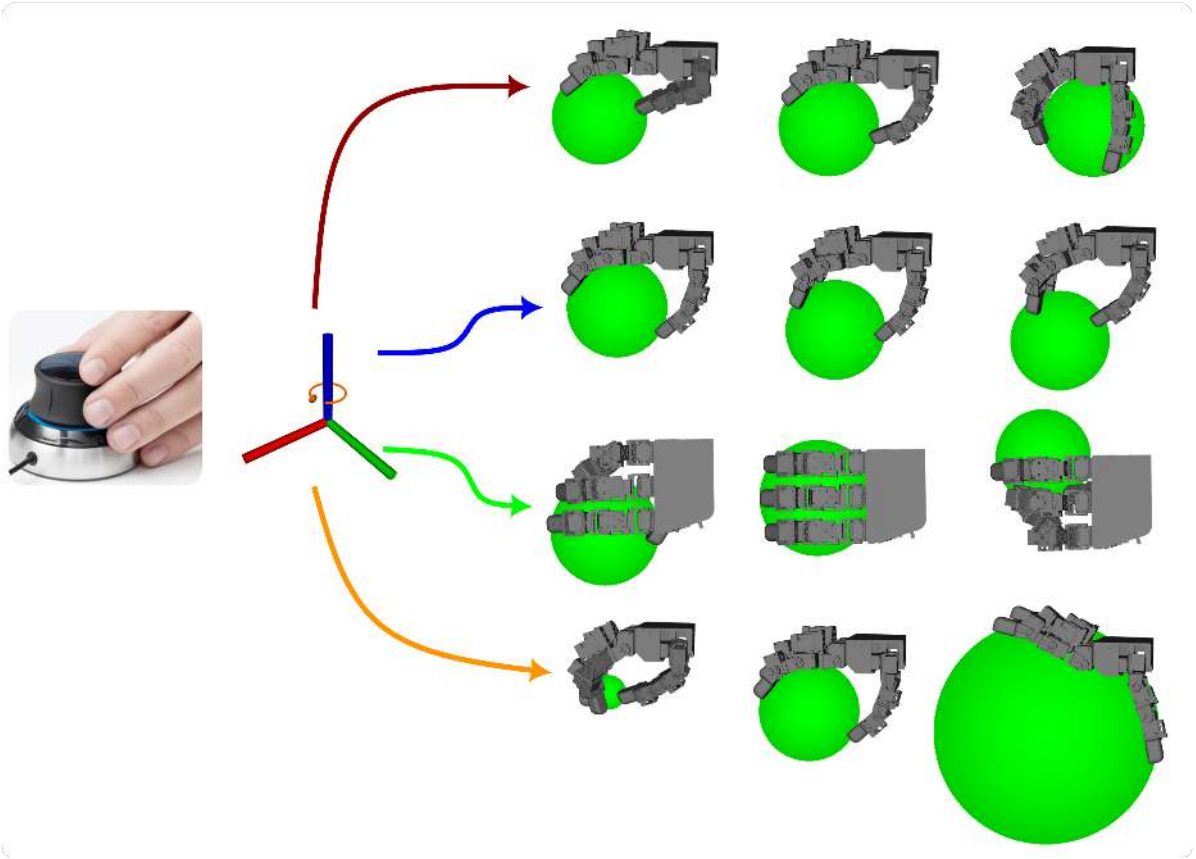
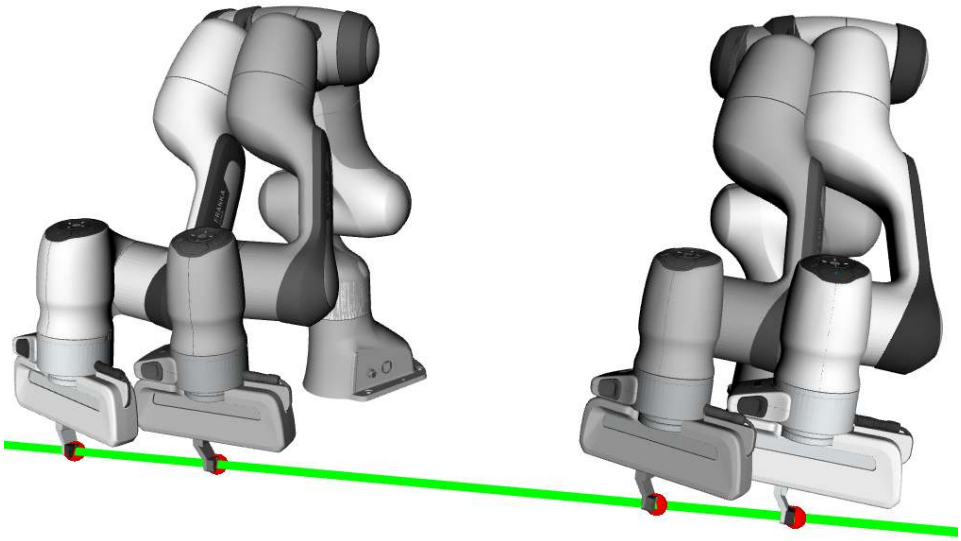


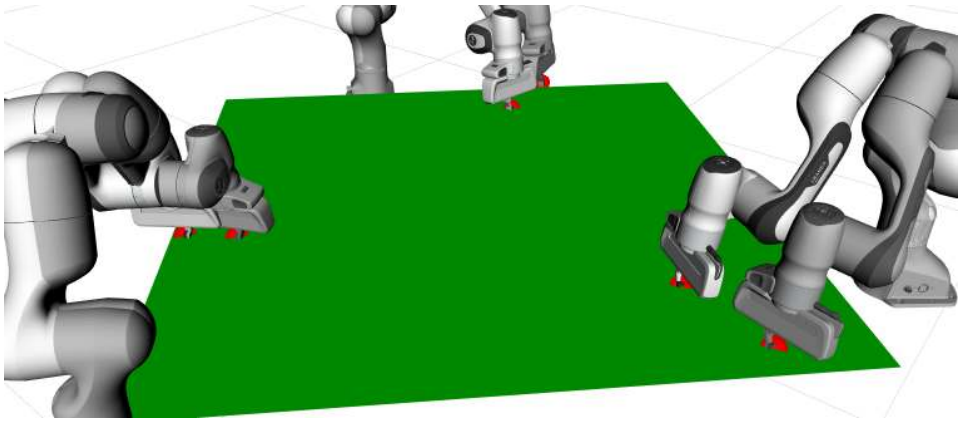
Figure 6.15.: Teleoperation of an anthropomorphic hand. The axes of teleoperation device on the left, are mapped to the different similarity transformations as shown in the center. The three principal axes map to the corresponding translations, while the rotation around the z-axis is mapped to the dilation. Rotations are not required, since a sphere is invariant to rotations around its center. The resulting motion of the robotic hand is then depicted on the right.

total. The resulting systems are highly redundant, and thus are inherently challenging to control. Even when restricted to the task space, the dimensionality remains high, i.e. ranging from 12 to 24 DoF. Notably, encoding the cooperative task space using geometric primitives significantly reduces the effective dimensionality of the problem. The similarity transformations associated with these primitives have at most 7 DoF, accounting for translation, rotation, and uniform scaling. Depending on the symmetry of the primitive, the effective number of DoF can be even lower. For instance, a sphere (see Figure 6.13) admits only 4 DoF, as rotation does not alter its configuration. As can be inferred from the teleoperation experiment given in Figure 6.15, this reduction in dimensionality is critical: no teleoperation device exists for 28-DoF systems, nor can a human or an existing algorithm feasibly control such a high-dimensional system directly.

In contrast, using our proposed method for controlling the translation and dilation of a target sphere is straightforward.



(a) The line nullspace formed by two Franka robots. Each robot's end-effector is free to move along the line unrestricted. Secondary control objectives tangential to the line will not change the line.



(b) The plane nullspace formed by three Franka robots. Each robot's end-effector is free to move in the plane unrestricted. Secondary control objectives tangential to the plane will not change the plane.

Figure 6.16.: Geometric nullspaces that admit the introduction of secondary control objectives. Perturbations that are orthogonal to the geometric primitives will be rejected. The definition of these geometric nullspaces is coordinate-free.

A second key aspect of the redundancy resolution arises from the use of geometric nullspaces. Geometric primitives naturally define subspaces of the underlying algebra, giving rise to geometric nullspace formulations for control. This structure ensures that

the controller is inherently stiff when the system deviates from the primitive, and compliant when moving along it, as we previously showed in the context of optimal control for a single robotic arm [148]. This principle extends directly to the cooperative geometric primitives introduced in this work, as illustrated in Figures 6.16a and 6.16b.

The geometric nullspaces induced by these primitives enable a decoupling of orthogonal control objectives in a coordinate-free manner, while eliminating the need for basis-specific representations or manual tuning of off-diagonal matrix terms. For example, aligning the end-effector with a target line does not constrain motion along the line, enabling the controlled application of contact forces in that direction. We previously demonstrated this feature with a single robot in an impedance control application on curved surfaces [31]. As shown in Figure 6.16a, this property naturally extends to cooperative settings, such as coordinated object lifting.

Similarly, in the case of cooperative reaching toward a plane, optimization yields the closest configuration due to the presence of a geometric nullspace, as shown in Figure 6.16b. Although not illustrated here for brevity, this principle generalizes to other geometric primitives such as circles and spheres. For example, when four robotic arms cooperatively maintain a spherical constraint, they can move compliantly along the tangent directions of the sphere's surface without violating the constraint, while the controller remains stiff in directions orthogonal to the sphere.

6.4.3. Singularity Resolution

Singularities in kinematic chains typically result in the loss of one or more degrees of freedom. These configurations are characterized by a degenerate Jacobian matrix that loses full rank. Similarly, singularities can arise in cooperative geometric primitives when the primitive is no longer uniquely defined. To illustrate this, consider the example of the circle primitive. If all three end-effector points lie on a straight line, the defining circle degenerates into a line. In the conformal geometric model, lines are interpreted as circles with infinite radius. As a result, the cooperative primitive becomes ill-defined, and its behavior cannot be uniquely determined.

These degeneracies produce effects akin to conventional singularities: it becomes ambiguous which end-effector should move, and the associated geometric Jacobian grows unbounded as the three points approach co-linearity. While such configurations are rare and typically do not disrupt control in practice, we included manipulability matrices for the geometric primitives in Section 6.2.4 to provide tools for analyzing and avoiding these cases. Just as manipulability is used to avoid singularities in single-robot systems, these matrices can be employed alongside standard techniques for singularity-avoiding cooperative control.

6.4.4. Extension to Arbitrary Contact Points on the Robot Body

In this chapter, we presented cooperative geometric primitives with respect to end-effector points. While this is a reasonable and fairly standard assumption, since the end-effector typically has the highest manipulability and can be equipped with a tool or sensor, there are many scenarios in which it is desirable to define cooperative geometric primitives with respect to arbitrary points on the robot's body.

For example, when carrying a large and bulky object, humans often use additional contact points such as the torso to help distribute the load and improve stability. We illustrated a preliminary example of this idea in Figure 6.10, where the torso of a humanoid robot was used to define the third point of a cooperative circle.

In the general case, however, this requires to define a distance function from the robot surface to the geometric primitives and use its Jacobian to guide the optimization. Accordingly, the optimization can select these points dynamically, depending on the task requirements.

6.4.5. Extension to Human-Robot Collaboration

In Figures 6.11 and 6.14, we presented experiments involving humanoid robots. With a slight change in perspective, these humanoids can be interpreted as abstractions of humans in a human-robot collaboration setting. This observation suggests that cooperative geometric primitives could also be applied to model scenarios involving human-robot collaboration.

The primary distinction between a human and a humanoid robot lies in controllability: the human's motion cannot be directly controlled. As a result, the system must adopt a leader-follower control paradigm, where the human acts as the leader and the robot as the follower. The robot then adapts its end-effector motion in response to the human's hand movements, in order to maintain a desired cooperative primitive, such as a circle or a sphere, defined by the relative positions of the human hand and the robot end-effector(s). Due to the benefits of using the cooperative geometric primitives, the resulting control scheme would remain geometrically consistent.

6.4.6. Extension to n Kinematic Chains

While we presented control strategies for up to four parallel kinematic chains, this practical limitation only stems from our choice of algebra, i.e. conformal geometric algebra $\mathbb{G}_{4,1}$. Here, CGA is the smallest algebra that allows for the representation of geometric primitives capturing the collaborative behaviour of up to four parallel kinematic chains. This choice is justified, since three and four-fingered hands are common and able to secure a grasp by removing all DoFs of a target object. Hence, we want to point out

that this limitation is only of practical nature, but not of theoretical one. More concretely, using a different choice of the underlying quadratic space $(\mathbb{R}^{p,q,r}, g(\boldsymbol{x}, \boldsymbol{x}))$, the corresponding geometric algebra $\mathbb{G}_{p,q,r}$ would allow for the representation of more parallel kinematic chains, such as in the case of five-fingered hands. The presented results based on the derivation of the collaborative geometric primitives would remain valid.

6.5. Conclusion

In this work, we introduced a framework for modeling the cooperative task space of multiple parallel kinematic chains through the integration of cooperative geometric primitives and similarity transformations. By deriving the mathematical foundations of this approach, we demonstrated that cooperative geometric primitives offer a powerful abstraction for simplifying the representation and analysis of complex robotic systems, particularly those with a high number of degrees of freedom. This abstraction reduces the inherent modeling complexity by encapsulating intricate kinematic interactions into unified geometric constructs, enabling intuitive and scalable coordination strategies. In this chapter, we focused on the mathematical derivation of the cooperative task spaces based on the geometric primitives and presented their application in abstract experiments in order to preserve generality. Hence, this work offers various interesting extension opportunities that could be addressed in future work.

Our formulation of the cooperative similarity transformation presents a direct link to classical control methods. Based on the cooperative similarity transformation, we derived an analytic and geometric Jacobian that can be used in standard control techniques. We demonstrated this using optimal control and teleoperation with differential kinematics as examples. Even though the controlled systems have a high number of degrees of freedom, the controller based on the similarity transformation is almost identical to single-arm controllers, and only adds a single dimension for controlling the dilation. Combined with the concept of geometric nullspaces, which decouple orthogonal control objectives in a coordinate-free manner, we have derived the theory of a versatile control framework that leverages the algebraic properties of geometric primitives. Our experiments have shown the general applicability of this control formulation. Actual real-world tasks, however, will require more complex modeling approaches, where different geometric primitives are combined in a single objective function in order to represent the task constraints. We will address this in future work by focusing on the applications, while using the findings of this chapter as the theoretical foundations. These applications could include the modeling of human-robot collaboration tasks, where the cooperative geometric primitives are used to implement geometrically consistent leader-follower paradigms that exploit the inherent nullspaces.

7. Probabilistic Geometric Primitives

This chapter introduces a probabilistic view on the geometric primitives. We introduce probabilistic geometric primitives into the optimization framework presented in the previous chapters.

Publication Note

The material presented in this chapter has not been published yet.

7.1. Introduction

In previous chapters, we explored various methodologies for modeling robotic manipulation tasks via the formulation as optimization problems. These approaches assumed idealized scenarios that included a perfectly calibrated perception system, precise geometric models of objects and environments, and deterministic control execution. While theoretical formulations are simplified by these assumptions, they rarely hold in real-world applications and assuming perfect perception is a critical limitation. The dynamic, unstructured environments that robots nowadays are expected to operate in, lead to inherently noisy perception pipelines. The contact dynamics are hard to model due to their complexity. The actuation of the robotic system might suffer from elasticity. All of these unknown factors lead to uncertainties in the modeling of the manipulation tasks. Consequently, the perceived geometric primitives, e.g. points, lines, planes, are often inaccurate. Subsequent transformations that are required for pose estimation and trajectory planning also become inaccurate, since the errors propagate. If these uncertainties are not considered, they could impact the performance, safety, and reliability of the robotic system.

In particular, applications that require physical interaction with the environment have complex physical dynamics and precise modeling is difficult. Contacts that can occur along the whole body of robots, will lead to unpredictable contact forces. Predictions are further complicated by environmental factors like variations in object stiffness or surface friction. In these scenarios, compliance and delicate force modulation becomes paramount, since small errors in contact force or geometry can lead to task failure, mechanical damage, or unsafe interactions. Rather than implementing compliance mechanically via passive elastic elements, it can be achieved algorithmically via adaptive control strategies.

This chapter addresses these challenges by integrating a probabilistic representation of the geometric primitives into the optimization and control framework. Often, persisting uncertainties are modeled as probabilistic distributions, such as Gaussian covariances. Therefore, we use geometric algebra to propagate Gaussian distributions through geometric transformations in order to integrate them into the control objectives. The probabilistic modeling of geometric primitives enables the inclusion of robustness metrics into the optimization framework by adapting the control strategy based on the covariance. To this end, we extend these optimization objectives from the previous chapters to account for covariance-shaped cost functions.

This chapter is organized as follows, in Section 7.2 we present the mathematical background on propagating Gaussian covariances through the products of geometric algebra, we then present our derivation of covariance-aware constraints for optimization problems in Section 7.3, and show simulation results in Section 7.4.

7.2. Background: Probabilistic Geometric Primitives

A general multivector $X \in \mathbb{G}_{4,1}$ can be written as

$$X = \sum_{k=1}^{32} x_k \boldsymbol{e}_k, \tag{7.1}$$

where $\boldsymbol{e}_k$ are the blades of the algebraic basis of $\mathbb{G}_{4,1}$ and x_k are the components of the parameter vector $\boldsymbol{x} \in \mathbb{R}^{32}$. We define the mapping from $\mathbb{G}_{4,1}$ to $\mathbb{R}^{32}$ as the bijective function $\mathcal{E} : \mathbb{G}_{4,1} \to \mathbb{R}^{32}$ and its inverse as $\mathcal{E}^{-1} : \mathbb{R}^{32} \to \mathbb{G}_{4,1}$. At this point, we would like to point out that the geometric primitives, as subspaces of the algebra, are generally sparse, with known zero values. Therefore, the parameter vectors, covariance matrices and product tensors can be reduced to the known subspace containing the geometric primitive, which is generally much smaller than $\mathbb{R}^{32}$.

Since points are the basic building blocks of CGA, we start by explaining how an uncertain conformal point can be found from an uncertain Euclidean point that is expressed by a Gaussian distribution parameterized by its mean $\boldsymbol{\mu}_{\boldsymbol{x}}$ and covariance $\boldsymbol{\Sigma}_{\boldsymbol{x}}$. Recall that a Euclidean point can be embedded in the conformal space using Equation (2.20). Hence, we find the mean conformal point as

$$P_{\boldsymbol{\mu}} = P(\boldsymbol{\mu}_{\boldsymbol{x}}). \tag{7.2}$$

For embedding the covariance matrix into conformal space, we follow the approach presented in [179] and derive the Jacobi matrix of the embedding function

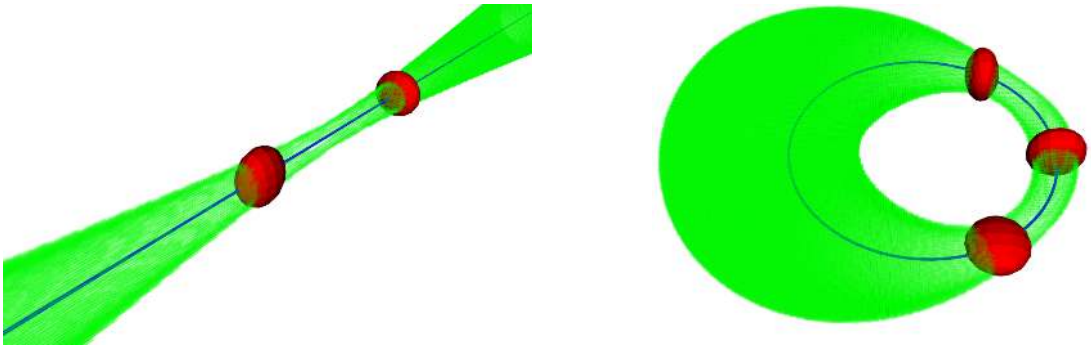
$$\boldsymbol{J}_P(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{x}} P(\boldsymbol{x}) = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ x_1 & x_2 & x_3 \end{bmatrix}. \tag{7.3}$$

The covariance matrix in conformal space can then be found as

$$\boldsymbol{\Sigma}_P = \boldsymbol{J}_P \boldsymbol{\Sigma}_{\boldsymbol{x}} \boldsymbol{J}_P^\top. \tag{7.4}$$

Due to the quadratic term in the embedding function from Equation (2.20) that corresponds to the basis blade $\boldsymbol{e}_\infty$, the conformal covariance matrix $\boldsymbol{\Sigma}_P$ is only an approximation of the covariance. This approximation lies in the tangent space of the cone that is formed by the embedding to conformal points. In general, for geometric primitives constructed using the point at infinity $\boldsymbol{e}_\infty$, the covariance matrix becomes exact, since the squared term vanishes due to $\boldsymbol{e}_\infty^2 = 0$. This is the case for flat primitives, such as

homogeneous points, lines and planes. For the round primitives, i.e. point pairs, circles and spheres, the covariance matrix remains an approximation that is tangential to the subspace of those primitives.



(a) Uncertain line resulting in a cylindrical shape.

(b) Uncertain circle resulting in a toroidal shape.

Figure 7.1.: Uncertain geometric primitives. The red ellipsoids show the uncertain points from which the geometric primitives are constructed. We then show the resulting mean in blue and covariance in green.

Starting from points as the basic building block for constructing more complex geometric primitives, we show here the covariance propagation as it was derived in [179]. Notably, since geometric algebra is a certain kind of tensor algebra, the products can be expressed using tensor operations. Here, we denote the bilinear functions for the inner, outer and geometric products as $\mathrm{i}(X_1, X_2)$, $\mathrm{o}(X_1, X_2)$, and $\mathrm{g}(X_1, X_2)$, respectively. The third-order tensors encoding these products are then denoted as $\mathrm{I}_{ijk}, \mathrm{O}_{ijk}, \mathrm{G}_{ijk} \in \mathbb{R}^{32\times32\times32}$. The derivation is the same for all products. Hence, we show it in terms of the abstract function $\gamma(\boldsymbol{x}_1, \boldsymbol{x}_2)$ with an associated tensor Γ_{ijk} that can stand for either the inner, outer or geometric product. Now, given two arbitrary multivectors $X_1, X_2 \in \mathbb{G}_{4,1}$, and recalling the summation of Equation (7.1), we can write the products as a bilinear function

$$\gamma(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sum_k x_{1,i} x_{2,j} \Gamma_{ijk} \boldsymbol{e}_k, \tag{7.5}$$

where tensor contraction, i.e. the summation over the indices i, j, is implied.

From Equation (7.5), we can find a left and a right Jacobian matrix as

$$\begin{aligned} \boldsymbol{\Gamma}_L(\boldsymbol{x}_2) &= \frac{\partial}{\partial \boldsymbol{x}_1} \boldsymbol{\gamma}(\boldsymbol{x}_1, \boldsymbol{x}_2), \\ \boldsymbol{\Gamma}_R(\boldsymbol{x}_1) &= \frac{\partial}{\partial \boldsymbol{x}_2} \boldsymbol{\gamma}(\boldsymbol{x}_1, \boldsymbol{x}_2), \end{aligned} \tag{7.6}$$

such that

$$\boldsymbol{x}_3 = \boldsymbol{\Gamma}_L(\boldsymbol{x}_2)\boldsymbol{x}_1 = \boldsymbol{\Gamma}_R(\boldsymbol{x}_1)\boldsymbol{x}_2, \tag{7.7}$$

where $\boldsymbol{x}_3$ is the parameter vector of the resulting multivector $X_3 = \mathcal{E}^{-1}(\boldsymbol{x}_3)$. Given the covariance matrices $\boldsymbol{\Sigma}_1$ and $\boldsymbol{\Sigma}_2$ for the parameter vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, we can now compute the covariance matrix $\boldsymbol{\Sigma}_3$ associated with the parameter vector $\boldsymbol{x}_3$, using the left and right Jacobian matrices

$$\boldsymbol{\Sigma}_3 = \boldsymbol{\Gamma}_L(\boldsymbol{x}_2)\boldsymbol{\Sigma}_1\boldsymbol{\Gamma}_L^\top(\boldsymbol{x}_2) + \boldsymbol{\Gamma}_R(\boldsymbol{x}_1)\boldsymbol{\Sigma}_2\boldsymbol{\Gamma}_R^\top(\boldsymbol{x}_1). \tag{7.8}$$

Following this approach, we can construct uncertain geometric primitives from conformal points with covariance matrices using the outer product. We show in Figure 7.1 various geometric primitives that are constructed from uncertain points. Note that, despite the complex shapes, the displayed covariances are fully defined by a single matrix.

7.3. Method

In the previous chapters, we have defined manipulation tasks using geometric algebra as optimization problems. In this chapter, we augment these optimization problem by the precision matrix $\boldsymbol{\Sigma}^{-1}$ found from the covariance matrix $\boldsymbol{\Sigma}$ of geometric primitives. Here, $\boldsymbol{\Sigma}$ is the chosen metric for calculating the norm. The precision matrix naturally represents a (cross-)weighting of the different components of a vector valued cost function. Given an arbitrary multivector valued function $F(X_c(\boldsymbol{q}), X_d)$ this optimization problem can then be written as

$$\min \left\| F(X_c(\boldsymbol{q}), X_d) \right\|_\Sigma. \tag{7.9}$$

The function $F(X_c(\boldsymbol{q}), X_d)$ can either represent the objective function using the outer product as shown in Chapter 4, or the logarithm of the similarity transformation between the geometric primitives $X_c(\boldsymbol{q})$ and X_d. In the following subsections we explain the derivation of the corresponding covariances.

7.3.1. Outer Product Covariance

In Equation (4.5) we briefly showed the formulation of an optimization problem for reaching geometric primitives using the outer product that we proposed in [148]. The covariance matrices of geometric primitives can be incorporated into this framework in the form of precision matrices. Using $\boldsymbol{e}(\boldsymbol{q}) = \mathcal{E}(X \wedge P(\boldsymbol{q}))$, we can reformulate it as

$$d(\boldsymbol{q}) = \boldsymbol{e}^\top(\boldsymbol{q})\boldsymbol{\Sigma}_E^{-1}(\boldsymbol{q})\boldsymbol{e}(\boldsymbol{q}), \tag{7.10}$$

where the covariance matrix $\boldsymbol{\Sigma}_E(\boldsymbol{q})$ is computed as

$$\boldsymbol{\Sigma}_E(\boldsymbol{q}) = \boldsymbol{\Lambda}_L(\boldsymbol{q})\boldsymbol{\Sigma}_X\boldsymbol{\Lambda}_L^\top(\boldsymbol{q}) + \boldsymbol{\Lambda}_R\boldsymbol{\Sigma}_P\boldsymbol{\Lambda}_R^\top, \tag{7.11}$$

where $\boldsymbol{\Sigma}_X$ is the covariance of the target geometric primitive and $\boldsymbol{\Sigma}_P$ that of the end-effector point. The function $d(\boldsymbol{q})$ is in some sense a distance function to the geometric primitives. In Figure 7.2 we show corresponding contour plots for different geometric primitives with and without covariance matrices. Here, we assume that the point $P(\boldsymbol{x})$ has a zero covariance matrix to simplify the visualization. For real applications, the point could also have a non-zero covariance matrix. Furthermore, we only showed the distance function of a conformal point to other geometric primitives for brevity and expressiveness. Since the covariance propagation works for all bilinear functions, other combinations of geometric primitives are possible.

To use this distance function for control, we derive the gradient of the distance function $d(\boldsymbol{q})$ from Equation (7.10). Note that we present the derivation w.r.t. $\boldsymbol{p}(\boldsymbol{q}) = \mathcal{E}(P(\boldsymbol{q}))$, since using the chain rule we find

$$\boldsymbol{\nabla}_{\boldsymbol{q}} d(\boldsymbol{q}) = \frac{\partial}{\partial \boldsymbol{p}} d(\boldsymbol{q}) \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{q}} = \boldsymbol{\mathcal{J}}_P^\top(\boldsymbol{q}) \boldsymbol{\nabla}_{\boldsymbol{p}} d(\boldsymbol{q}), \tag{7.12}$$

where $\boldsymbol{\mathcal{J}}_P(\boldsymbol{q})$ is the Jacobian of the end-effector point. We drop the dependence on $\boldsymbol{q}$ from the notation for this paragraph for increased conciseness and readability.

We separate the problem of deriving the gradient $\boldsymbol{\nabla}_{\boldsymbol{p}}$ into two parts. First, we want to recall that the error vector can be expressed using $\boldsymbol{p}$ and the right Jacobian matrix of the outer product, i.e.

$$\boldsymbol{e} = \boldsymbol{\Lambda}_R \boldsymbol{p}. \tag{7.13}$$

We can therefore rewrite Equation (7.10) as

$$d = \boldsymbol{p}^\top \boldsymbol{\Lambda}_R^\top \boldsymbol{\Sigma}_E^{-1} \boldsymbol{\Lambda}_R \boldsymbol{p}. \tag{7.14}$$

Neglecting that $\boldsymbol{\Sigma}_E^{-1}$ also depends on $\boldsymbol{p}$, we can find the first part of the gradient as

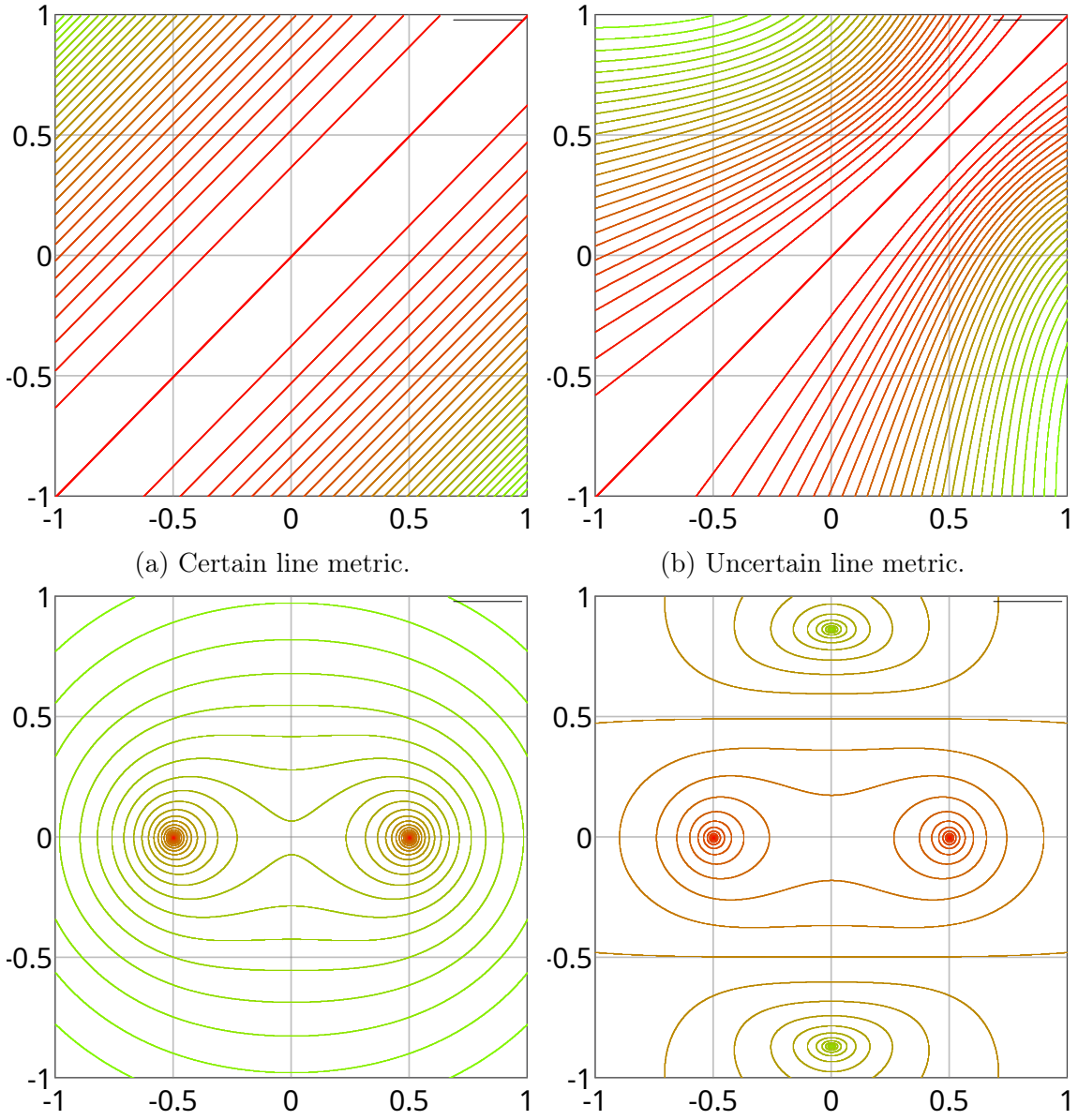
$$\boldsymbol{g}_1 = 2\boldsymbol{\Lambda}_R^\top \boldsymbol{\Sigma}_E^{-1} \boldsymbol{\Lambda}_R \boldsymbol{p}. \tag{7.15}$$

The second part of deriving the gradient requires finding the derivative of $\boldsymbol{\Sigma}_E^{-1}$ w.r.t. $\boldsymbol{p}$. We start by deriving the derivative of the covariance matrix $\boldsymbol{\Sigma}_E$ from Equation (7.11). This derivative is the third-order tensor $\hat{\boldsymbol{\Sigma}}_E$ and it is easy to see that the matrices for the partial derivatives can be found as

$$\hat{\boldsymbol{\Sigma}}_{E,j} = \frac{\partial}{\partial p_j} \boldsymbol{\Sigma}_E = \boldsymbol{\Lambda}_{ik} \boldsymbol{\Sigma}_X \boldsymbol{\Lambda}_L^\top + \boldsymbol{\Lambda}_L \boldsymbol{\Sigma}_X \boldsymbol{\Lambda}_{ik}^\top, \tag{7.16}$$

where $\boldsymbol{\Lambda}_{ik}$ is the partial derivative matrix of the left Jacobian

$$\boldsymbol{\Lambda}_{ik} = \frac{\partial}{\partial p_j} \boldsymbol{\Lambda}_L, \tag{7.17}$$



(a) Certain line metric.

(b) Uncertain line metric.

(c) Certain circle metric.

(d) Uncertain circle metric. The two green poles here correspond to local maxima.

Figure 7.2.: Comparison of the distance function for certain and uncertain geometric primitives. All figures are displayed in the $\boldsymbol{e}_1\boldsymbol{e}_2$-plane. The line is passing through the Euclidean points $\boldsymbol{x}_1 = 0$ and $\boldsymbol{x}_2 = \boldsymbol{e}_1 + \boldsymbol{e}_2$. The circle is defined by the three points $\boldsymbol{x}_1 = -0.5\boldsymbol{e}_1$, $\boldsymbol{x}_2 = 0.5\boldsymbol{e}_1$ and $\boldsymbol{x}_3 = 0.5\boldsymbol{e}_3$. Note that the third point is out of plane. The covariance matrix of all points is set to be the identity matrix. The scale between the level sets is logarithmic.

which corresponds to the ik matrix slices of the outer product tensor Λ_{ijk}. With Equation

(7.16), we then find the partial derivative of the inverse covariance matrix to be

$$\frac{\partial}{\partial p_j}\hat{\boldsymbol{\Sigma}}_E^{-1} = -\boldsymbol{p}^\top \boldsymbol{\Lambda}_R^\top \boldsymbol{\Sigma}_E^{-1} \hat{\boldsymbol{\Sigma}}_{E,j} \boldsymbol{\Sigma}_E^{-1} \boldsymbol{\Lambda}_R \boldsymbol{p}, \tag{7.18}$$

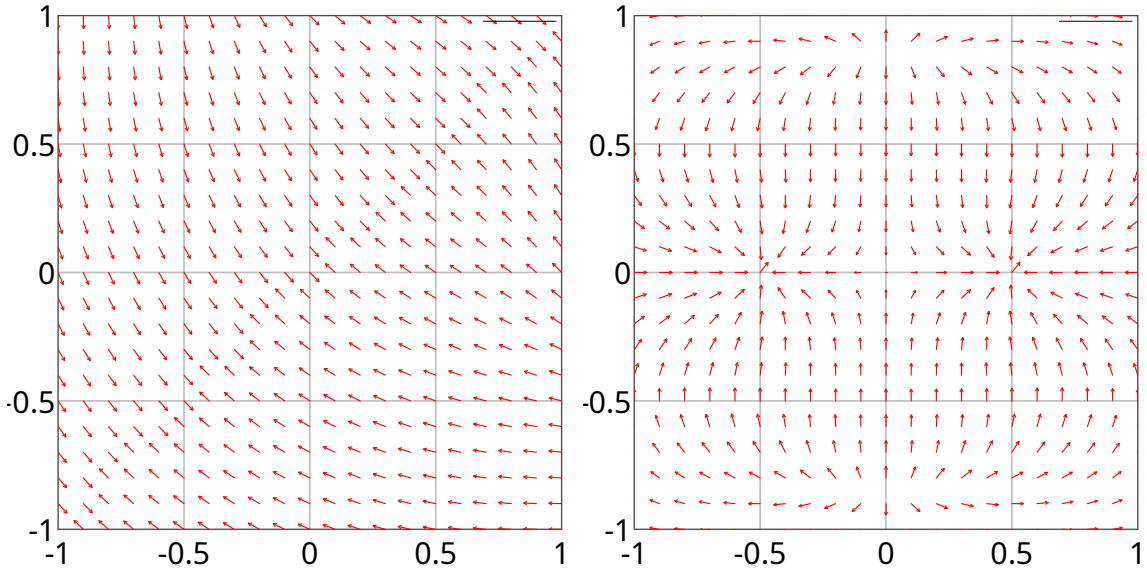
where we used the basic identity for matrix inverses from matrix calculus $\frac{\partial}{\partial x}\boldsymbol{X}^{-1}(x) = -\boldsymbol{X}^{-1}(x)\frac{\partial}{\partial x}\boldsymbol{X}(x)\boldsymbol{X}^{-1}(x)$ as it can be found in [225]. Now the second part of the gradient can be written as

$$\boldsymbol{g}_2 = \sum_j \frac{\partial}{\partial p_j}\hat{\boldsymbol{\Sigma}}_E^{-1}\boldsymbol{e}_j. \tag{7.19}$$

By the product rule of differentiation and plugging the results from Equations (7.15) and (7.19) in Equation (7.12) we finally find the gradient $\boldsymbol{\nabla}_{\boldsymbol{x}} d(\boldsymbol{x})$ to be

$$\begin{aligned} \boldsymbol{\nabla}_{\boldsymbol{x}} d(\boldsymbol{x}) &= \boldsymbol{J}_P^\top (\boldsymbol{g}_1 + \boldsymbol{g}_2) \\ &= \boldsymbol{J}_P^\top \Big(2\boldsymbol{\Lambda}_R^\top \boldsymbol{\Sigma}_E^{-1} \boldsymbol{\Lambda}_R \boldsymbol{p} \\ &\quad - \sum_j \boldsymbol{p}^\top \boldsymbol{\Lambda}_R^\top \boldsymbol{\Sigma}_E^{-\top} \hat{\boldsymbol{\Sigma}}_{E,j} \boldsymbol{\Sigma}_E^{-1} \boldsymbol{\Lambda}_R \boldsymbol{p} \boldsymbol{e}_j \Big). \end{aligned} \tag{7.20}$$

We calculate the gradient field for the uncertain geometric primitives from Figure 7.2 and display it in Figure 7.3.



(a) Gradient of the uncertain line metric. (b) Gradient of the uncertain circle metric.

Figure 7.3.: Gradients towards uncertain geometric primitives. The line and circle were constructed using the same uncertain points as in Figure 7.2. The resulting gradients are displayed using normalized vectors for a better visualization.

7.3.2. Probabilistic Similarity Transformations

In this section, we present how the cooperative control based on similarity transformations can be extended to include probabilistic versions of both the cooperative and desired geometric primitives. To this end, we derive the covariance matrix associated with the similarity transformation between the geometric primitives. We can insert this covariance matrix into Equation (7.9). Then, rewriting this Equation using $\boldsymbol{b}_Z(\boldsymbol{q}) = \mathcal{E}(B_Z(\boldsymbol{q}))$ and $\boldsymbol{\Sigma}_B(\boldsymbol{q})$ yields

$$\min \boldsymbol{b}_Z^\top(\boldsymbol{q})\boldsymbol{\Sigma}_B^{-1}(\boldsymbol{q})\boldsymbol{b}_Z(\boldsymbol{q}). \tag{7.21}$$

In order to compute the bivector covariance that we require for the controller, we use the Jacobian of the logarithmic map $\boldsymbol{J}_{\mathcal{Z}\to\mathbb{B}}\big(V_{Z,cd}(\boldsymbol{q})\big)$ to evaluate it from the similarity transformation covariance $\boldsymbol{\Sigma}_Z(\boldsymbol{q})$ as

$$\boldsymbol{\Sigma}_B(\boldsymbol{q}) = \boldsymbol{J}_{\mathcal{Z}\to\mathbb{B}}\big(V_{Z,cd}(\boldsymbol{q})\big)\boldsymbol{\Sigma}_Z(\boldsymbol{q})\boldsymbol{J}_{\mathcal{Z}\to\mathbb{B}}^\top\big(V_{Z,cd}(\boldsymbol{q})\big). \tag{7.22}$$

Recalling that $X_d = ZX_c\widetilde{Z}$, we directly write the expression for $\boldsymbol{\Sigma}_d$ in terms of the covariances $\boldsymbol{\Sigma}_c$ and $\boldsymbol{\Sigma}_Z$ and the left and right Jacobians of the geometric product, $\boldsymbol{G}_L$ and $\boldsymbol{G}_R$

$$\begin{aligned}\boldsymbol{\Sigma}_d =& \mathbf{G}_L(\widetilde{Z})\mathbf{G}_L(X_c)\boldsymbol{\Sigma}_Z\mathbf{G}_L^\top(X_c)\mathbf{G}_L^\top(\widetilde{Z})\\ &+ \mathbf{G}_L(\widetilde{Z})\mathbf{G}_R(Z)\boldsymbol{\Sigma}_c\mathbf{G}_R^\top(Z)\mathbf{G}_L^\top(\widetilde{Z})\\ &+ \mathbf{G}_R(ZX_c)\boldsymbol{\Sigma}_Z\mathbf{G}_R^\top(ZX_c)\\ &+ \mathbf{G}_L(\widetilde{Z})\mathbf{G}_L(X_c)\boldsymbol{\Sigma}_Z\mathbf{G}_R^\top(ZX_c)\\ &+ \mathbf{G}_R(ZX_c)\boldsymbol{\Sigma}_Z\mathbf{G}_L^\top\mathbf{G}_L^\top(X_c)(\widetilde{Z}),\end{aligned} \tag{7.23}$$

where the expressions $\mathbf{G}_L(\widetilde{Z})\mathbf{G}_L(X_c)\boldsymbol{\Sigma}_Z\mathbf{G}_R^\top(ZX_c)$ and $\mathbf{G}_R(ZX_c)\boldsymbol{\Sigma}_Z\mathbf{G}_L^\top\mathbf{G}_L^\top(X_c)(\widetilde{Z})$ are the cross-covariances, since in the geometric product between (ZX_c) and $\widetilde{Z}$ both terms depend on the covariance matrix $\boldsymbol{\Sigma}_Z$ making them statistically not independent. We can rewrite Equation (7.23) as

$$\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{X}\boldsymbol{A}^\top + \boldsymbol{B}\boldsymbol{X}\boldsymbol{B}^\top + \boldsymbol{A}\boldsymbol{X}\boldsymbol{B}^\top + \boldsymbol{B}\boldsymbol{X}\boldsymbol{A}^\top, \tag{7.24}$$

where we substituted recurring expressions using the matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{X}$, and $\boldsymbol{Y}$ that

are defined as

$$\begin{aligned}\boldsymbol{A} &= \mathbf{G}_L(\widetilde{Z})\mathbf{G}_L(X_c),\\ \boldsymbol{B} &= \mathbf{G}_R(ZX_c),\\ \boldsymbol{X} &= \boldsymbol{\Sigma}_Z,\\ \boldsymbol{Y} &= \boldsymbol{\Sigma}_d - \mathbf{G}_L(\widetilde{Z})\mathbf{G}_R(Z)\boldsymbol{\Sigma}_c\mathbf{G}_R^\top(Z)\mathbf{G}_L^\top(\widetilde{Z}).\end{aligned} \tag{7.25}$$

By further specifying a matrix $\boldsymbol{C}$ as

$$\boldsymbol{C} = \boldsymbol{A}\otimes\boldsymbol{A} + \boldsymbol{B}\otimes\boldsymbol{B} + \boldsymbol{A}\otimes\boldsymbol{B} + \boldsymbol{B}\otimes\boldsymbol{A}, \tag{7.26}$$

where $\otimes$ denotes the Kronecker product, we find $\boldsymbol{\Sigma}_Z(\boldsymbol{q})$ as the solution of the linear system

$$\boldsymbol{\Sigma}_Z(\boldsymbol{q}) = \boldsymbol{X} = \mathtt{vec}^{-1}\big(\boldsymbol{C}^{-1}\mathtt{vec}(\boldsymbol{Y})\big), \tag{7.27}$$

where the operator `vec` specifies stacking the columns of a matrix $\boldsymbol{X}\in\mathbb{R}^{m\times n}$ to a vector $\boldsymbol{x}\in\mathbb{R}^{mn\times 1}$.

7.4. Experiments

7.4.1. Simulation Experiments

First, to verify our approach, we present results from simulation experiments. We show different scenarios, with various different robots and describe the behavior of the different control strategies, both using certain and uncertain geometric primitives, and give insights about the influence of the tuning parameters.

Planar Single-Arm Reaching a Line

The setup for this experiment is a single robot manipulator with three degrees of freedom that is restricted to two dimensions for simplifying the visualization. The objective is to reach an uncertain line in space using another line at the end-effector as reference. We show this in Figure 7.4.1. In the more general case, the uncertain line might correspond to an uncertain plane that is estimated from the camera input. The idea of this experiment is to show the influence of the stiffness matrix adaption using the bivector covariance of the transformation as described in Section 7.3.2. To this end, we modify the stiffness matrix to introduce a mixing coefficient k_m that causes a trade-off between nominal and adaptive behavior, i.e.

$$\boldsymbol{\mathcal{K}} = k\Big((1-k_m)\boldsymbol{I} + k_m\frac{1}{\max(\mathrm{diag}(\boldsymbol{\Sigma}_B^{-1}))}\boldsymbol{\Sigma}_B^{-1}\Big) \tag{7.28}$$

The resulting likelihood is depicted in Figure 7.5 for different values of k_m. The plot clearly shows that increasing the influence of the covariance matrix changes the convergence behavior.

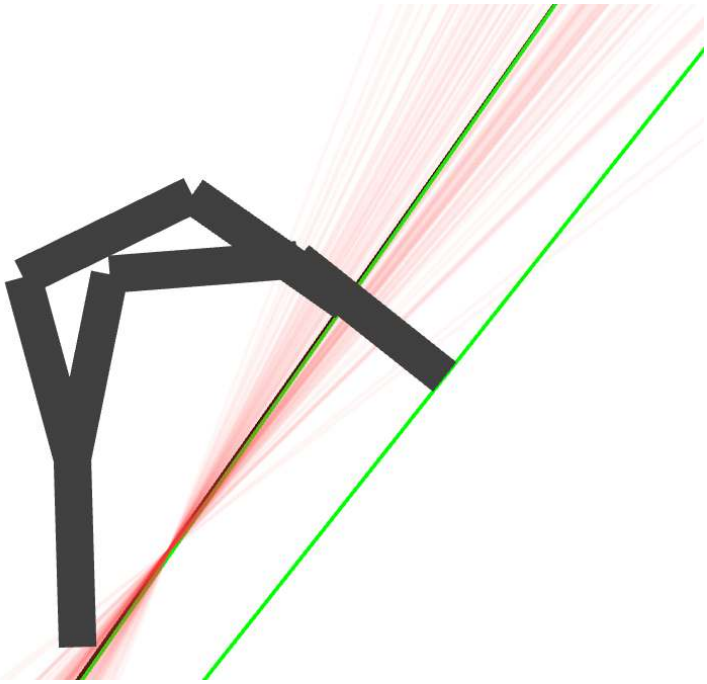


Figure 7.4.: A planar manipulator reaching a line with uncertainty. The light red lines are samples from the line distribution to show the variance.

Planar Single-Arm Reaching a Circle

In this experiment, we again use the same planar manipulator with three degrees of freedom. Here, the objective is to reach an uncertain circle using the gradient that we derived in Section 7.3.1. This uncertain circle could also correspond to a sphere that was fitted locally to a curved surface, which provides a better approximation than a tangent plane if the curvature is non-zero. We compare the trajectories of the end-effector for reaching a certain and an uncertain circle. We show this experiment in Figure 7.6. It can be seen that the trajectory for reaching the circle changes depending on whether the covariance is considered or not.

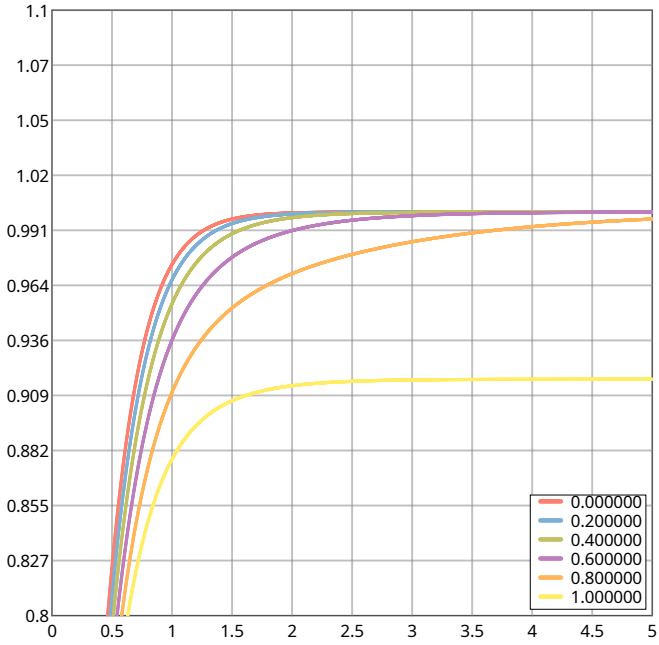


Figure 7.5.: Likelihood for of the current end-effector line for different values of k_m.

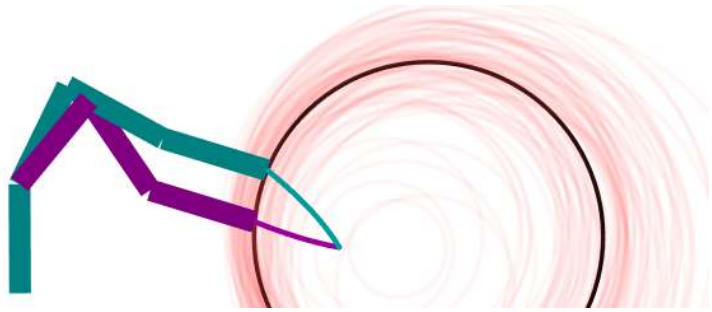


Figure 7.6.: A planar manipulator reaching a circle compared to reaching a circle with uncertainty. The light red circles are samples from the circle distribution to show the variance.

7.5. Discussion

7.5.1. Learning from Demonstration

In this work, we provided a probabilistic view on the geometric primitives and integrated them into the control formulations of the previous Chapters 4 and 6. In general, the covariance of the geometric primitives can represent the uncertainty coming from modeling or perception errors. However, another view on this covariance could also be the representation of the variability of the geometric primitives. This variability could come from task demonstrations, i.e. it could build a bridge to learning from demonstration techniques. In the simplest case, target geometric primitives could be demonstrated by

the necessary number of points. Each of these demonstrated points would be captured by a Gaussian distribution, and propagated to the covariance of the target geometric primitive.

Part III.

Practical Applications

8. Implementation

This chapter presents the implementation details as well as a tutorial of the *gafro* library, an efficient C++ library targeting robotics applications using geometric algebra, that was developed for this thesis. *gafro* implements geometric algebra for robotics and provides fundamental algorithms. While there are already efficient open-source implementations of geometric algebra available, none of them is targeted at robotics applications. The *gafro* library addresses this shortcoming. The library focuses on using conformal geometric algebra, which makes various geometric primitives as well as rigid body transformations available for computation. It implements various algorithms for calculating the kinematics and dynamics of such systems as well as objectives for optimization problems. The software stack is completed by Python bindings in *pygafro* and a ROS interface in *gafro_ros*.

Publication Note

The material presented in this chapter is adapted from the following publication:

> Tobias Löw, Philip Abbet, and Sylvain Calinon. "Gafro: Geometric Algebra for Robotics". In: *IEEE Robotics & Automation Magazine* (2024). DOI: 10.1109/MRA.2024.3433109

Philip Abbet helped in the implementation of the python bindings.

Website

Videos and supplemental material are available at:
`https://geometric-algebra.tobiloew.ch/gafro/`

Source Code

Source codes related to this chapter are available at:
`https://gitlab.com/gafro/`

8.1. Introduction

Although geometric algebra has great potential for modeling, learning and control in robotics, it has not been widely adopted in robotics research. One reason for this is the lack of easy-to-use libraries for robotics applications, while at the same time tools based on matrix algebra are very mature and readily available. We aim to change that by providing a ready-to-use geometric algebra library for robotics that can be used with the most popular programming frameworks, namely C++, Python and ROS. In our *gafro* library we provide an implementation of the GA variant of algorithms to compute the kinematics and dynamics of robots. These algorithms are well studied and have been implemented in various software frameworks. It is important to point out that geometric algebra can be used to compute these important quantities that are classically computed using matrix algebra, while also offering a richer toolset, i.e. it also includes tools for geometric reasoning that matrix algebra does not have. These tools include the construction of geometric primitives that are covariant under motion. The primitives can directly be used for projections, reflections and intersections. All operations do not depend on the choice of an origin, i.e. they are coordinate-free definitions. We demonstrate in this chapter how these geometric primitives can be used in robotics to facilitate and unify the formulation of control laws and optimization problems.

In this chapter, we want to explain the implementation details of our geometric algebra library *gafro* and show how to use it for common robotics problems such as inverse kinematics and optimal control. We compare *gafro* with existing libraries for geometric algebra and robot modeling. These libraries can be seen in Table 8.1. Our aim is to make geometric algebra more accessible for robotics research by providing this ready-to-use library.

This chapter is organized as follows: in Section 8.2 we explain the implementation details of the algebra, in Section 8.3 we compare *gafro* to other GA and robot modeling libraries and finally in Section 8.4 we demonstrate various applications and give a tutorial on how to use the library. The documentation and the links to all repositories can be found on our website `https://geometric-algebra.tobiloew.ch/gafro`.

8.2. The *gafro* Library

In this section we will explain in detail our implementation of Conformal Geometric Algebra (CGA). The aspects that are highlighted are the implementation of a general multivector and the expressions that are acting on it. In this section we explain the programming interfaces that *gafro* offers. The main library is written in C++ for which we provide Python bindings called *pygafro* as well as the ROS package *gafro_ros*. All mentioned repositories can be found at `https://gitlab.com/gafro`. An overview of the software stack can be found in Table 8.2.

Table 8.1.: Comparison of different libraries.

(a) Overview of other geometric algebra libraries.

Garamon	[39]	a generator of C++ libraries dedicated to geometric algebra
GATL	[78]	C++ library for Euclidean, homogeneous/projective, Minkowski/spacetime, conformal, and arbitrary geometric algebras using template meta-programming
Versor	[57]	(fast) generic C++ library for geometric algebras
GAL	[173]	C++17 expression compiler and engine for computing with geometric algebra
Gaigen	[82]	code generator for geometric algebra
Gaalet	[189]	C++ library for evaluation of geometric algebra expressions offering comfortable implementation and reasonable speed by using expression templates and meta-programming techniques
Gaalop	[103]	software to optimize geometric algebra files
TbGAL	[201]	C++/Python library for Euclidean, homogeneous/projective, Minkowski/spacetime, conformal, and arbitrary geometric algebras representing blades (and versors) in their decomposed state to scale to scale high dimensions

(b) Overview of other libraries for robot modeling.

DQ Robotics	[5]	library for robot modeling and control based on dual quaternion algebra
Pinocchio	[50]	state-of-the-art rigid body algorithms for poly-articulated systems
Raisim	[106]	multi-body physics engine for robotics and AI
KDL	[200]	application independent framework for modeling and computation of kinematic chains
Mujoco	[210]	physics engine for model-based optimization
RBDL	[77]	highly efficient code for both forward and inverse dynamics for kinematic chains and branched models

8.2.1. Design Goals and Implementation Details

We had several design goals in mind when designing the library and additionally wanted to cover several points that were proposed in [29] as a wishlist for geometric algebra implementations. Since it is targeted at robotics applications including robot learning,

Table 8.2.: Overview of the *gafro* software stack.

gafro	core C++20 library
pygafro	Python bindings
gafro_ros	interface to ROS and URDF
gafro_benchmarks	robot kinematics/dynamics benchmarks
gafro_examples	various code examples
gafro_robot_descriptions	classes defining different robot models

control and optimization, we wanted to ensure fast and efficient computation. To this end, the core implementation of *gafro* is done in C++20 and relies heavily on templates, which also serve the additional purpose of alleviating the effect of numerical imprecision that is known to occur in geometric algebra implementations by only evaluating elements of the resulting multivectors of expressions that are known to be non-zero. We further exploit that sparsity of the multivectors by only storing the data blades that are non-zero by the structure of the objects. We have designed the library in an object-oriented way, so the classes also reflect the mathematical inheritance relationships. Furthermore, all classes, i.e. all specialized multivectors, are instantiated as different types, which allows them to be distinguished at compile time for type-safety and to have persistent storage. These specialized classes also enable the computation with partial multivectors, i.e. the library exploits the fact that the most commonly-used multivectors are sparse and only use certain subspaces of the algebra. Mathematical operations are implemented as expression templates, which lets us determine the type of resulting multivectors at compile time. The implementation via expression templates allows the allocation of memory only if the expression is evaluated and also enables partial evaluations. We handle the type explosion of binary operators by automatically evaluating partial expressions when the full expressions get too complex. 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 [1], which is de facto the standard tool in robotics, to implement the sparse parameter vector of the multivectors.

In terms of using the library, we wanted to provide an accessible interface and ensure seamless integration with existing software. Geometric algebra is currently not well known in robotics, hence it can be daunting having to simultaneously learn about the algebra and the software implementation. Therefore, *gafro* provides the computation of the most important quantities for robot kinematics and dynamics with an interface that is close to similar robotics libraries. By basing our implementation on the *Eigen* library, these quantities can be returned in the familiar vector/matrix format. This essentially

[1] `https://eigen.tuxfamily.org`

makes *gafro* a drop-in replacement for other kinematics and dynamics libraries, without the need to know about all the details of geometric algebra at first. Afterwards, however, the geometric modeling of primitives, the singularity-free incidence computations, the uniform distance computation, the connections to differential geometry and the conformal group as well as the general structure of the algebra offer distinct advantages compared to other libraries.

8.2.2. General Multivector

The core element of computation in geometric algebra is the multivector. Hence, it is very important to think about the design choices when implementing its structure, as this will determine the memory usage and computational performance. The general structure of a multivector in CGA can be seen in Figure 2.1. It is composed of 32 basis blades, divided into grades zero to five. A general multivector would therefore be quite heavy in terms of memory and computation. The important structural aspect of CGA that facilitates the design process here is the sparsity of its representations and the fact that the structure of multivector expressions is known at compile time. Both of these properties mean that we can implement the data vector of a multivector by only storing its known non-zero elements. This is achieved by using a template that takes list of blade indices as input:

```
template <class T, int... blades>
class Multivector
{
    public:
        constexpr static int size = sizeof...(blades);
        {...}
    private:
        Eigen::Matrix<T, size, 1> data_;
};
```

The list of indices is then stored internally as a bitset that facilitates the comparison of the subspaces of two multivectors. A bitset is simply a list of 32 bits that are either 0 or 1, depending on whether the corresponding blade is present in the multivector or not. The memory that is allocated corresponds to the number of blade indices that is given to the template. It uses an `Eigen::Matrix` to store the data, which is exposed via an accessor function called `vector()`. This makes it possible to directly use the parameter vector of any multivector, which is useful for e.g. optimization solvers. The `Multivector` class and all its derived classes (including the expressions) have a method called `get` that is templated on the blade index. This method is fundamental to the design of the library, since all expressions use it for evaluation. Therefore, we add a

template constraint using the `requires` keyword of C++20 to ensure that multivector expressions only compile if they contain the requested blade index.

The underlying data type `T` is a template argument, which makes it possible to either use e.g. `float` or `double`, depending on the system architecture. Furthermore, it allows the usage of general purpose automatic differentiation libraries such as *autodiff*[2]. This helps when formulating optimization problems in geometric algebra using *gafro* since it facilitates the coding of complex objective functions and thus accelerates prototyping. Another relevant data type is the `torch::Tensor` class of the *libtorch* library[3], i.e. the C++ distribution of *PyTorch*. This effectively allows parallel computations with geometric algebra, which is important for various methods related to robot learning. While this combination can already be used, we are currently developing a library *gafro_torch* that facilitates the usage.

8.2.3. Algebraic Computations using Expression Templates

In order to do algebraic computations, there are several required operations on multivectors, which are implemented as expression templates. There are two types of expressions, unary and binary expressions, which are listed in Tables 8.3 and 8.4, respectively. We explained their mathematical meaning in Chapter 2 and thus focus here on their implementations.

Table 8.3.: Unary expressions that are implemented as member functions of the `Multivector` class.

member function	mathematical symbol
`reverse`	$\widetilde{X}$
`inverse`	X^{-1}
`dual`	X^{*}

The challenge in the implementation is the fact that the resulting multivectors only rarely have the same blades as the input operands. Given the structure of the algebra, however, the expression templates can determine the result type of the expression at compile time. Note that the expressions are evaluated in a lazy fashion, which means that the blades are evaluated on demand. This makes it possible to for example only evaluate a single blade of the resulting multivector.

```
template <class Derived, class Result>
class Expression
{
```

[2] `https://autodiff.github.io/`
[3] `https://pytorch.org/cppdocs/`

Table 8.4.: Binary expressions. The term operator here refers to the programming operators that are implemented for multivectors. Symbol means the mathematical symbol that is used in the equations. Note that usually the geometric product is written in equations without a symbol, e.g. AB.

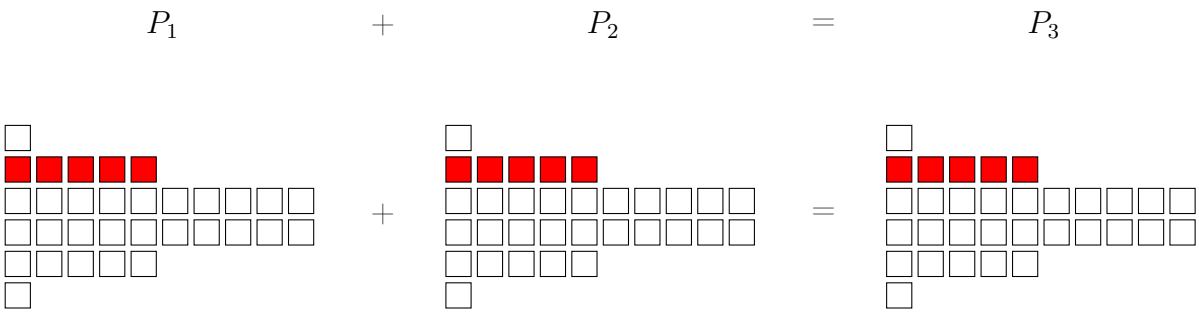
	operator	symbol
addition	+	+
subtraction	-	−
outer product	^	∧
inner product	|	·
geometric product	*	

```
    public:
        template <int blade>
        requires(Result::has(blade))
        typename Result::Vtype get() const
        {
            return static_cast<const Derived &>(*this).
               template get<blade>();
        }
};
```

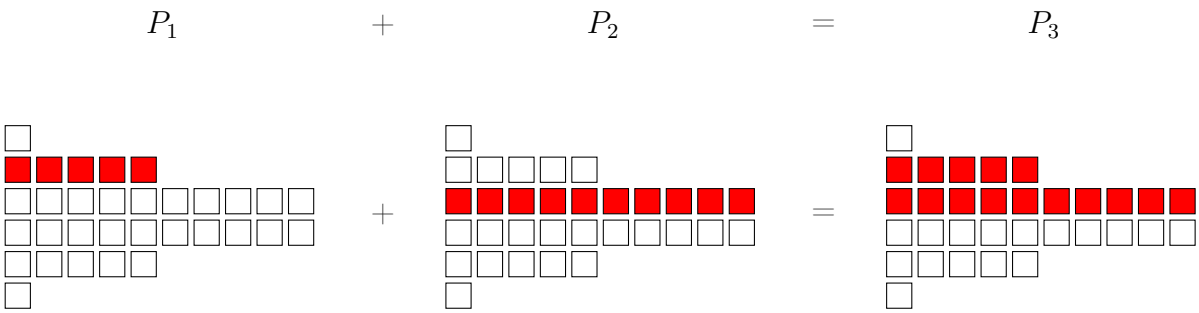
A first example of this can be seen in the `Sum` expression in Figure 8.1. The corresponding type evaluation class constructs the type of the resulting multivector at compile time. In the case of addition, this amounts to a simple bitwise OR operation comparing the bitsets of the input multivectors.

The inner and outer products work essentially in the same way and thus the corresponding expression both inherit from a base `Product` class, i.e. `InnerProduct` and `OuterProduct`. The `Product` class takes a class structure implementing the corresponding Cayley table as template argument. This Cayley table defines the resulting blades of a blade by blade product under the inner and outer product, respectively. Thus, in the case of CGA, it defines 1024 operations. In order to determine the type of the resulting multivector, we employ fold expressions that allow us to iterate over the blades of both input multivectors at compile time. In this loop, we obtain the resulting blade per pair of blades using the respective Cayley table and then assemble them into the resulting multivector again using OR operations. Figure 8.2 shows an example for each the inner and the outer product.

The geometric product class `GeometricProduct` also inherits from the base `Product` class and comes with its own Cayley table. So implementation-wise it is the same as the inner and outer products. The main difference is that two blades can result from a blade product, which causes the resulting multivector to potentially have both a lower



(a) The addition of two multivectors with the same blades results in another multivector with the same blades.



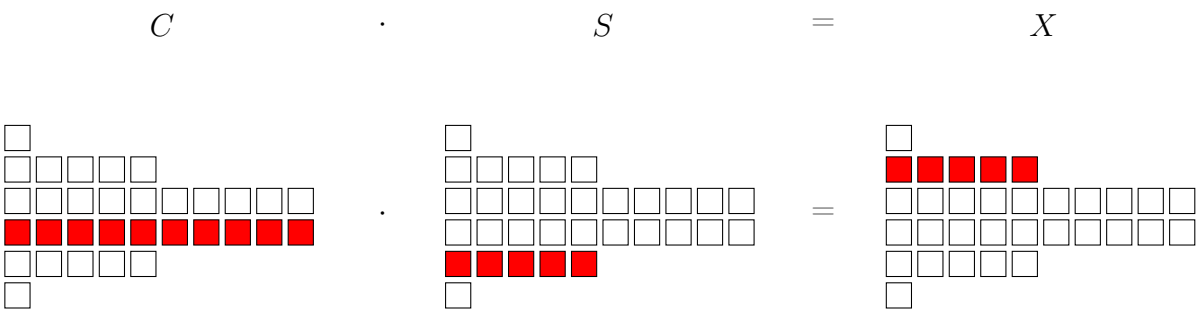
(b) The addition of two multivectors with the different blades results in a multivector with the blades of both input multivectors.

Figure 8.1.: Addition operation.

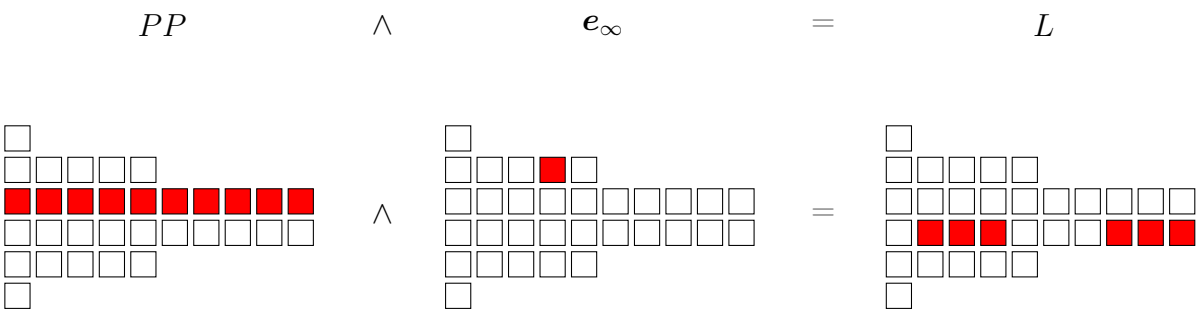
and a higher grade than the inputs, as can be seen in Figure 8.3a. Furthermore, we have products that are based on the geometric product such as the sandwich product, which is also treated as a binary expression and shown in Figure 8.3b.

8.2.4. Geometric Primitives

Since we know the subspaces of all the geometric primitives, we chose to implement them in an object-oriented way by inheriting from the base `Multivector` class. Hence, the available classes are `Vector`, `DirectionVector`, `TangentVector`, `Point`, `PointPair`, `Line`, `Circle`, `Plane` and `Sphere`. Their corresponding subspaces within the geometric algebra can be seen in Figure 2.2. Having the geometric primitives as explicit classes allows the implementation of commonly-used equations as members functions, which facilitates the usage. For example, the constructors of the geometric primitive classes implement the various ways they can be defined. The explicit classes are meant to facilitate the use and construction, but of course, using computation with base multivectors is also possible. This preserves the property of covariant computation within the algebra.



(a) The inner product is a grade-lowering operation, i.e. the resulting multivector will be of lower grade than the inputs. The example shows that the inner product of a circle C with a sphere S results in a point P.

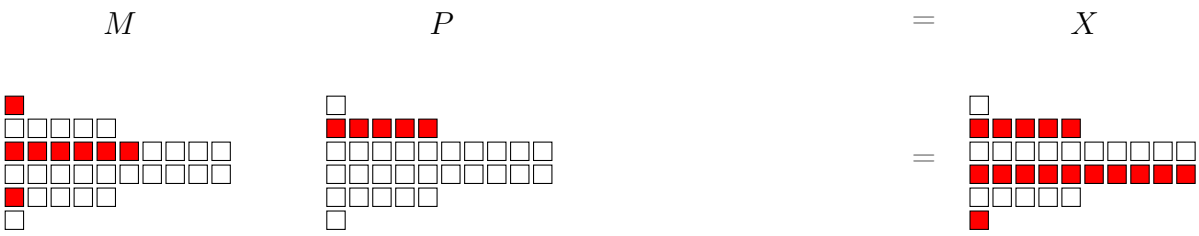


(b) The outer product is a grade-raising operation, i.e. the resulting multivector will be of higher grade than the inputs. The example shows that the outer product of a point pair PP and $\boldsymbol{e}_\infty$ results in a line L

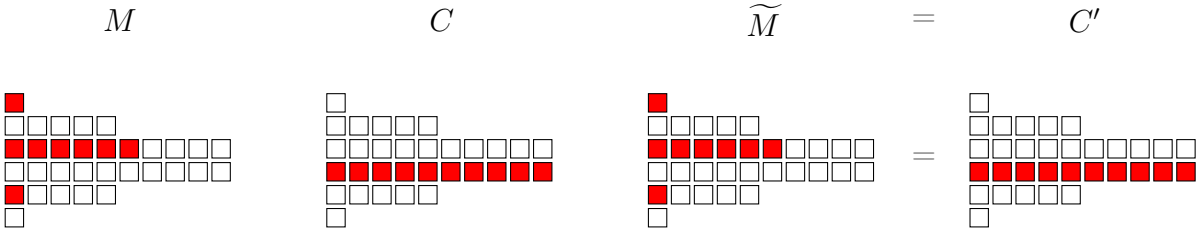
Figure 8.2.: The resulting multivector of the inner and outer product operations has a different grade than the inputs.

8.2.5. Rigid Body Transformations

The rigid body transformations that are currently available are implemented in the classes `Rotor`, `Translator`, `Motor` and `Dilator`. They all inherit from the base class `Versor`. Since all three classes are exponential mappings of bivectors they are accompanied by the expressions `Logarithm` and `Exponential`, respectively. The main method of the rigid body transformations is `apply`, which implements the sandwich product $X' = MX\widetilde{M}$ and ensures type safety. While X can technically be any multivector, the intended usage is with the geometric primitives that were presented in Section 8.2.4. Hence, in this context, type safety means that X' stays the same geometric primitive as X, e.g. a `Point` stays a `Point`. This ensures that the expression only evaluates blades that are part of the geometric primitive, which not only reduces the number of floating-point operations, but also deals with numerical imprecision in the computation that is known to occur in geometric algebra implementations.



(a) Using the geometric product results in both blades of lower and higher grade.



(b) The sandwich product is a grade-preserving operation. Numerical issues might lead to residuals in other blades, which we avoid by simply not evaluating them in the expressions. This is a schematic representation of Equation (2.17), where a motor transforms a circle.

Figure 8.3.: The geometric product is a combination of the inner and outer product.

8.2.6. Robot Modeling

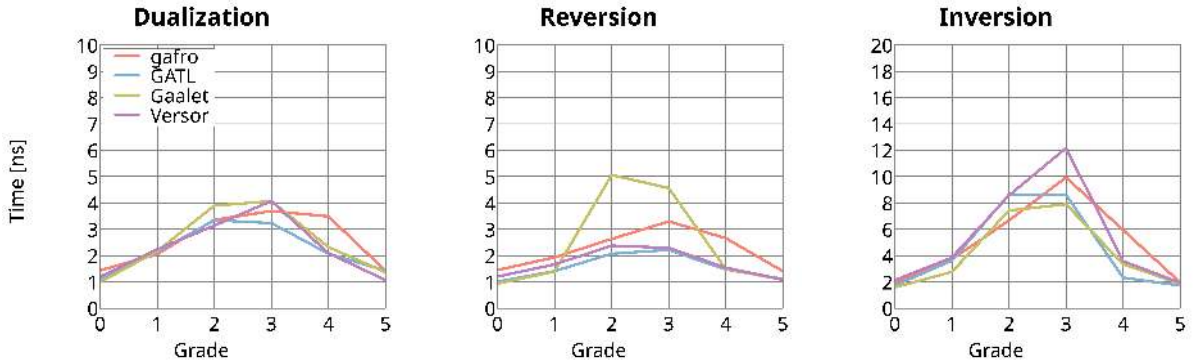
The previous sections introduced the features related to the underlying geometric algebra implementation of *gafro*. This section will now introduce the higher level features of the library related to robot modeling, which distinguish it from other geometric algebra libraries. The main aspects of robot modeling are the computation of the kinematics and dynamics of robotic systems, which is implemented in the base class called `System`. It contains member functions that compute the forward kinematics and forward/inverse dynamics using recursive algorithms. We also provide classes to model optimization problems for robotic systems, such as inverse kinematics. We will present more on this in Section 8.4 with concrete examples.

8.3. Comparison to Other Libraries

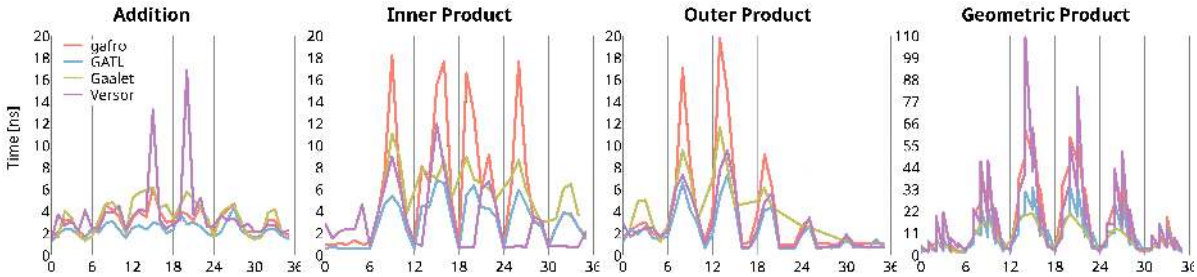
In this section, we compare *gafro* to other geometric algebra and robot kinematics/dynamics libraries. We first provide quantitative benchmark results and then give qualitative comparisons of what we believe to be advantages of *gafro* over other libraries.

8.3.1. Algebraic Operations Benchmarks

In order to compare the performance of our *gafro* library to other geometric algebra libraries we forked the *ga_benchmark*[4] repository in order to integrate *gafro*. Our fork can be found at `https://github.com/loewt/ga-benchmark`.



(a) Benchmarks of unary algebraic operations.



(b) Benchmarks of binary algebraic operations.

Figure 8.4.: Benchmarks of different geometric algebra libraries. All operations are computed using conformal geometric algebra. Some entries for *Gaalet* are missing due to segmentation faults during the execution.

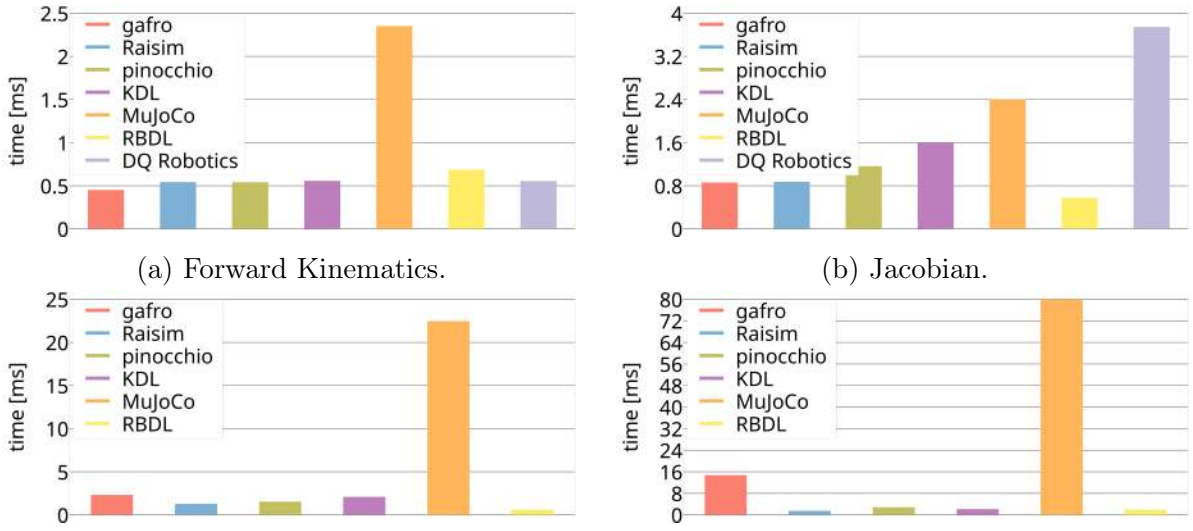
We omitted *TbGAL* and *Garamon* from the plots of the benchmark results, since they are by far the slowest libraries. The benchmarks show that *gafro* can compete in terms of performance with *GATL* and *Versor*, which were previously reported to be the fastest GA libraries.

8.3.2. Robotics Algorithms Benchmarks

Since this library implements robot kinematics and dynamics algorithms, we are comparing and benchmarking *gafro* against several libraries that are commonly used in robotics applications. The current benchmarking results on our system can be found in Figure 8.5. These libraries include Raisim [106], Pinocchio [50] and KDL [200]. As can be seen, *gafro* is very competitive when it comes to the computation of the kinematics of a robotic

[4] `https://github.com/ga-developers/ga-benchmark`

system. These advantages come from the fact that motors in geometric algebra are a more compact representation and require fewer arithmetic operations than transformation matrices. Note that previous publications have already shown the advantages that dual quaternions have over homogeneous transformation matrices [62] 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 [62] a 30%-40% improvement in performance of dual quaternions compared to homogeneous transformation matrices was reported, which is similar to our findings for motors. The computation of the dynamics, however, especially the forward dynamics, is still slower at this point. This is because at this stage we were prioritizing the research aspect of the algorithms, since they needed to first be derived in CGA. This means that the forward dynamics present a novelty not only in the implementation, but also in the mathematical derivation. This lead to a naive implementation, which includes unnecessary copy operations that affect the performance negatively. The issue will be addressed and fixed in a future release of *gafro*, which should make the computation of the dynamics also competitive compared to the established libraries.



(a) Forward Kinematics.

(b) Jacobian.

(c) Inverse Dynamics.

(d) Forward Dynamics.

Figure 8.5.: Benchmarks of robotics algorithms. The benchmark was run on an AMD Ryzen 7 4800U CPU. All libraries were compiled using *gcc 13.1.1* with the compiler flags `-O3 -march=native`. The reference system is the Franka Robot.

8.3.3. Advantages of *gafro*

There have been various works that published implementations of geometric algebra [39, 78, 57, 173, 82, 189, 103, 201]. These libraries all have in common that they are

meant to be generic geometric algebra implementations focusing on the computational and mathematical aspects of the algebra itself. In contrast to that, our implementation is targeted specifically at robotics applications and thus not only implements the low-level algebraic computations but also features the computation of the kinematics and dynamics of serial manipulators as well as generic cost functions for modeling optimization problems. We will explain these cost functions in detail in Section 8.4.3. Here, we want to point out that these cost functions simultaneously present an advantage over other geometric algebra libraries and over other robot modeling libraries, since neither of them target the geometric modeling robotics problems. *gafro* can therefore be seen as bridging the gap between these libraries.

We further believe that the programming interface of *gafro* is a lot more approachable and easy to use than other geometric algebra libraries. One reason for this is that we provide explicit classes for the geometric primitives and by making use of the C++ constructors, they can be created and used without having to explicitly use the algebraic operations. These classes also allow us to implement commonly used operations of multivectors directly as member functions, such as the sandwich product of motors to transform geometric primitives.

8.4. Applications and Tutorial

In this section we provide some example applications of how the library can be used. For that purpose, we provide an accompanying repository *gafro_examples* that contains coding examples. Note that in the text we are always referencing the C++ files, but the same examples can also be found in Python in the corresponding folder. These examples use the same naming scheme.

8.4.1. Geometric Algebra

Since many potential users of *gafro* are likely to be unfamiliar with the concept of geometric algebra we are providing some examples on how to do computations using this algebra. In a first example, we are showing how to create different general multivectors and use them for algebraic computations. For this purpose we demonstrate how to compute the intersection of a sphere S and a plane E, which in this case results in a circle C. The construction of a plane requires three points and the one at infinity $\boldsymbol{e}_\infty$. The construction of a sphere requires four points. Hence, we first define seven P_i using their Euclidean coordinates.

```
gafro::Point<double> p1(x, y, z);
```

From these points we then calculate the plane and the sphere, where `Ei` corresponds to $\boldsymbol{e}_\infty$.

```
gafro::Plane<double> plane = p1 ^ p2 ^ p3 ^ gafro::Ei<double
    >(1.0);
gafro::Sphere<double> sphere = p4 ^ p5 ^ p6 ^ p7;
```

Note that here, we choose to construct the plane and the sphere by the outer product of points, according to their mathematical definition, which derives from Equation (2.21). They could, however, be equivalently created by passing the same points to the respective constructors, which we show in the coding examples in the online repository.

The circle primitive that is found from the intersection of a plane and a sphere, which is expressed mathematically as the meet operator in Equation (2.29). This Equation can directly be translated to code to find the circle.

```
gafro::Circle<double> circle = (plane.dual() ^ sphere.dual()
    ).dual();
```

Note that, this code will compile and can be executed successfully without runtime errors whether the plane and the sphere intersect or not. This geometric relationship can be determined from the resulting circle by inspecting the squared norm, which will be positive or negative, depending on the incidence relationship.

8.4.2. Robot Differential Kinematics

One of the targeted use cases of the *gafro* library is the modeling of robotic systems. In this section, we will show how to do that in practice by explaining the example of a differential kinematics controller that tracks a line in task space using an arbitrary reference line at the robot end-effector. We use in this example the class `FrankaEmikaRobot` and assume we have instantiated it as `panda`.

First, the forward kinematics, i.e. the pose of the end-effector of a kinematic chain given a certain joint configuration, are represented by the motors in geometric algebra. For a given joint configuration `q` the end-effector motor is found

```
gafro::Motor<double> ee_motor = panda.getEEMotor(q);
```

We implement the differential kinematics controller w.r.t to the robot end-effector frame. Hence we use the end-effector motor for transforming an object of type `gafro::Line` called `target_line` to this frame. We skip the creation of this line here, but note that it is similar to the creation of geometric primitives in the previous section. The transformation of the line using `ee_motor` is implemented as

```
gafro::Line<double> transformed_line = ee_motor.reverse() *
    target_line * ee_motor;
```

This effectively corresponds to the equation $L' = \widetilde{M}LM$, as opposed to $L' = ML\widetilde{M}$, which was shown previously in Equation (2.17). The difference is that here we use the inverse transform from the base frame to the end-effector frame.

Next, we find the twist, which moves the reference line to the transformed target line. This twist is found as the logarithmic mapping of the motor that transforms one line to the other. Mathematically this can be expressed as

$$\mathcal{V} = \log\left(\frac{1}{c}\left(1 + L_r L_t\right)\right), \tag{8.1}$$

where $\mathcal{V}$ is the resulting twist and c is a normalization constant. L_r and L_t are the reference and target line, respectively. We have implemented this as member function of the `Line` class, such that it can be called directly as

```
gafro::Twist<double> twist = transformed_line.getMotor(
    reference_line).log();
```

The last step is the computation of the joint velocities $\dot{\boldsymbol{q}}$ from the twist $\mathcal{V}$. This is achieved using the inverse of the end-effector frame Jacobian, which can be obtained by the member function `getEEFrameJacobian` of the `panda` robot. This function returns a *gafro* specific object, which can be transformed to an `Eigen::Matrix` using the `embed` method. The control law according to the equation $\dot{\boldsymbol{q}} = \boldsymbol{J}^{-1}\mathcal{V}$ can therefore be implemented as

```
Eigen::Vector<double,7> qdot = inverse(panda.
    getEEFrameJacobian(q).embed()) * twist.vector();
```

Here we use the `inverse` function as shorthand for the pseudoinverse of a 6×7 matrix.

The lines in this example can be chosen arbitrarily, which is a very appealing property, since it has two important consequences. First, the reference line at the end-effector constrains two axes of rotation, while allowing a rotation around the line. This line does not need to coincide with the axes of the end-effector frame. The axes are not even required to be known explicitly, the line is sufficient. This essentially avoids having to deal with coordinate frames when encoding the target. Moreover, the two lines are invariant under translations along them, i.e. moving a line in the direction it is pointing to does not change the line. In practice, this effectively means that the control law is completely compliant to disturbances along the superposed lines. These two properties are very hard to achieve using classical methods and require many coordinate frame changes and non-trivial precision matrices. This example shows that the definition of a control law using geometric primitives can be done entirely geometrically and resulting equations are very simple since they are also algebraic objects.

8.4.3. Optimization Problems with Geometric Primitives

Many problems in important domains of robotics, such as learning and control, can be cast as optimization problems. Hence, in this section we are providing an example on how *gafro* can be used for the uniform modeling of optimization problems using

geometric algebra. Here, we cast the optimization problem as an inverse kinematics problem for simplicity, so we are optimizing for the joint angle configuration in which the end-effector reaches a certain geometric primitive and we show how GA extends the cost function to be uniformly applicable across the different geometric primitives. The optimization problem can be formulated as

$$\boldsymbol{q}^* = \min_{\boldsymbol{q}} \frac{1}{2} \left\| E(\boldsymbol{q}) \right\|^2, \tag{8.2}$$

where $\boldsymbol{q}$ is the joint angle configuration and $E(\boldsymbol{q})$ is a multivector-valued residual. In *gafro* this formulation is implemented in the generic template class `SingleManipulatorTarget` and Equation (8.2) can be evaluated using the method `getValue`. The below code snippet of this shows that it has the template arguments `Tool` and `Target`, which are meant to be different geometric primitives.

```
template <class T, int dof, template <class Type> class Tool
   , template <class Type> class Target>
class SingleManipulatorTarget{...};
```

`Tool` is a geometric primitive at the end-effector of the robot arm, e.g. a point, and `Target` is a desired geometric primitive that should be reached by the end-effector, e.g. a line or a circle. The problem of reaching can be expressed as minimizing a distance measure between the two primitives. Mathematically, this distance measure can be expressed as a residual multivector stemming from the outer product, i.e.

$$E(\boldsymbol{q}) = X_d \wedge M(\boldsymbol{q}) X \widetilde{M}(\boldsymbol{q}), \tag{8.3}$$

where X corresponds to the `Tool` and X_d to the `Target`, the motor $M(\boldsymbol{q})$ is the end-effector motor at the current joint configuration $\boldsymbol{q}$, which transforms any geometric primitive to the end-effector, expressed w.r.t. the base frame. By definition, this outer product results in zero, if `Tool` has reached the `Target`. Its norm therefore corresponds to a distance measure that we want to minimize here. In the implementation this residual multivector from Equation (8.3) is obtained by calling the function `getResidual`, which can be seen in the code snippet below.

```
Eigen::Matrix<T, Result::size, 1> getResidual(const VectorX
   &q) const
{
    return Result(target_ ^ arm_.getEEMotor(q).apply(tool_))
       .vector();
}
```

The Jacobian of Equation (8.3) w.r.t. the joint configuration vector $\boldsymbol{q}$ is found by

applying the chain rule to the geometric product of the motor $M(\boldsymbol{q})$, i.e.

$$\boldsymbol{\mathcal{J}}^E(\boldsymbol{q}) = X_d \wedge \left(\boldsymbol{\mathcal{J}}^A(\boldsymbol{q}) X \widetilde{M}(\boldsymbol{q}) + M(\boldsymbol{q}) X \widetilde{\boldsymbol{\mathcal{J}}^A}(\boldsymbol{q})\right), \tag{8.4}$$

where $\boldsymbol{\mathcal{J}}^A(\boldsymbol{q})$ is the analytic Jacobian of the kinematic. The following code snippet shows the implementation of Equation (8.4). Both implementations closely follow the mathematical formulation.

```
Eigen::Matrix<T, Result::size, dof> getJacobian(const
    VectorX &x) const
{
    Motor<T> motor = arm_.getEEMotor(x);
    MultivectorMatrix<Motor<T>, 1, dof> jacobian_ee = arm_.
        getEEAnalyticJacobian(x);

    Eigen::Matrix<T, Result::size, dof> jacobian;

    for (unsigned i = 0; i < dof; ++i)
    {
        jacobian.col(i) = Result(target_ ^ (jacobian_ee[i] *
            tool_ * motor.reverse() + motor * tool_ *
            jacobian_ee[i].reverse())).vector();
    }

    return jacobian;
}
```

Note that both `getResidual` and `getJacboian` return a matrix of the *Eigen* library where the size is determined based on the combination of geometric primitives. More specifically the size can vary depending on the primitives, but due to the structure of the algebra and its implementation using expression templates, the size is determined at compile time. In practice, the actual sizes of the residual and Jacobian can be neglected, since for solving an optimization problem, we are actually interested in the gradient vector $\boldsymbol{g} \in \mathbb{R}^{N\times 1}$ and Hessian matrix $\boldsymbol{H} \in \mathbb{R}^{N\times N}$ of Equation (8.2) and their size is only determined by the number of degrees of freedom N of the robot and is therefore agnostic to the choice of geometric primitives.

Given the residual and the Jacobian, the optimization problem can easily be solved using for example a Gauss-Newton type algorithm. Both of these quantities can be accessed from the class via the method `getGradientAndHessian` which returns them in the form of matrices from the *Eigen* library. This choice fulfills one of the design goals of the library, i.e. the seamless integration with existing optimization solvers.

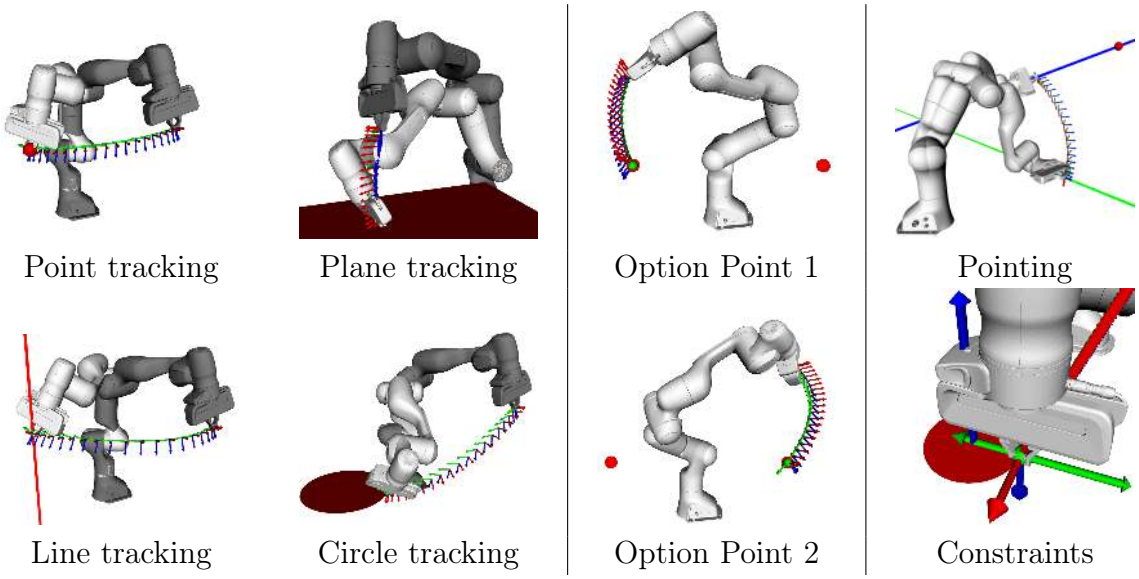


Figure 8.6.: RViz visualizations of the Franka robot reaching various geometric primitives. These visualizations were created using the tools from *gafro_ros*.

Hence, it is possible to use these geometric algebra computations in existing pipelines, without having to fundamentally rewrite existing software to accommodate the geometric algebra, which keeps the integration effort low. This example is also applicable across a wider range of applications. In previous work, we have shown the application of CGA to modeling manipulation tasks in an optimal control framework for model predictive control [148], which can of course be achieved using the same cost function.

We give several examples of this inverse kinematics problem in *gafro_examples*. The files are following the naming scheme *inverse_kinematics_PRIMITIVE1_PRIMITIVE2.cpp*. We visualize the results of optimization problems using various geometric primitives in Figure 8.6. We want to point out that the implementations only differ in the instantiation of the `SingleManipulatorTarget` template class, which shows the ability of geometric algebra to unify formulations and simplify their implementations.

8.5. Conclusion

In this chapter we presented the implementation details as well as some examples for our software stack around *gafro*, which is a C++ library that implements conformal geometric algebra for robotics. The software stack also includes Python bindings in *pygafro* as well as a ROS package in *gafro_ros*. Tutorial material and toy examples can be found in *gafro_examples*.

While showing comparable performance for the robot modeling, geometric algebra also offers an easy and intuitive way to model various geometric relationships as shown by

examples of intersecting geometric primitives, differential kinematics using line objects and optimization based inverse kinematics with different geometric primitives. The motors are a more general concept of transformations that can be directly applied to all geometric primitives within the algebra, alleviating the need to compute special adjoint operations. Combined with the fact that motors are a more compact representation of rigid body transformations that requires less operations, geometric algebra offers a very rich and appealing mathematical framework for robotics, without losing any of the existing tools that are offered by standard matrix algebra.

Our library *gafro* provides the standard algorithms for robot modeling and the computation of the kinematics and dynamics. It then augments them with concepts that are exclusive to geometric algebra, such as direct representations of geometric primitives and operations on them which are then used for the implementation of general optimization problems. In fact, by the design of the library, which exposes the parameter vectors using `Eigen`, these standard libraries could directly be replaced by *gafro* without having to use geometric algebra directly. Providing this library that makes geometric algebra easily accessible for robotics research should allow for a wider adoption and facilitate research on using this powerful framework for robotics.

9. Dual-Arm Admittance Control

We propose a task-space admittance controller for dual-arm robotic systems using conformal geometric algebra. The controller is a reinterpretation of a previous work using dual quaternion algebra. By introducing conformal geometric algebra, we aim to enhance the geometric expressiveness, which simplifies the modeling of various tasks and opens doors to more complex applications, such as the modeling of multiple points of contact in the robotic arm in a whole-body manipulation task. We first show the derivation of the controller for a single-arm robot, which is then extended to a dual-arm robot. The closed-loop system is therefore composed of an outer loop admittance controller that imposes the apparent impedance, and an inner loop that transforms the twist acceleration to a control input that is sent to the robot. Experiments executed on a setup with two LBR KUKA iiwa 14 R820 robots with a force/torque sensor in each end-effector show good performance of the proposed controller, for both single and dual-arm tasks, where the error decays in the absence of external wrenches. In the presence of external wrenches, the controller is able to adapt the robot's motion to keep the desired impedance.

Publication Note

The material presented in this chapter has not been published yet. It is adapted from the planned publication:

> Tobias Löw, Mariana de Paula Assis Fonseca, Vitalii Pruks, Graham Deacon, Jelizaveta Konstantinova, and Sylvain Calinon. "Dual-Arm Admittance Control Using Conformal Geometric Algebra". In: *in preparation* ()

Mariana de Paula Assis Fonseca and Vitali Pruks integrated the controller with the robot setup and performed the experiments. My contribution was the mathematical derivation of the controller and its implementation.

9.1. Introduction

Nowadays, robots can be found in a variety of environments, from the factory floor to the home. The various different tasks that robots are then expected to execute in these environments increasingly require the robots to interact with objects in a way that goes beyond traditional parallel jaw grippers or suction cups that are often featured on single arm manipulators. For instance, big objects may call for dual-arm manipulators for picking and placing. For those systems, the coordination between two arms is vital when grasping, lifting, and transport objects of varying shapes and fragility.

Moreover, when manipulating objects, the robots are subjects to contacts, and thus contact wrenches may appear. In order to have a safe interaction, it is essential that the robots interacts with the objects in a compliant manner, which can be imposed by controlling the apparent impedance of the system [43]. To prevent unnatural behavior, it is important to define a stiffness term that is consistent with the task geometry, for both position and orientation terms. With that in mind, Caccavale et al. proposed to use an energy-based impedance equation, and to use the imaginary part of the unit quaternion to represent the orientation displacement between the desired and current frames [42]. However, in this work, the authors propose a stiffness that is geometrically consistent only for infinitesimal displacements. Therefore, they extended the previous work to have a stiffness that is geometrically consistent for also finite displacements [43]. A reformulation of this controller using dual quaternion algebra showed an exponential decay of the error norm due to the linearity of the stiffness term, while also not suffering from the problem of topological obstruction [81].

Coordinating the motion of a dual-arm manipulator requires control strategies that enable collaborative behavior. To this end, the collaborative dual-task space (CDTS) was proposed [4]. This approach unifies the end-effector poses into an absolute and a relative poses that are expressed using dual quaternions. The CDTS was then used to formulate an admittance controller for a dual-arm mobile robot [79], which was then further improved by also introducing an adaptive admittance behavior [168]. Using conformal geometric algebra, the CDTS was extended by introducing the cooperative pointpair primitive, and integrated into an optimal control formulation and use inverse dynamics control to compute desired torque commands [147]. While this approach enabled a compliant behaviour of the manipulators, it did not take into account actively controlling desired contact wrenches.

In this work, we are employing conformal geometric algebra to formulate an admittance control strategy and show how it not only enables the integration of geometric primitives within the control formulation, but also allows for a natural extension to more than one manipulator.

9.2. Method

Given a desired motor M_d and a current motor $M_{\boldsymbol{q}}$, the desired apparent impedance of the system can be imposed on the displacement between M_d and $M_{\boldsymbol{q}}$ [42]. Therefore, we propose an admittance control law formulated using bivector B_e, which is the logarithm of the motor error $M_e = \widetilde{M_{\boldsymbol{q}}} M_d$.

9.2.1. Admittance Control

To impose a desired apparent impedance on the system, we can formulate the dynamics in bivector space as a mass-spring-damper system where the bivectors B_e, $\dot{B}_e$ and $\ddot{B}_e$ describe the error dynamics in bivector space

$$\boldsymbol{\mathcal{I}}\left[\ddot{B}_e\right] + \boldsymbol{\mathcal{D}}\left[\dot{B}_e\right] + \boldsymbol{\mathcal{K}}\left[B_e\right] = \mathcal{W}_{ext} - \mathcal{W}_d. \tag{9.1}$$

The tensors $\boldsymbol{\mathcal{I}}$, $\boldsymbol{\mathcal{D}}$ and $\boldsymbol{\mathcal{K}}$ are inertia, damping and stiffness, respectively. The bivectors B_e, $\dot{B}_e$, $\ddot{B}_e$ describe the error dynamics in bivector space. Assuming that we are tracking a static target, i.e. the desired velocity $\dot{B}_d$ and acceleration $\ddot{B}_d$ are zero, and defining the bivector velocity error as $\dot{B}_e = 2\left(\dot{B}_d - \dot{B}_{\boldsymbol{q}}\right)$, we can use (3.20) to replace $\dot{B}_e$ and $\ddot{B}_e$ using the current twist $\mathcal{V}$ and twist acceleration $\dot{\mathcal{V}}$.

Since admittance is dual to impedance, instead of solving for the wrench, we solve for the twist acceleration

$$\dot{\mathcal{V}} = \boldsymbol{\mathcal{I}}^{-1}\left[\mathcal{W}_{ext} - \mathcal{W}_d - \boldsymbol{\mathcal{D}}\left[\mathcal{V}\right] - \boldsymbol{\mathcal{K}}\left[B_e\right]\right]. \tag{9.2}$$

We can then find the joint accelerations from the twist acceleration using

$$\ddot{\boldsymbol{q}} = \boldsymbol{\mathcal{J}}^{G,-1}\left(\dot{\mathcal{V}} - \dot{\boldsymbol{\mathcal{J}}}^{G}\dot{\boldsymbol{q}}\right), \tag{9.3}$$

where $\boldsymbol{\mathcal{J}}^G$ and $\dot{\boldsymbol{\mathcal{J}}}^G$ are the geometric Jacobian and its time derivative. Note that, this control law is defined here without an explicit reference frame. In practice, a common choice is either the end-effector frame of the manipulator or target frame. For position or velocity control, the desired joint accelerations $\ddot{\boldsymbol{q}}_d$ can be integrated once or twice from the current robot state $(\boldsymbol{q}, \dot{\boldsymbol{q}})$ using an appropriate time step Δt to either get the joint velocity $\dot{\boldsymbol{q}}$ or position $\boldsymbol{q}$. If the manipulator that is being controlled allows torque control, the desired torque command $\boldsymbol{\tau}_d$ can be computed using inverse dynamics

$$\boldsymbol{\tau} = \boldsymbol{f}^{-1}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}). \tag{9.4}$$

9.2.2. Single Arm

In a first step we demonstrate how the admittance control law from the previous section can be used with a single arm manipulator. Given a desired target motor M_d and the current end-effector motor $M_{\boldsymbol{q}}$ we can find the error bivector as

$$B_e = -\log\left(\widetilde{M}_{\boldsymbol{q}} M_d\right). \tag{9.5}$$

Here, the error bivector B_e is expressed w.r.t. the current end-effector frame $M_{\boldsymbol{q}}$. The twist that is required for the damping term then becomes the end-effector twist $\mathcal{V}_{\boldsymbol{q},\dot{\boldsymbol{q}}}$ and can be computed either from (3.19) or (3.20).

9.2.3. Dual Arm

The CGA formulation of the cooperative dual task space uses two motors to describe the relationship of the two end-effectors, the relative motor M_r and the absolute one M_a. Based on these two motors, it is straightforward to define desired motors and obtain the respective error bivectors, $B_{e,r}$ and $B_{e,a}$, using (9.5). When controlling both absolute and relative motors at the same time, (9.2) can be extended such that we have individual expressions for the absolute components and the relative ones.

Using the admittance control law in the CDTS then requires the derivation of the geometric Jacobians of the relative and absolute motors. Using the relationship between the geometric and analytic Jacobians from (3.15), we find them as

$$\boldsymbol{\mathcal{J}}_r^G = -2\widetilde{M}_r \boldsymbol{\mathcal{J}}_r^A, \tag{9.6}$$

and

$$\boldsymbol{\mathcal{J}}_a^G = -2\widetilde{M}_a \boldsymbol{\mathcal{J}}_a^A, \tag{9.7}$$

respectively. Here, $\boldsymbol{\mathcal{J}}_r^A$ and $\boldsymbol{\mathcal{J}}_a^A$ are the analytic Jacobians of the CDTS that we derived in [147]. Consequently, the absolute and relative twists can be found as

$$\mathcal{V}_r = \boldsymbol{\mathcal{J}}_r^G \dot{\boldsymbol{q}} \quad \text{and} \quad \mathcal{V}_a = \boldsymbol{\mathcal{J}}_a^G \dot{\boldsymbol{q}}, \tag{9.8}$$

respectively, where $\dot{\boldsymbol{q}} = \begin{bmatrix}\dot{\boldsymbol{q}}_1 & \dot{\boldsymbol{q}}_2\end{bmatrix}^\top$.

Accounting for the two wrenches in the CDTS requires them to be correctly mapped to the relative and absolute frames. For the relative motor, we can find the relative wrench as

$$\mathcal{W}_r = \frac{1}{2}\widetilde{M}_2(\mathcal{W}_2 - \mathcal{W}_1)M_2, \tag{9.9}$$

where M_2 is the motor of the second arm, w.r.t. the origin frame, and $\mathcal{W}_1$ and $\mathcal{W}_2$ are the external wrenches acting on the end-effectors of robots 1 and 2, respectively, also

w.r.t. the origin coordinate system. We use M_2 here as the relative motor M_r in (5.1) is defined w.r.t. the second arm. The absolute wrench can be found as

$$\mathcal{W}_a = \widetilde{M}_a(\mathcal{W}_1 + \mathcal{W}_2)M_a, \tag{9.10}$$

where M_a is the absolute motor defined in (5.3).

9.2.4. Practical Considerations

Dissipative Term

When the robot is redundant with respect to the task, the joint velocities can be different from zero even if the system is in equilibrium. In this case, a dissipative term can be used to prevent the robot from moving after reaching the equilibrium. We opted to add a damping term to the joint velocity, in the null space:

$$\dot{\boldsymbol{q}}_{diss} = \dot{\boldsymbol{q}} + \left(\mathbb{I} - \boldsymbol{\mathcal{J}}^{A,-1}\boldsymbol{\mathcal{J}}^{A}\right)\left(-k_{diss}\dot{\boldsymbol{q}}\right). \tag{9.11}$$

where $\mathbb{I}$ is the identity matrix, $k_{diss} \in (0, \infty)$ and $\dot{\boldsymbol{q}}$ is the joint velocity obtained by integrating (9.3).

Unwinding Phenomenon

Since motors are an equivalent representation to dual quaternions and also double cover $\boldsymbol{SE}(3)$, they also suffer from what is known as the unwinding phenomenon. This is a topological issue that is caused by the fact that the motors M and $-M$ represent the same transformation, and it prevents global asymptotic stability, since there are two stable equilibria. In practice, this will cause unnecessary motion when the current end-effector motor $M_{\boldsymbol{q}}$ is topologically closer to $-M_d$ rather than M_d. There exist, however, already numerous solutions to address this problem for dual quaternions [81] and they can be seamlessly adapted for motors in our proposed CGA admittance control scheme. One possible solution is to check the condition $\|M - 1\|_2 \leq \|M + 1\|_2$ and then choose M or $-M$ such that it yields the same result as $M_{\boldsymbol{q}}$. For instance, we use the following unwinding logic in calculating a bivector B_e from a given motor $M_e = \widetilde{M}_{\boldsymbol{q}}M_d$:

$$B_e = \begin{cases} -\log M_e, & \text{if } \|M_e - 1\|_2 \leq \|M_e + 1\|_2, \\ -\log(-M_e), & \text{otherwise.} \end{cases} \tag{9.12}$$

9.2.5. Geometric Primitives

One advantage that geometric algebra offers over other frameworks is the direct representation of geometric primitives within the algebra. These primitives can therefore be

used in a uniform manner in order to generate the desired behaviors by deriving error bivectors based on their difference. In general, this is achieved by the formula

$$B_e = \log(M_e) = \log\left(\frac{1}{c}\left(1 + X_2 X_1\right)\right), \tag{9.13}$$

where X_1 and X_2 are two geometric primitives and c is a normalization constant that depends on the geometric primitives. More details can be found in [130]. The resulting motor M_e then represents the transformation that transforms X_1 to X_2. In [31], we have used this to align the end-effector z-axis with the surface normals. Here, we show that other geometric primitives are possible as well to be included in the admittance control law. The general idea is that X_1 is a geometric primitive that is attached to the robotic system. In that case, X_2 is a desired geometric primitive on the object where we want to have forceful interactions. By construction, the controller will be stiff when moving away from the geometric primitives and compliant when moving on them.

9.3. Experimental Results

To evaluate our proposed controller, experiments were run on a robotic system with two KUKA LBR iiwa 14 R820 robot manipulators equipped with an ATI Gamma Force/-Torque sensor at their wrists, and a plate grippers attached after the sensors. The setup is connected to a computer running Ubuntu 22.04, with 64GB of RAM and an Nvidia RTX 2080 TI. The controller is implemented in C++ in the gafro library[1], which is an open source library to use geometric algebra for robotics [145]. Since we use position control, the control input found in (9.3) is numerically integrated twice, using Newton's first-order approximation, to obtain the joint velocity $\dot{\boldsymbol{q}}$ and position $\boldsymbol{q}$. To prevent reaching the joints' maximum velocities, they were saturated at 0.2 rad/s. The experiments were run with a sampling time of 0.001 ms. In order to show the performance of the controller, we present two sets of experiments: single-arm and dual-arm experiments.

9.3.1. Single Arm Experiments

Two types of experiments were performed with the single arm: *hold pose* and *push* experiments. For all results reported in this chapter using single arm robot, the control law defined in (9.2) was used with inertia, damping and stiffness tensors chosen empirically as $\boldsymbol{I} = 1.5\mathbb{I}_{6\text{x}6}$, $\boldsymbol{D} = 300\mathbb{I}_{6\text{x}6}$, $\boldsymbol{K} = 80\mathbb{I}_{6\text{x}6}$, where $\mathbb{I}_{6\text{x}6}$ is a 6x6 identity matrix.

[1] *https://gitlab.com/gafro*

Hold Pose Experiment

Consider a scenario where the current motor $M_{\boldsymbol{q}}$ at the beginning of the experiment is equal to the desired one M_d. In the absence of an external wrench acting on the robot end-effector, the robot should remain in the desired pose. When an external wrench is applied, the robot should react to it, making $M_{\boldsymbol{q}}$ differ from M_d to impose the desired apparent impedance. After releasing the contact, the contact wrench vanishes and the end-effector should return to its original (and desired) pose. This experiment shows the performance of the controller in a single-arm scenario for both contact rich and free-motion tasks. We performed this experiment by setting the robot's end-effector in a horizontal configuration and attaching items of different weights to its plate grippers.

The experiments are shown in Fig. 9.2 and 9.4, where the first item weights 0.633kg and the second one 4.6kg. Since the second item is heavier than the first one, the wrench acting on the end-effector is higher, which leads to a higher difference between the desired pose and the actual one. The coefficients of the poses w.r.t. robot base and the corresponding wrenches acting on the gripper w.r.t. gripper are shown in Fig. 9.1 and 9.3. In both cases, the wrench is close to zero at the beginning of the experiment, when the robot is in free motion, and the gripper pose is equal to the desired one. After adding the object, the external wrench acting on the gripper increases due to the weight of the object, and then the robot starts to move compliantly to ensure the desired apparent impedance of the system. When the object is removed, the wrench returns to close to zero and the end-effector converges back to the desired pose. The moments when the weight is added and removed from the gripper are shown in the graphs. This experiment shows that the controller adapts the robot's motion to ensure the desired apparent impedance according to the wrench acting on it.

Push Experiment

The previous experiment showed that the controller leads to a compliant behavior under external wrenches, when the system is already at the desired pose. In this experiment, the robot actively moves to push an item that weighs 2.238kg by applying a wrench, as shown in Fig. 9.6. The experiment has three phases: *go downwards*, *push forward*, and *retract*. Only in the second phase, the robot is in contact with the object and experiences an external wrench. Figure 9.5 shows the current and desired poses for the experiment and the corresponding wrenches. The black vertical lines in the graphs indicate the phase changes. Therefore, the first part of the graphs represents the movement of the gripper going down, without any contact with the environment. In this phase, it is possible to see that the wrench values are close to zero, and the orientation of the gripper remains the same during the whole movement. The position in z-axis gets close to the desired one, indicating that the robot is moving as expected. In the second phase, the gripper is moving forward, in the x-axis, and it is in contact with the object it is pushing.

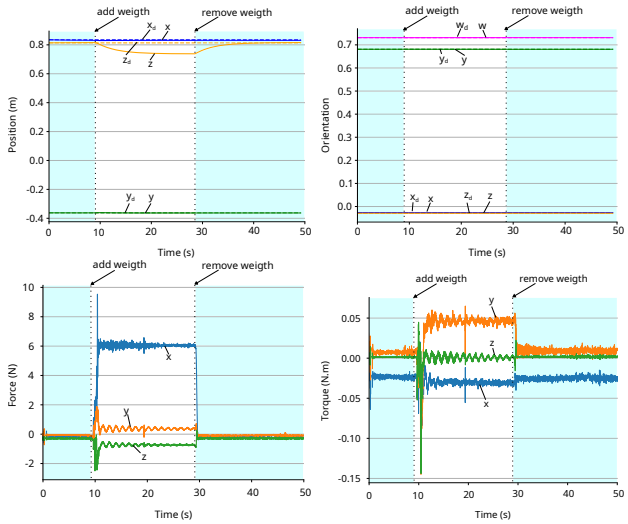


Figure 9.1.: Holding an object that weighs 0.633kg. *Top:* Pose. *Bottom:* Wrench. *Left:* Position/Force. *Right:* Orientation/Torque. These images were prepared by Ocado Technology.

Figure 9.2.: *Left:* the start configuration of the robot, showing the desired pose of the plate gripper. *Right:* the result of the movement after attaching the item with 0.633kg to the gripper. These images were prepared by Ocado Technology.

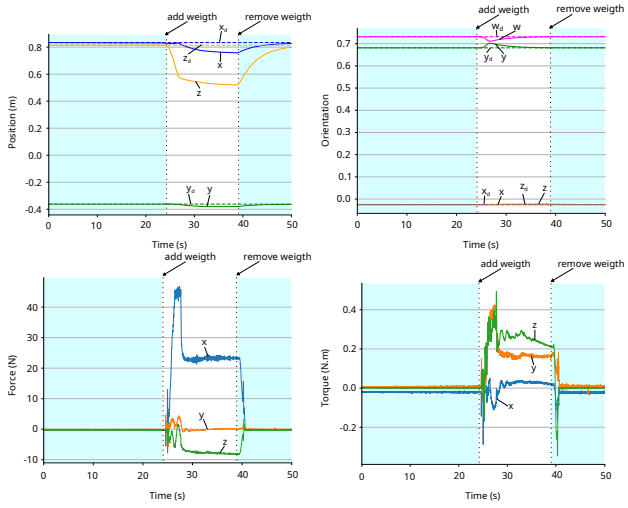


Figure 9.3.: Holding an object that weighs 4.6kg. *Top:* Pose. *Bottom:* Wrench. *Left:* Position/Force. *Right:* Orientation/Torque. These images were prepared by Ocado Technology.

Figure 9.4.: *Left:* the start configuration of the robot, showing the desired pose of the plate gripper. *Right:* the result of the movement after attaching the item with 4.6kg to the gripper. These images were prepared by Ocado Technology.

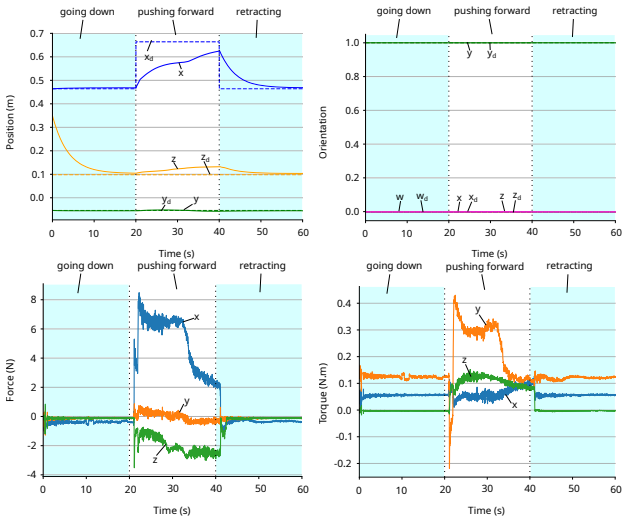


Figure 9.5.: Pushing an object weighing 2.238kg. *Top:* Pose. *Bottom:* Wrench. *Left:* Position/Force. *Right:* Orientation/Torque. These images were prepared by Ocado Technology.

Although there is a torque read, the values are still small, and not enough to change the orientation of the gripper. The force values are higher, especially in the x-axis. In this phase, the x-axis coefficient of the position changes to try to reach the desired value. Due to the force due to the contact with the object, the robot is compliant and cannot fully achieve the desired value, although it gets close to it. It is also possible to see that the z-axis diverges a couple of centimeters from the desired pose, due to the force in the z-axis. In the third phase, when the plate releases the contact with the object and returns to the previous pose, no external force is acting on the plate gripper anymore, and both position coefficients for x and z-axes reaches to the desired values.

9.3.2. Dual Arm Experiments

Consider the task of cooperative manipulation of an object. The two arms must pick an object with small to negligible mass, so the movement of the arms are not considerably

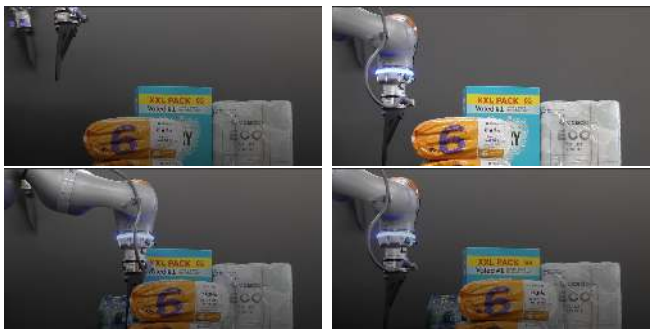


Figure 9.6.: Top left image shows the initial state of the experiment. Top right illustrates the plate gripper pose after going down. Bottom left shows the end of the pushing movement, where the gripper is in contact with the object. Bottom right shows the robot after coming back to the previous pose. These images were prepared by Ocado Technology.

affected by gravity effect due to the object. Here, the inertia and damping matrices were chosen empirically as $\boldsymbol{I} = 1.5\mathbb{I}_{6x6}$, $\boldsymbol{D} = 300\mathbb{I}_{6x6}$, respectively, for both absolute and relative poses. The relative stiffness was set as $\boldsymbol{K} = 100\mathbb{I}_{6x6}$, and the absolute as $\boldsymbol{K} = 200\mathbb{I}_{6x6}$. The object used in this experiment weighs 1.119kg.

The task is broken into different phases, as in Fig. 9.9:

Initial phase: The robot goes to an initial state, defined by an absolute and a relative pose.

Approach phase: The desired absolute pose is modified, such that the grippers go down, approaching the item. The relative pose remains the same.

Grasping phase: In order to grasp the item, the relative pose is changed, bringing grippers closer to each other. The absolute pose remains the same.

Lifting phase: The absolute pose is modified to lift the item, while the relative one is kept unchanged.

Transport phase: The robot moves the item sideways, which is achieved by a modification of the absolute pose.

Placing phase: The absolute pose is changed in order for the grippers to go down.

Releasing phase: The relative pose is changed, which increases the distance between the grippers to release the item. The absolute pose remains unchanged.

Final phase: The robots lift their grippers by changing the absolute pose to re-initialise the process.

In the *grasping phase*, a desired force of 5N is chosen for each arm in order to ensure a stable grasp. This desired force is kept until the *releasing phase*, when the desired force is set back to 0N. The desired absolute and relative wrenches are calculated using (9.10) and (9.9), respectively. The compensated wrenches acting in both arms are depicted in Fig. 9.8, and the absolute and relative poses in Fig. 9.7. In both the *initial phase* and

the *approach phase* there is no wrench acting on the arms to deviate the poses from the desired ones. In the *grasping phase*, when the arms approach the object, there is an internal wrench acting on the object when the plate grippers come into contact with it. Therefore, the controller imposes the desired apparent impedance to the system that prevents the arms to completely reach the desired pose. This can be observed in Fig. 9.7, where the x-axis of the relative position does not reach the desired value. This error is kept during all phases where the arms are grasping the object. In the *lifting phase*, the z-axis of the absolute pose gets closer to the desired one, but it is not able to reach it. This is due to the wrench acting on the arms due to gravity. This can be observed in Fig. 9.8, where the y-axis of the forces has a value close to 5N in each arm. [2] In the *transport phase*, the arms move sideways, and the there is no wrench influencing the movement in the y-axis. Therefore, the y component reaches the desired value. In the *placing phase*, the arms go down to place the object in a table. After releasing the item in *releasing phase*, no wrench is acting on the arms anymore, and then all components of the absolute pose achieve the desired values. The same is observed in the *final phase*. We can observe that there is an error in the x-axis of the relative position in the *releasing phase*. This is due to the timeout that we imposed, that was reached before the pose was fully reached. The error becomes zero in the next phase, when the movement continues. It is possible to see that the orientation is kept practically the same during the whole experiment. This is because the torque component observed in both arms were very small ($<$0.4N).

9.4. Conclusion

In this work, we presented a dual-arm admittance control framework leveraging CGA to enhance the manipulation capabilities of robotic systems. By utilizing CGA, we achieved a compact and intuitive representation of spatial transformations, enabling efficient computation and improved task modelling.

The proposed approach was validated through experiments, demonstrating its effectiveness in handling single and dual-arm coordination tasks.

Future work will focus on extending this framework to multi-arms/fingers and incorporating adaptive stiffness term so the controller can automatically adapt to different manipulated objects, with various weights and deformability. Furthermore, we will expand the controller to account for inaccuracies in both perception and the robot itself, such as tendon driven robots. Moreover, CGA can also be used to model multiple contact points, when using tactile sensors [88].

[2] According to (9.10), the absolute wrench (or external wrench) is the sum of the wrenches in each arm, transformed to be with respect to the absolute frame. Since the weight of the object is 1.119Kg, the absolute wrench is around 10.96N, which if divided equally between the arms, results in 5.48N on each arm.

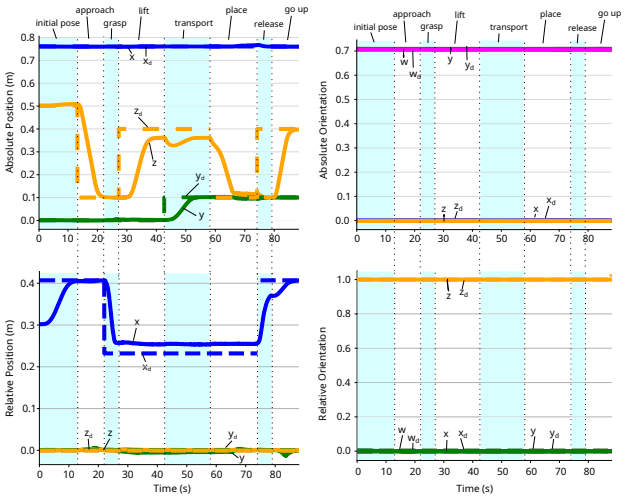


Figure 9.7.: Absolute and relative poses. *Left:* coefficients of the current and desired absolute (up) and relative (bottom) positions. *Right:* coefficients of the quaternion representing the current and desired absolute (up) and relative (bottom) orientation. These images were prepared by Ocado Technology.

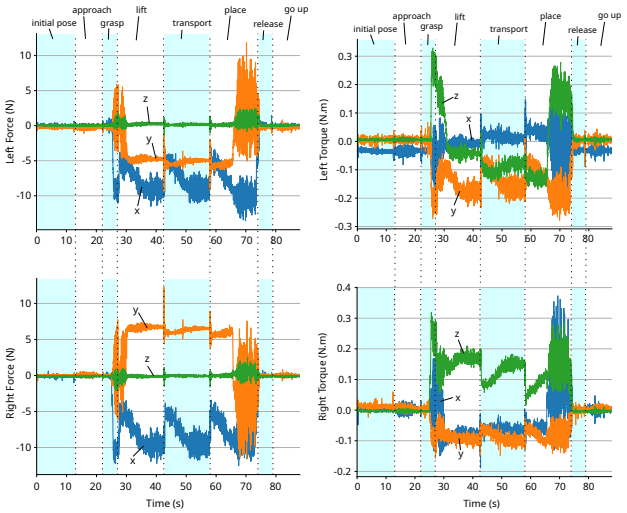


Figure 9.8.: External wrenches applied at each plate gripper, w.r.t. the grippers. *Left:* coefficients of the force. *Right:* coefficients of the torque. These images were prepared by Ocado Technology.

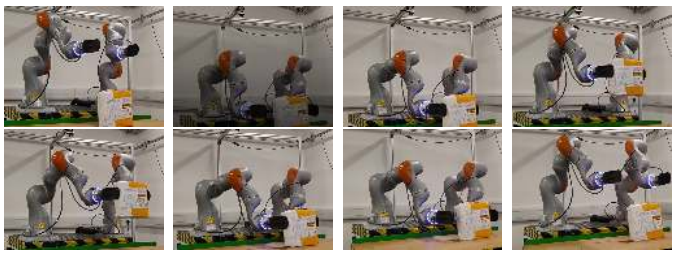


Figure 9.9.: Setup of the dual arm experiments. From left to right, the arm approaches the object, grasp it, lift it, transport it, place it, and then release it. These images were prepared by Ocado Technology.

10. Tactile Ergodic Control

In this chapter, we present a feedback control method for tactile coverage tasks, such as cleaning or surface inspection. Although, these tasks are challenging to plan due to the complexity of continuous physical interactions, the coverage target and progress can be effectively measured using a camera and encoded in a point cloud. We propose an ergodic coverage method that operates directly on point clouds, guiding the robot to spend more time on regions requiring more coverage. For robot control and contact behavior, we use geometric algebra to formulate a task-space impedance controller that tracks a line while simultaneously exerting a desired force along that line. We evaluate the performance of our method in kinematic simulations and demonstrate its applicability in real-world experiments on kitchenware.

Publication Note

The material presented in this chapter is adapted from the following publication:

> Cem Bilaloglu, Tobias Löw, and Sylvain Calinon. "Tactile Ergodic Coverage on Curved Surfaces". In: *IEEE Transactions on Robotics* (2025)

Cem Bilaloglu formulated the ergodic coverage using diffusion on point-clouds. My part was the formulation of the robot controller using geometric algebra. We both performed the experiments and the writing.

Website

Videos and supplemental material are available at:
`https://sites.google.com/view/tactile-ergodic-control/`

10.1. Introduction

The long-term vision of robotics is to assist humans with daily tasks. The success of robot vacuum cleaners and lawnmowers as consumer products highlights the potential of robotic assistance for common household chores [44]. These tasks involve covering a region in a repetitive and exhaustive manner. Currently, these robots are limited to relatively large, planar surfaces, and even navigating slopes remains challenging [214, 205]. Other daily tasks, such as washing dishes or grocery items, present even greater challenges due to the complex physical interactions with intricate, curved surfaces. Similarly, numerous coverage tasks on curved surfaces arise in industrial and medical applications. In industrial settings, such tasks include surface operations that remove material, such as sanding [154], polishing [118, 9] or deburring [174] as well as surface inspection tasks leveraging contact [180]. In medical settings, similar applications range from mechanical palpation [**ayvaliUsingBayesianOptimization2016a**, 217] and ultrasound imaging [113, 83] to massage [151, 124] and bed bathing [53, 152]. Last but not least, datasets combining the tactile properties of objects with their shape and visual appearance remain scarce and expensive to collect, as they rely on teleoperation [140]. Thus, tactile coverage is critical for automating the collection of tactile datasets that complement visual ones. The problem definitions of this diverse range of settings and applications can be distilled into two key requirements: (i) tactile interactions with a possibly non-planar surface and (ii) a continuous trajectory of contact points covering a region of interest on the surface. Accordingly, this chapter addresses the overarching problem of tactile coverage on curved surfaces.

Tactile interaction tasks, by definition, involve multiple contact interactions with the environment, making these systems notoriously difficult to control [15]. While humans solve these tasks effortlessly, they remain extremely challenging for robots. For instance, when cleaning an object, achieving adequate coverage depends on recognizing dirt, understanding the object's material, and assessing their interaction to determine the required contact force for removal. Consequently, the success of coverage depends on unknown or difficult-to-measure parameters, making it challenging to model all interactions. Without an accurate model, motion planning is prone to failure. By analyzing previous research [125] and observing how humans address these challenges, we argue that humans bypass the complexities of planning by solving the simpler closed-loop control problem. Humans leverage visual and tactile feedback for online adaptation. Similarly, robots can measure progress in tactile coverage tasks using vision, turning the task of identifying uncovered regions into an image segmentation problem. This problem has been addressed using various model-based [115, 163] or learning-based algorithms [46, 152]. However, determining how to control a robot to cover these target regions on curved surfaces remains an open challenge.

Existing research on coverage has primarily focused on coverage path planning, which

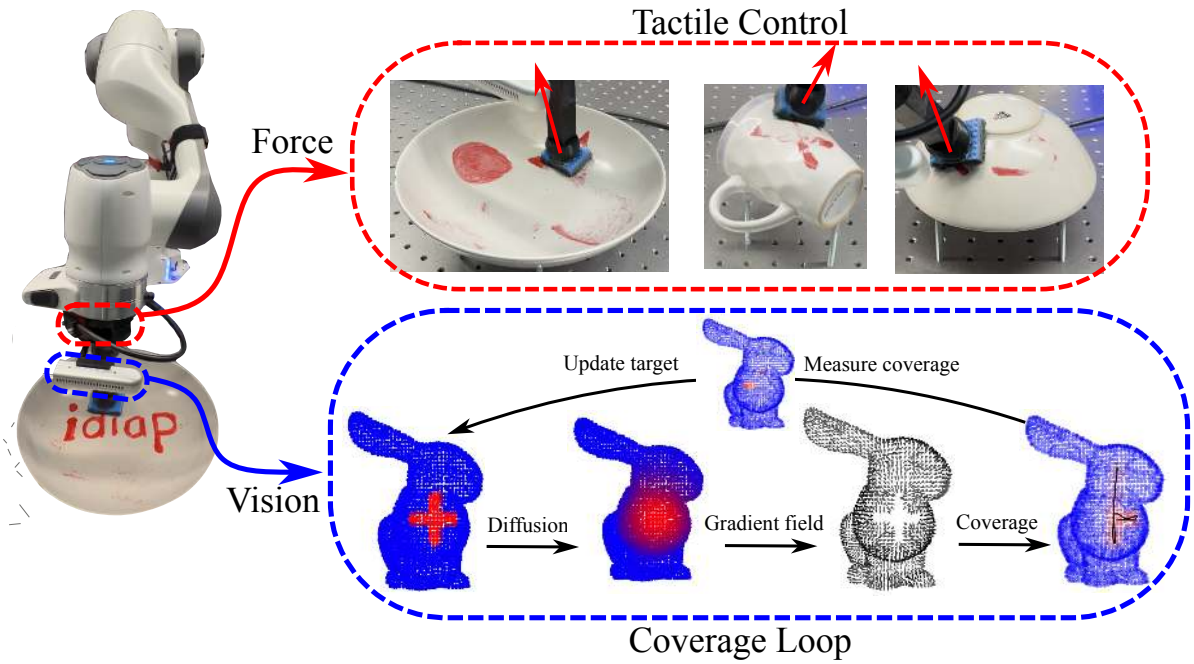


Figure 10.1.: Overview of our feedback control method for tactile coverage. *Left:* We measure the surface and the red target using the camera and encode them in a point cloud. *Bottom-right:* We diffuse the target and use its gradient field to guide the coverage. Then, we close the loop by measuring the actual coverage with the camera and use it as the next target. *Top-right:* We measure the tactile interaction forces using the force sensor and the tool orientation using the joint positions. We solve the geometric task-space impedance control problem using a line target and a force target along the line.

involves optimizing a path to ensure that a specified region of interest is covered within a set time frame. Traditionally, the underlying assumption is that visiting each point in the region of interest only once is sufficient for full coverage, an assumption that is reasonable for simple interactions, but not for many tactile tasks. Tactile interactions are often too complex to model deterministically, making it challenging to ensure full coverage after a single visit. Instead, for a cleaning task, a relatively dirty region requires more visits compared to a less dirty region. Similarly, in a surface inspection task, regions requiring higher precision demand more visits to compensate for sensor uncertainty. Furthermore, the robot is expected to keep in contact with the surface while moving, which significantly increases the cost of movement. This cost depends on the geodesic distance on the surface rather than the Euclidean distance. Therefore, naive sampling strategies that fail to account for the cost or constraints of movement and/or surface geometry are unsuitable for tactile coverage tasks. In contrast, ergodic coverage [161] controls the trajectories of *dynamical systems* by correlating the average time spent in

a region to the target spatial distribution. Therefore, ergodic coverage incorporates the motion model as the system dynamics and directly controls the coverage trajectories by using the spatial distribution measured by the vision system.

Considering these challenges, we present a closed-loop tactile ergodic control method that operates on point clouds for tactile coverage tasks. Using point clouds enables us to acquire the target object and spatial distribution at runtime using vision, measure coverage progress, and compensate for unmodeled dynamics in tactile coverage tasks. Our method then constrains the ergodic control problem to arbitrary surfaces to cover a target spatial distribution on the surface. We propagate coverage information by solving the diffusion equation on point clouds, which we compute in real-time by exploiting the surface's intrinsic basis functions called Laplacian eigenfunctions. These eigenfunctions generalize the Fourier series to manifolds (i.e., curved spaces). In order to exert a desired force on the surface while moving, we formulate a geometric task-space impedance controller using geometric algebra. This controller uses surface information to track a line target that is orthogonal to the surface while simultaneously exerting the desired force in the direction of that line. Notably, the geometric formulation ensures that these two objectives do not conflict with each other and can therefore be included in the same control loop without requiring exhaustive parameter tuning. In summary, our proposed closed-loop tactile ergodic control method offers the following contributions:

formulating the tactile coverage as closed-loop ergodic control problem on curved surfaces

closing the coverage loop by solving ergodic control problem on point clouds using diffusion

achieving real-time frequencies by computing the diffusion using Laplacian eigenfunctions

contact line and force tracking without conflicting objectives

The rest of the chapter is organized as follows. Section 10.2 describes related work. Section 10.3 provides the mathematical background. Section 10.4 presents our method. In Section 10.5, we demonstrate the effectiveness of our method in simulated and real-world experiments. Finally, we discuss our results in Section 10.6.

10.2. Related Work

The majority of the coverage methods consider the problem from a planning perspective and are generally known as coverage path planning (CPP) algorithms [131, 55]. Although these methods can handle planar regions with various boundaries [172, 56, 3], their extension to curved surfaces imposes limiting assumptions, such as projectively planar [99] or pseudo-extruded surfaces [14]. Additionally, CPP methods assume that the coverage target is uniformly distributed in space. Extending CPP methods to

account for spatial correlations in the information leads to informative path planning (IPP) [223]. Most IPP and CPP approaches address a variant of the NP-hard traveling salesman problem [206], which limits their scalability as domain complexity increases. Consequently, existing methods are either open-loop [66] or impose limiting assumptions, such as convexity, for online planning updates [223].

Closely related to coverage is the problem of exploration, where the environment is initially unknown, and robots gather information using onboard sensors [2, 193]. Tactile exploration is particularly necessary for gathering information on surfaces that can only be acquired through contact [54]. A notable example is non-invasive probing (palpation) of tissue stiffness, which aids in disease diagnosis or surgery by providing additional anatomical information. For this purpose, Gaussian processes (GP) have been used for discrete [16] and continuous [51] probing to map tissue stiffness. While GP-based approaches effectively guide sampling locations, they do not account for the robot's dynamics. This limitation was later addressed by using trajectory optimization to actively search for tissue abnormalities [184]. Unlike other sensing modalities that depend solely on position, tactile interactions also depend on conditions such as relative velocity and contact pressure [122]. To address this, methods have been developed to model forces [169] and more complex interactions between robotic tools and surfaces [215].

The complexity of the problem increases further if we consider scenarios with a robot physically interacting with the environment. For example, in tasks like surface finishing (e.g., polishing, sanding, grinding), the surface itself changes, as material is removed [170]. Similarly, in cleaning tasks, the robot's actions affect the distribution of dirt on the surface [159]. To avoid complex modeling, there are approaches either relying on reinforcement learning [136] or deep learning [183]. In a very similar setting to ours, a manipulator was used to clean the stains on a curved surface by performing multiple passes [115]. However, this work used a sampling-based planner, which required to predefine the maximum number of cleaning passes. In contrast, we relate the target distribution (e.g., stain) directly to feedback control through ergodicity without requiring any task-specific assumptions.

In tactile coverage scenarios, visiting a region once can not guarantee full coverage, and predicting how many times the robot should revisit a particular spot is challenging. Consequently, defining a time horizon for trajectory optimization is difficult, as the quality of the result would be significantly affected by this hard-to-make choice. Instead, ergodic control relates how often the robot should revisit a particular spot to the target density at that spot. In this context, *ergodic* describes a dynamical system in which the time averages of functions along its trajectories are equal to their spatial averages [162]. The key advantage of ergodic control is its ability to handle arbitrary spatial target distributions without requiring a predefined time horizon. When the spatial target distribution is measurable, the ergodic controller can use this feedback to direct the system to visit regions with higher spatial probabilities more frequently. Recent findings

have demonstrated that ergodicity is not merely a heuristic [30]; it is the optimal method for collecting independent and identically distributed data while accounting for system dynamics.

The concept of ergodic control was introduced in the seminal work by Mathew and Mezić [161], which presented the spectral multiscale coverage (SMC) algorithm. SMC is a feedback control law based on the Fourier decomposition of the target distribution and robot trajectories, where *multiscale* aspect prioritizes low-frequency components over high-frequency ones, corresponding to starting with large-scale spatial motions before refining finer details. Since this behavior is achieved through a myopic feedback controller rather than an offline planner, the ergodic controller remains effective even when motion is obstructed [192]. Furthermore, various works have adapted SMC's objective within a trajectory optimization framework to incorporate additional objectives, such as obstacle avoidance [135], time-optimality [67] and energy-awareness [187]. Ergodic control has been used for tactile coverage and exploration in applications such as non-parametric shape estimation [1] and table cleaning through learning from demonstration [117]. However, all these formulations, which rely on the Fourier decomposition-based ergodic metric, are limited to rectangular domains in Euclidean space.

The first attempt to extend the ergodic control to Riemannian manifolds [110] utilized Laplacian eigenfunctions, which generalize the Fourier series to curved spaces. However, this approach was restricted to homogeneous manifolds, such as spheres and tori, where closed-form expressions for the Laplacian eigenfunctions are available. More recently, the *kernel ergodic metric* [204] was introduced as an alternative to SMC's ergodic metric, enabling extensions to Lie groups and offering improved computational scalability. Nonetheless, arbitrary curved surfaces collected using sensors, such as point clouds, lack both the group structure and the homogeneous manifold properties, presenting additional challenges.

Another alternative to SMC is the heat equation-driven area coverage (HEDAC) algorithm [108], which uses the diffusion equation, a second-order partial differential equation (PDE), to propagate information about uncovered regions to agents across the domain. Similar to SMC, the original HEDAC implementation was restricted to rectangular domains and lacked collision avoidance. Subsequent extensions have adapted HEDAC to planar meshes with obstacles [109], maze exploration [61], and CPP on non-planar meshes [107]. However, its application on curved surfaces remains limited to meshes and offline planning due to the heavy pre-processing required, which is both time and computation-intensive.

In addition to its use in HEDAC, the diffusion equation is widely used in geometry processing tasks, ranging from geodesic computation [60] to learning on surfaces [190]. Its key advantage lies in its ability to account for surface geometry while remaining agnostic to the underlying representation and discretization [190]. The diffusion equation is governed by a second-order differential operator called the Laplacian which can be

computed for arbitrary surfaces represented as meshes or point clouds using various discretization schemes [27, 141, 47]. In this work, we use a recent approach proposed by Sharp *et al.* which provides a robust and efficient implementation [191], capable of handling partial and noisy point clouds.

10.3. Background

10.3.1. Ergodic Control using Diffusion

The ergodic control objective correlates the time that a coverage agent spends in a region to the probability density specified in that region. The HEDAC method [108] encodes the coverage objective in the domain $\boldsymbol{x} \in \Omega$ at time t using a virtual source term

$$s(\boldsymbol{x}, t) = \max\left(p(\boldsymbol{x}) - c(\boldsymbol{x}, t), 0\right)^2, \tag{10.1}$$

where $p(\boldsymbol{x})$ is the probability distribution corresponding to the coverage target and $c(\boldsymbol{x}, t)$ is the normalized coverage of the N virtual coverage agents over the domain

$$c(\boldsymbol{x}, t) = \frac{\tilde{c}(\boldsymbol{x}, t)}{\int_\Omega \tilde{c}(\boldsymbol{x}, t) d\boldsymbol{x}}. \tag{10.2}$$

A single agent's coverage is the convolution of its footprint $\varphi(\boldsymbol{r})$ with its trajectory $\boldsymbol{x}_i(t')$. Then, the total coverage becomes the time-averaged sum of these convolutions

$$\tilde{c}(\boldsymbol{x}, t) = \frac{1}{Nt} \sum_{i=1}^{N} \int_0^t \varphi\left(\boldsymbol{x} - \boldsymbol{x}_i(t')\right) dt'. \tag{10.3}$$

HEDAC diffuses the source term across the domain Ω and computes the potential field $u(\boldsymbol{x}, t)$ using the stationary ($\dot{u}(\boldsymbol{x}, t) = 0$) diffusion (heat) equation

$$\alpha \Delta u(\boldsymbol{x}, t) - u(\boldsymbol{x}, t) + s(\boldsymbol{x}, t) = 0, \tag{10.4}$$

with the diffusion coefficient $\alpha > 0$ and the Laplacian operator Δ. The Laplacian is a second-order differential operator which reduces to the sum of the second partial derivatives in Euclidean spaces

$$\Delta f = \nabla \cdot \nabla f = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial \boldsymbol{x}_i^2}, \quad \forall \boldsymbol{x} \in \mathbb{R}^n. \tag{10.5}$$

The stationary diffusion equation (10.4) governs the potential field within the interior of the domain Ω, while the behavior on the boundary $\partial\Omega$ is dictated by the zero-Neumann

boundary condition

$$\mathbf{n} \cdot \nabla u(\boldsymbol{x}, t) = 0, \quad \forall \boldsymbol{x} \in \partial\Omega, \tag{10.6}$$

where $\mathbf{n}$ represents the outward unit normal vector to the boundary $\partial\Omega$. To guide the i-th coverage agent, HEDAC utilizes the smooth gradient field of the diffused potential $u(\boldsymbol{x}, t)$ and simulates first-order dynamics

$$\dot{\boldsymbol{x}}_i = \nabla u(\boldsymbol{x}_i, t). \tag{10.7}$$

10.4. Method

We present our closed-loop tactile ergodic coverage method in three parts: (i) surface preprocessing; (ii) tactile coverage; and (iii) robot control. The surface preprocessing computes the quantities that need to be calculated only once when the surface is captured. Tactile coverage generates the motion commands for the virtual coverage agent using the precomputed quantities from the surface preprocessing and the robot controller tracks the generated motion commands with a manipulator using impedance control.

10.4.1. Problem Statement

We formulate a tactile ergodic controller that covers target spatial distributions on arbitrary surfaces. Similar to HEDAC, we propagate the information encoding the coverage objective by solving the diffusion equation. However, we utilize the non-stationary ($\dot{u} \neq 0$) diffusion equation

$$\dot{u}(\boldsymbol{x}, \tau) = \Delta_{\mathcal{M}} u(\boldsymbol{x}, \tau), \tag{10.8}$$

as it allows control over the desired smoothness [32]. Since the diffusion equation depends on time, we introduce an additional time variable τ for diffusion. The diffusion time τ is independent of the coverage time t used by the HEDAC algorithm and unlike HEDAC, we require the initial condition $u(\boldsymbol{x}, 0)$. We set the initial condition using the source term given in Equation (10.1) which encodes the coverage objective at the t-th timestep of the coverage, i.e., $u(\boldsymbol{x}, 0) = s(\boldsymbol{x}, \tau)$. Additionally, here we use $\Delta_{\mathcal{M}}$, which generalizes the Laplacian for Euclidean spaces Δ to non-Euclidean manifolds $\mathcal{M}$. This operator $\Delta_{\mathcal{M}}$ is also known as Laplace-Beltrami operator but for conciseness we will use the term Laplacian.

Our coverage domains are curved surfaces (i.e. 2-manifolds) and we capture the underlying manifold $\mathcal{M}$ as a point cloud $\mathcal{P}$ composed of $n_{\mathcal{P}}$ points using an RGB-D camera

$$\mathcal{P} := \left\{ (\boldsymbol{x}_i, \boldsymbol{c}_i) \;\middle|\; \begin{array}{l} \boldsymbol{x}_i \in \mathbb{R}^3,\ \boldsymbol{c}_i \in \{0, \ldots, 255\}^3 \\ \text{for } i = 1, \ldots, n_{\mathcal{P}} \end{array} \right\}, \tag{10.9}$$

where $\boldsymbol{x}_i$ is the position of the i-th surface point in Euclidean space and $\boldsymbol{c}_i$ is the vector of RGB color intensities. We assume there is a processing pipeline (i.e., such as [115, 190, 49]) which maps the point positions and colors to the probability mass p_i of the spatial distribution encoding the coverage objective. Accordingly, our coverage target becomes a discrete spatial distribution $p(\boldsymbol{x}_i) = p_i$ on the point cloud $\mathcal{P}$.

In order to solve (10.8) on irregular and discrete domains, such as point clouds, we discretize the problem in space and time. Hence, we use $u_{i,\tau}$ to denote the value of the potential field at the i-th point at the τ-th timestep. We omit the subscript i if we refer to all points.

10.4.2. Surface Preprocessing

First, we compute the spatial discretization of the Laplacian $\Delta_{\mathcal{M}}$. Note that there are various approaches for discretizing the Laplacian on point clouds [191, 27, 141, 47]. In this work, we follow the approach presented in [191] and show a simplified version of it here, but refer the readers to the original work for more details. Using this method, the discrete Laplacian is represented by the matrix $\boldsymbol{L} \in \mathbb{R}^{n_{\mathcal{P}} \times n_{\mathcal{P}}}$

$$\boldsymbol{L} = \boldsymbol{M}^{-1}\boldsymbol{C}, \tag{10.10}$$

where $\boldsymbol{M}$ is the diagonal mass matrix and $\boldsymbol{C}$ is a sparse symmetric matrix called the weak Laplacian. The entries of $\boldsymbol{M}$ correspond to the Voronoi cell areas in the local tangent plane around each point of $\mathcal{P}$. Similarly, the entries of $\boldsymbol{C}$ are determined by the connectivity of the points on the local tangent space and the distance between the connected points. Note that the local tangent space structure also identifies the boundary points. For a given point, the lines between the original point and its neighbors are constructed. If the angle between two consecutive lines is greater than $\pi/2$, the point is a boundary and its boundary condition is set as zero-Neumann.

Next, we discretize the diffusion equation (10.8) in time and incorporate the discrete Laplacian $\boldsymbol{L}$. Using the backward Euler method, we derive the implicit time-stepping equation, which remains stable for any timestep τ

$$\frac{1}{\tau}(\boldsymbol{u}_\tau - \boldsymbol{u}_0) = \boldsymbol{L}\boldsymbol{u}_\tau, \tag{10.11}$$

where $\boldsymbol{u}_0$ and $\boldsymbol{u}_\tau$ are column vectors containing the potential field values at the vertices of the point cloud at the initial and final times, respectively. Then, combining Equations (10.10) and (10.11) and solving for $\boldsymbol{u}_\tau$ we obtain the linear system

$$\boldsymbol{u}_\tau = (\boldsymbol{M} - \tau\ \boldsymbol{C})^{-1}\boldsymbol{M}\boldsymbol{u}_0. \tag{10.12}$$

Note that solving (10.12) requires inverting a large sparse matrix, which might be com-

putationally expensive depending on the size of the point cloud and requires the timestep to be set before the inversion. Alternatively, we can solve the problem in the spectral domain by projecting the original problem and reprojecting the solution back to the point cloud. This procedure generalizes using the Fourier transform for solving the diffusion equation on a rectangular domain in $\mathbb{R}^n$ to arbitrary manifolds. Note that the Fourier series are the eigenfunctions of the Laplacian Δ in $\mathbb{R}^n$. Therefore, we can use the eigenvectors of the discrete Laplacian $\boldsymbol{L}$ for solving the diffusion equation on point clouds.

We can write the generalized (i.e., $\boldsymbol{M} \neq \boldsymbol{I}$) eigenvalue problem for the Laplacian as

$$\boldsymbol{C}\boldsymbol{\phi}_m = \lambda_m \boldsymbol{M}\boldsymbol{\phi}_m, \tag{10.13}$$

where $\{\lambda_m, \boldsymbol{\phi}_m\}$ are the eigenvalue/eigenvector pairs. Since $\boldsymbol{M}$ is diagonal and $\boldsymbol{C}$ is symmetric positive definite, by the spectral theorem, we know that the eigenvalues are real, non-negative, and in ascending order analogous to the frequency. Therefore, we can use the first n_M eigenvalue/eigenvector pairs as a low-frequency approximation of the whole spectrum. Furthermore, the eigenvectors are orthonormal with respect to the inner product defined by the mass matrix $\boldsymbol{M}$. Accordingly, we can stack the first n_M eigenvectors $\boldsymbol{\phi}_m$ as column vectors to construct the matrix $\boldsymbol{\Phi} \in \mathbb{R}^{n_\mathcal{P} \times n_M}$ encoding an orthonormal transformation $\boldsymbol{\Phi}^\top \boldsymbol{M} \boldsymbol{\Phi} = \boldsymbol{I}$. Then, we can transform the coordinates (shown with superscripts) from the point cloud to the spectral domain

$$\boldsymbol{u}^{\boldsymbol{\phi}} = \boldsymbol{\Phi}^\top \boldsymbol{M} \boldsymbol{u}^{\boldsymbol{x}}. \tag{10.14}$$

Note that this step is equivalent to computing the Fourier series coefficients of a target distribution in SMC. Due to the orthonormal transformation, the PDE on the point cloud becomes a system of decoupled ODEs in the spectral domain. It is well known that the solution of a first-order linear ODE $\dot{x}(\tau) = -cx(\tau)$ is given by $x(\tau) = e^{-c\tau}x(0)$, where c is a constant and $x(0)$ is the initial condition. Therefore, the solution of the system of ODEs in the spectral domain is given in matrix form as

$$\boldsymbol{u}^{\boldsymbol{\phi}}_\tau = \begin{bmatrix} e^{-\lambda_1 \tau} & \dots & e^{-\lambda_m \tau} \end{bmatrix}^\top \odot \boldsymbol{u}^{\boldsymbol{\phi}}_0, \tag{10.15}$$

where $\odot$ denotes the Hadamard product. We observe from (10.15) that the exponential terms with larger eigenvalues (i.e., higher frequencies) will decay faster. Therefore, approximating the diffusion using the first n_M components introduces minimal error. Secondly, similar to the mixed norm used in SMC, the low-frequency spatial features are prioritized. Next, we transform the solution back to the point cloud to get the diffused potential field

$$\boldsymbol{u}^{\boldsymbol{x}} = \boldsymbol{\Phi} \boldsymbol{u}^{\boldsymbol{\phi}}. \tag{10.16}$$

We can combine (10.14), (10.15) and (10.16) into a unified spectral scheme

$$\boldsymbol{u}_\tau = \Phi \left[\begin{array}{ccc} e^{-\lambda_1 \tau} & \dots & e^{-\lambda_m \tau} \end{array} \right]^\top \odot (\Phi^\top \boldsymbol{M} \boldsymbol{u}_0) \,. \tag{10.17}$$

We omit the superscripts when working on the point cloud for brevity. Note that τ is the only free parameter in the diffusion computation. However, its value should be adapted according to the mean spacing between the adjacent points h on the point cloud. For that purpose, we introduce the hyperparameter $\alpha > 0$ and embed it into the timestep calculation

$$\tau = \alpha h^2. \tag{10.18}$$

Accordingly, we can control the diffusion behavior independently of the point cloud size. Increasing α results in longer diffusion times and attenuates the high-frequency spatial features (see (10.15) for details). This corresponds to a more global coverage [32]. Conversely, decreasing α results in shorter diffusion times, which leads to preserving the high-frequency spatial features, hence more local coverage behavior.

Note that the Laplacian is determined completely by the connectivity on the local tangent space and the distance between these connected points. Therefore, it is invariant to distance preserving (i.e., isometric) transformations such as rigid body motion or deformation without stretching. Accordingly, we compute $\boldsymbol{C}$, $\boldsymbol{M}$ and derived quantities only once in the preprocessing step for a given surface. Recomputation is not necessary if the object stays still, moves rigidly, or the target distribution p_i changes.

10.4.3. Tactile Ergodic Coverage

We model the actual coverage tool/sensor as a compliant virtual coverage agent shaped as a disk with radius r_a. Notably, one can represent arbitrary tool/sensor footprints as a combination of disks [32]. We position our agent at the end-effector of our manipulator. Thus, for a given kinematic chain and joint configuration $\boldsymbol{q}$, we can use the forward kinematics to compute the position of our agent as a conformal point P_a

$$P_a = M(\boldsymbol{q}) \boldsymbol{e}_0 \widetilde{M}(\boldsymbol{q}). \tag{10.19}$$

Since the point cloud is discrete and the agent should move continuously on the surface, we project our agent P_a and its footprint to the closest local tangent space on the point cloud.

Local Tangent Space and Coverage Computation

Given the agent's position P_a, we first compute the closest tangent space on the point cloud. For that, we query a K-D tree $\mathcal{T}(\mathcal{P})$ for the points $\boldsymbol{x}_i \in \mathcal{P}$ that are within the radius r_a of the agent. Then, we compute the conformal embeddings P_i of the

neighboring Euclidean points $\boldsymbol{x}_i$ using (2.20). We refer to the set composed of points P_i as the local neighborhood. Then, we fit a tangent space to the local neighborhood by minimizing the classical least squares objective

$$\min \sum_{i=1}^{n_N} (P_i \cdot X^*)^2, \tag{10.20}$$

where X^* is the dual representation of either a plane or a sphere and the inner product $\cdot$ is a distance measure. In CGA, planes can be seen as limit cases of spheres, i.e. planes are spheres with infinite radius. This is also easy to observe by looking at Equations (2.24) and (2.25) which construct these geometric primitives. Note that fitting a local tangent sphere with the radius determined by the local curvature would always result in smaller or equal residuals than fitting a plane.

It has been shown in [102] that the solution to the least squares problem given in (10.20) is the eigenvector corresponding to the smallest eigenvalue of the 5×5 matrix

$$b_{j,k} = \sum_{i=1}^{n_N} w_{i,j} w_{i,k}, \tag{10.21}$$

where

$$w_{i,k} = \begin{cases} p_{i,k} & \text{if } k \in \{1,2,3\} \\ -1 & \text{if } k = 4 \\ -\frac{1}{2}\boldsymbol{p}_i^2 & \text{if } k = 5. \end{cases} \tag{10.22}$$

Using the five components v_i of this eigenvector we can find the geometric primitive as

$$X = (v_0 \boldsymbol{e}_0 + v_1 \boldsymbol{e}_1 + v_2 \boldsymbol{e}_2 + v_3 \boldsymbol{e}_3 + v_4 \boldsymbol{e}_\infty)^* . \tag{10.23}$$

Note that if X is a plane then $v_0 = 0$, otherwise X is a sphere. Next, we want to project P_a to X by using the general subspace projection formula of CGA

$$P_{pair} = \big((P_a \wedge \boldsymbol{e}_\infty) \cdot X\big) X^{-1}. \tag{10.24}$$

Here we first construct the pointpair $P_a \wedge \boldsymbol{e}_\infty$, where $\boldsymbol{e}_\infty$ corresponds to the point at infinity. $P_a \wedge \boldsymbol{e}_\infty$ is also called a flat point. Note that the projection essentially amounts to first constructing the dual line $(P_a \wedge \boldsymbol{e}_\infty) \cdot X$ that passes through the point P_a and is orthogonal to X, then intersecting this line with the primitive X.

If X is a sphere, then the intersection of the line and the sphere will result in two points on the sphere. If X is a plane, it will result in another flat point, i.e. one point on the plane and one at infinity. In any case, we can retrieve the closer one to the agent

position P_a using the split operation

$$P'_a = \text{split}\left[P_p\right]. \tag{10.25}$$

Here, P'_a is the projected agent position on the tangent space X. Next, we compute our agent's footprint (i.e., instantaneous coverage) by projecting its surface to the point cloud. If the target surface was flat, all the points within the radius r_a of our agent P'_a would be covered by the footprint. However, in the general case, both the tool and the surface can be curved and deformable. For simplicity, we assume that the surface is rigid, and it deforms the tool with a constant bending radius. We use the radius of the local tangent sphere that we computed using CGA as an approximation for the bending radius. Accordingly, we can quantify the error of the local tangent space approximation for the i-th neighbor P_i by the normalized residuals e_i of the least squares computation (10.20). We encode this approximation error into the footprint by weighting the i-th neighbor by the Gaussian kernel $\varphi(r)$ using the normalized residuals $r_i = e_i / \max(\boldsymbol{e})$

$$\varphi(r_i) = \exp\left(-\varepsilon^2 r_i^2\right), \tag{10.26}$$

where the hyperparameter $\varepsilon > 0$ controls the coverage falloff. Next, we substitute the Gaussian kernel weighted footprint into (10.3) to compute the coverage $\boldsymbol{c}_t$, which is then used to calculate the virtual source term $\boldsymbol{s}_t$ via (10.1). As mentioned earlier, this virtual source term serves as the initial condition for the diffusion equation (10.8) at each iteration of the tactile coverage loop, i.e., $\boldsymbol{u}_0 = \boldsymbol{s}_t$.

Gradient of the Diffused Potential Field

We guide the coverage agent using the gradient of the diffused potential field as the acceleration command

$$\ddot{P}'_a = \nabla u_{P'_a,\tau}, \tag{10.27}$$

where $\nabla u_{P'_a,\tau}$ denotes the gradient of the diffused potential field at the projected agent position P'_a. However, computing the gradient on the point cloud is more involved than a regular grid or a mesh. Recall that in Section 10.4.3, we already computed the projected agent position P'_a, the local neighborhood and the tangent space X^*. As the first step, we compute the tangent plane $E_{a,\tau}$ at P'_a, namely

$$E_{a,\tau} = L^*_{a,\perp} \wedge P'_a \wedge \boldsymbol{e}_\infty, \tag{10.28}$$

using the line $L_{a,\perp}$, which is orthogonal to the surface and passes through P'_a. It is found by wedging the dual primitive X with P'_a to infinity with

$$L_{a,\perp} = X^* \wedge P'_a \wedge \boldsymbol{e}_\infty. \tag{10.29}$$

Then, we project the points P_i in the local neighborhood to the tangent plane $E_{a,\tau}$ using (10.24) and (10.25), by setting $E_{a,\tau}$ as the primitive X. Next, we use the values of the potential field at the neighbor locations as the height $h_i = u_{i,\tau}$ of a second surface from the tangent plane. Then, we fit a 3-rd degree polynomial to this surface as shown by using the weighted least squares objective

$$\hat{\boldsymbol{A}} = \arg\min_{\boldsymbol{A}} \operatorname{tr}\left((\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A})^\top \boldsymbol{W} (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{A})\right), \tag{10.30}$$

with the diagonal weight matrix $\boldsymbol{W}$

$$\boldsymbol{W} = \operatorname{diag}\left(\varphi(r_1), \varphi(r_1), \ldots \varphi(r_m)\right), \tag{10.31}$$

whose entries are given by the Gaussian kernel (10.26). One can refer to [167] for the details. Lastly, we calculate the gradient at the projected agent's position using the analytical gradients of the polynomial. We depict the approach visually in Figure 10.2.

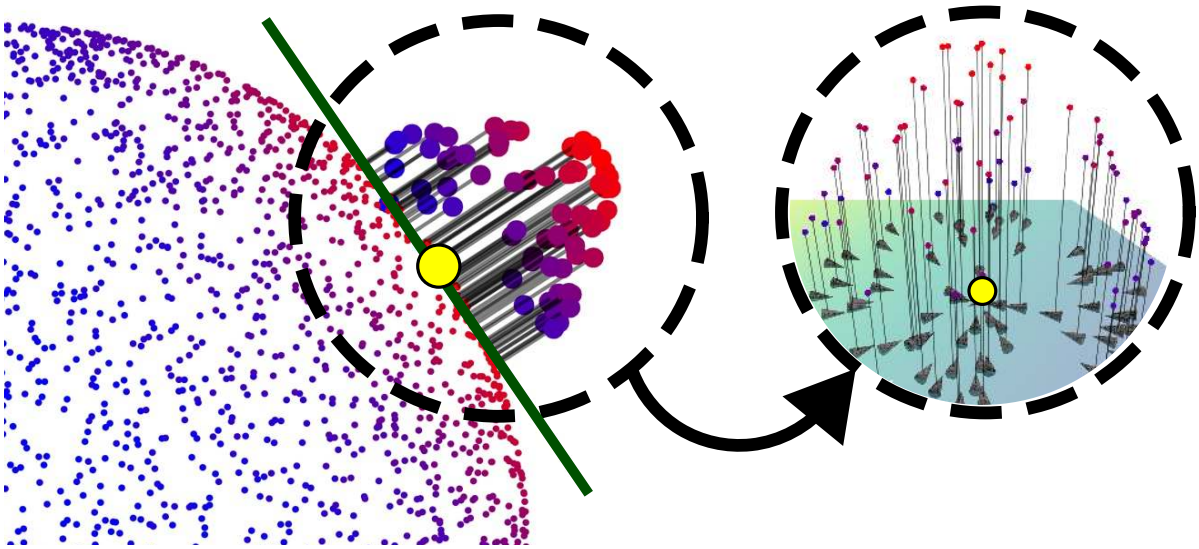


Figure 10.2.: Blue-red points show the value of the potential field u_τ on the pointcloud $\mathcal{P}$ and the yellow point is the projected agent position P'_a. We also project the agent's neighbors P_i to the tangent plane $E_{a,\tau}$, shown in green. Next, we use the height function $h_i = u_{i,\tau}$ which uses the values of the potential field to lift the projected points in the normal direction of the tangent plane. We show the lifted points with large blue-red points. We fit a polynomial to this lifted surface and compute its analytical gradients at the neighbor locations $\nabla u_{i,\tau}$, as shown with arrows in the detail view.

10.4.4. Robot Control

There are several aspects that the control of the physical robot needs to achieve. The first is to track the virtual coverage agent on the target surface, while keeping the end-effector normal to the surface. The second is to exert a desired force on the surface. To do so, we design a task-space impedance controller while further exploiting geometric algebra for efficiency and compactness. The control law is of the following form

$$\boldsymbol{\tau} = -\boldsymbol{\mathcal{J}}^{\top} \cdot \mathcal{W}, \tag{10.32}$$

where $\boldsymbol{\mathcal{J}} \in \mathbb{B}^{1\times N} \subset \mathbb{G}_{4,1}^{1\times N}$ is the Jacobian multivector matrix with elements corresponding to bivectors, $\mathcal{W}$ is the desired task-space wrench and $\boldsymbol{\tau}$ are the resulting joint torques. Before composing the final control law, we will explain its components individually.

Surface Orientation

From Equation (10.29), we obtained a line $L_{a,\perp}$ that is orthogonal to the surface that we wish to track. In [130], it was shown how the motor between conformal objects can be obtained. We use this formulation to find the motor between the target orthogonal line and the line that corresponds to the z-axis of the end-effector of the robot in its current configuration, which is found as

$$L_{ee} = M(\boldsymbol{q})(\boldsymbol{e}_0 \wedge \boldsymbol{e}_3 \wedge \boldsymbol{e}_\infty)\widetilde{M}(\boldsymbol{q}). \tag{10.33}$$

Then, the motor $M_{L_{ee}L_{a,\perp}}$, which transforms L_{ee} into $L_{a,\perp}$ can be found as

$$M_{L_{ee}L_{a,\perp}} = \frac{1}{C}\left(1 + L_{a,\perp}L_{ee}\right), \tag{10.34}$$

where C is a normalization constant. Note that C does not simply correspond to the norm of $1 + L_{a,\perp}L_{ee}$, but requires a more involved computation. We therefore omit its exact computation here for brevity and refer readers to [130].

We can now use the motor $M_{L_{ee}L_{a,\perp}}$ in order to find a control command for the robot via the logarithmic map of motors, i.e.

$$\mathcal{V}_{L_{a,\perp}} = \log\left(M_{L_{ee}L_{a,\perp}}\right). \tag{10.35}$$

Of course, if the lines are equal, $M_{L_{ee}L_{a,\perp}} = 1$ and consequently $\mathcal{V}_{L_{a,\perp}} = 0$. Note that $\mathcal{V}_{L_{a,\perp}}$ is still a command in task space (we will explain how to transform it to a joint torque command once we have derived all the necessary components).

Another issue is that algebraically, $\mathcal{V}_{L_{a,\perp}}$ corresponds to a twist, not a wrench. Hence, we need to transform it accordingly. From physics, we know that twists transform to wrenches via an inertial map, which we could use here as well. In the context of control,

this inertia tensor is, however, a tuning parameter and does not actually correspond to a physical quantity. Thus, in order to simplify the final expression, we will use a scalar matrix valued inertia, instead of a geometric algebra inertia tensor and choose to transform the twist command to wrench command purely algebraically. As it has been shown before, this can be achieved by the conjugate pseudoscalar $I_c = I\boldsymbol{e}_0$ [100]. It follows that

$$\mathcal{W}_{L_{a,\perp}} = \mathcal{V}_{L_{a,\perp}} I_c, \tag{10.36}$$

and $\mathcal{W}_{L_{a,\perp}}$ now algebraically corresponds to a wrench.

Target Surface Force

Since this chapter describes a method for tactile surface coverage, the goal of the robot control is to not simply stay in contact with the surface, but to actively exert a desired force on the surface. First of all, we denote the current measured wrench as $\mathcal{W}_m(t)$ and the desired wrench as $\mathcal{W}_d$. Both are bivectors as defined by Equation (2.44). We use $\mathcal{W}_d$ w.r.t. end-effector in order to make it more intuitive to define. Hence, we need to transform $\mathcal{W}_m(t)$ to the same coordinate frame, i.e.

$$\mathcal{W}'_m(t) = \widetilde{M}(\boldsymbol{q})\mathcal{W}_m(t)M(\boldsymbol{q}). \tag{10.37}$$

In order to achieve the desired interaction force, we simply apply a standard PID controller in wrench space, i.e.

$$\mathcal{W}_C = \boldsymbol{K}_{p,\mathcal{W}}\mathcal{W}_e + \boldsymbol{K}_{i,\mathcal{W}} \int_0^\top \mathcal{W}_e(\tau)\mathrm{d}\tau + \boldsymbol{K}_{d,\mathcal{W}} \frac{\mathrm{d}}{\mathrm{d}t}\mathcal{W}_e(t), \tag{10.38}$$

where the wrench error is

$$\mathcal{W}_e(t) \;=\; \mathcal{W}_d - \mathcal{W}'_m(t), \tag{10.39}$$

where $\boldsymbol{K}_{p,\mathcal{W}}$, $\boldsymbol{K}_{i,\mathcal{W}}$ and $\boldsymbol{K}_{d,\mathcal{W}}$ are the corresponding gain matrices, and $\mathcal{W}_C$ is the resulting control wrench.

Since the desired wrench is defined in end-effector coordinates, it usually amounts to a linear force in the z-direction of the end-effector frame, i.e. $\mathcal{W}_d = f_d\boldsymbol{e}_{03}$. Additionally, for an improved cleaning behavior one could also set a desired torque around that axis by adding $\tau_d\boldsymbol{e}_{12}$. The pattern of how to set this torque, however, would be subject to further investigation.

Task-Space Impedance Control

Recalling the control law from Equation (10.32), we now collect the terms from the previous subsections into a unified task-space impedance control law. We start by looking

in more detail at the Jacobian $\boldsymbol{\mathcal{J}}$. Previously, we mentioned that we are using the current end-effector motor as the reference, hence, we require the Jacobian to be computed w.r.t. that reference. This is therefore not the geometric Jacobian that was presented in Equation (3.13), but a variation of it. The end-effector frame geometric Jacobian $\boldsymbol{\mathcal{J}}_G^{ee}$ can be found as

$$\boldsymbol{\mathcal{J}}_G^{ee} = \begin{bmatrix} B_1^{ee} & \dots & B_N^{ee} \end{bmatrix}, \tag{10.40}$$

where the bivector elements can be found as

$$B_i^{ee} = \widetilde{M}_i^{ee}(\boldsymbol{q}) B_i M_i^{ee}, \tag{10.41}$$

with

$$M_i^{ee} = \prod_{j=N}^{i} M_i(q_i). \tag{10.42}$$

Hence, the relationship between $\boldsymbol{\mathcal{J}}_G$ and $\boldsymbol{\mathcal{J}}_G^{ee}$ can be found as

$$\boldsymbol{\mathcal{J}}_G^{ee} = \widetilde{M}(\boldsymbol{q}) \boldsymbol{\mathcal{J}}_G M(\boldsymbol{q}). \tag{10.43}$$

The wrench in the control law is composed of the three wrenches that we defined in the previous subsections. As commonly done, we add a damping term that corresponds to the current end-effector twist and as before, we transform it to an algebraic wrench, i.e.

$$\mathcal{W}_{\mathcal{V}} = \boldsymbol{\mathcal{J}}_G^{ee} \dot{\boldsymbol{q}} \boldsymbol{e}_{0\infty}. \tag{10.44}$$

With this, we now have everything in place to compose our final control law as

$$\boldsymbol{\tau} = -\boldsymbol{\mathcal{J}}_G^{ee,\top} \cdot \left(\boldsymbol{K}_{L_{a,\perp}} \mathcal{W}_{L_{a,\perp}} - \boldsymbol{D}_{\mathcal{V}} \mathcal{W}_{\mathcal{V}} + \mathcal{W}_C \right), \tag{10.45}$$

where $\boldsymbol{K}_{L_{a,\perp}}$ is a stiffness and $\boldsymbol{D}_{\mathcal{V}}$ a damping gain.

10.5. Experiments

Our experimental setup comprises a BotaSys SensOne 6-axis force torque (F/T) sensor attached to the wrist of a 7-axis Franka Emika robot manipulator and a custom 3-D printed part attached to the F/T sensor. The custom part interfaces an Intel Realsense D415 depth camera and a sponge at its tip. We consider the sponge's center point to be the coverage agent's position P_a. Before the operation, we perform extrinsic calibration of the camera to combine the depth and RGB feeds from the camera and to obtain its transformation with respect to the robot joints. Additionally, we calibrate the F/T sensor to compensate for the weight of the 3-D printed part and the camera. We show the experimental setup on the left of Figure 10.1.

10.5.1. Implementation Details

The pipeline of our tactile ergodic coverage method consists of three modules: (i) surface acquisition, (ii) surface coverage and (iii) robot control. Figure 10.3 summarizes the information flow between the components.

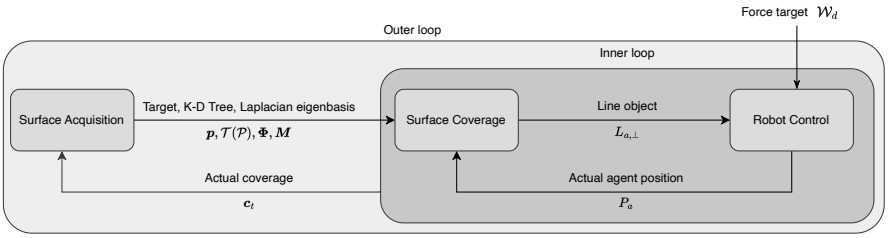


Figure 10.3.: Information flow between the three components. The pipeline is composed of an outer loop responsible for controlling the coverage progress with the feedback from the camera, whereas the inner loop compensates for the mismatch due to the robot dynamics.

Surface Acquisition

The surface acquisition node is responsible for collecting the point cloud and performing preprocessing operations described in Section 10.4.2. We use *scipy*[1] for the nearest neighbor queries and for solving the eigenproblem in (10.13). The matrices $\boldsymbol{C}$ and $\boldsymbol{M}$ composing the discrete Laplacian in (10.10) are computed with the *robust_laplacian* package[2] [191].

Surface Coverage

The surface coverage node performs the computations based on the procedure given in Section 10.4.3. It uses the information provided by the surface acquisition node and produces the target *line* for the robot control node.

Robot control

On a high level, the robot control can be seen as a state machine with three discrete states. The first two states are essentially two pre-recorded joint positions in which the robot is waiting for other parts of the pipeline to be completed. One of these positions corresponds to the picture-taking position, i.e., a joint position where the camera has the full object in its frame and the point cloud can be obtained. The robot is waiting in this position until the point cloud has been obtained, afterwards it changes

[1] `https://scipy.org`
[2] `https://github.com/nmwsharp/robust-laplacians-py`

its position to hover shortly over the object. In this second position, it is waiting for the computation of the Laplacian eigenfunctions to be completed, such that the coverage can start. The switching between those two positions is achieved using a simple joint impedance controller.

The third, and most important, state is when robot is actually controlled to be in contact with the surface and to follow the target corresponding to the coverage agent. This behaviour is achieved using the controller that we described in Section 10.4.4. The relevant parameters, that were chosen empirically for the real-world experiments, are the stiffness and damping of the line tracking controller, i.e. $K_{L_{a,\perp}} = \text{diag}(30, 30, 30, 750, 750, 300)$ and $D_{\mathcal{V}} = \text{diag}(10, 10, 10, 150, 150, 50)$, as well as the gains of the wrench PID controller, i.e. $K_{p,\mathcal{W}} = 0.5$, $K_{i,\mathcal{W}} = 5$ and $K_{d,\mathcal{W}} = 0.5$. The controller has been implemented using our open-source geometric algebra for robotics library *gafro*[3] that we first presented in [148]. Note that in some cases, matrix-vector products of geometric algebra quantities have been used for the implementation, where the mathematical structure of the geometric product actually simplifies to this, which can be exploited for more efficient computation.

10.5.2. Simulated Experiments

Computation Performance

In order to assess the computational performance, we investigated the two main operations of our method: (i) preprocessing by solving either the eigenproblem (10.13) or matrix inversion in (10.12) (ii) integrating the diffusion at runtime using either the spectral (10.17) or implicit (10.12) formulations. In this experiment, we used the Stanford Bunny as the reference point cloud and performed voxel filtering to set the point cloud resolution. We present the results for the preprocessing in Figure 10.4 and for the runtime in Figure 10.5.

Coverage Performance

We tested the coverage performance in a series of kinematic simulations. As the coverage metric, we used the normalized ergodicity over the target distribution, which compares the time-averaged statistics of agent trajectories to the target distribution

$$\boldsymbol{\varepsilon}_t = \frac{\| \max(\boldsymbol{p} - \boldsymbol{c}_t, 0) \|_2}{\sum_{i=1}^{n_{\mathcal{P}}} p_i}. \tag{10.46}$$

We ran the experiments for three different objects: a partial point cloud of the Stanford Bunny and two point clouds of a cup and a plate and their target distributions that we

[3] `https://gitlab.com/gafro`

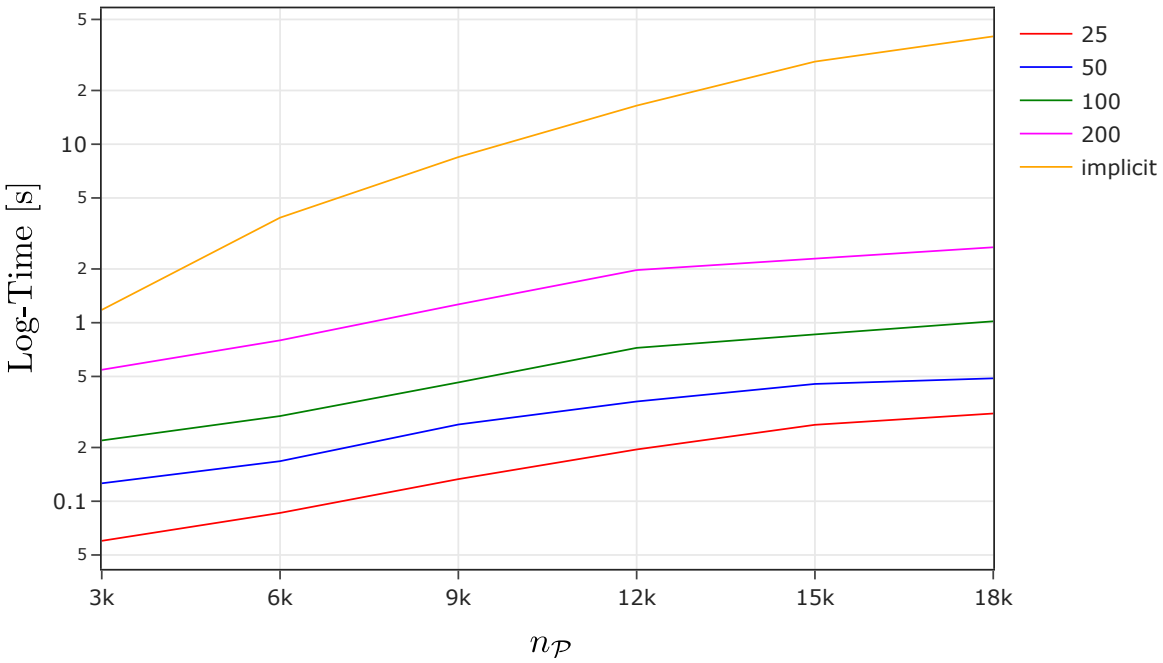


Figure 10.4.: Computational complexity of the preprocessing step for different $n_{\mathcal{P}}$ and n_M. Legend shows n_M values. The time axis is logarithmic and the legend shows n_M values.

collected using the RGB-D camera. For the Stanford bunny, we projected an 'X' shape as the target distribution. For each object, we sampled ten different initial positions for the coverage agent and kinematically simulated the coverage using different numbers of eigencomponents $n_M = 25, 50, 100, 200$ and diffusion timestep scalar $\alpha = 1, 5, 10, 50, 100$. Since the plate is larger compared to the Bunny and the cup, we used a larger agent radius $r_a = 15$ $[mm]$ for the plate and a smaller value $r_a = 7.5$ $[mm]$ for the cup and the Bunny. The other parameters that we kept constant in all of the experiments are $\ddot{x}_{\max} = 3$ $[mm/s^2]$, $\dot{x}_{\max} = 3$ $[mm/s]$. We selected six representative experiment runs to show the coverage performance qualitatively, and present them in Figure 10.6.

We show the quantitative results with respect to n_M and α in Figures 10.7 and 10.8, respectively. Note that, in order to better show performance trend in these plots, we have excluded parameter combinations leading to failure cases. We will discuss those in Section 3.7.

As the last experiment, we chose the best-performing pair (n_M, α) and show the time evolution of the coverage performance for different objects in Figure 10.9.

10.5.3. Real-world Experiment

In the real-world experiments, we tested the whole pipeline presented in Section 8.2.1. We used three different kitchen utensils (plate, bowl, and cup) with different target

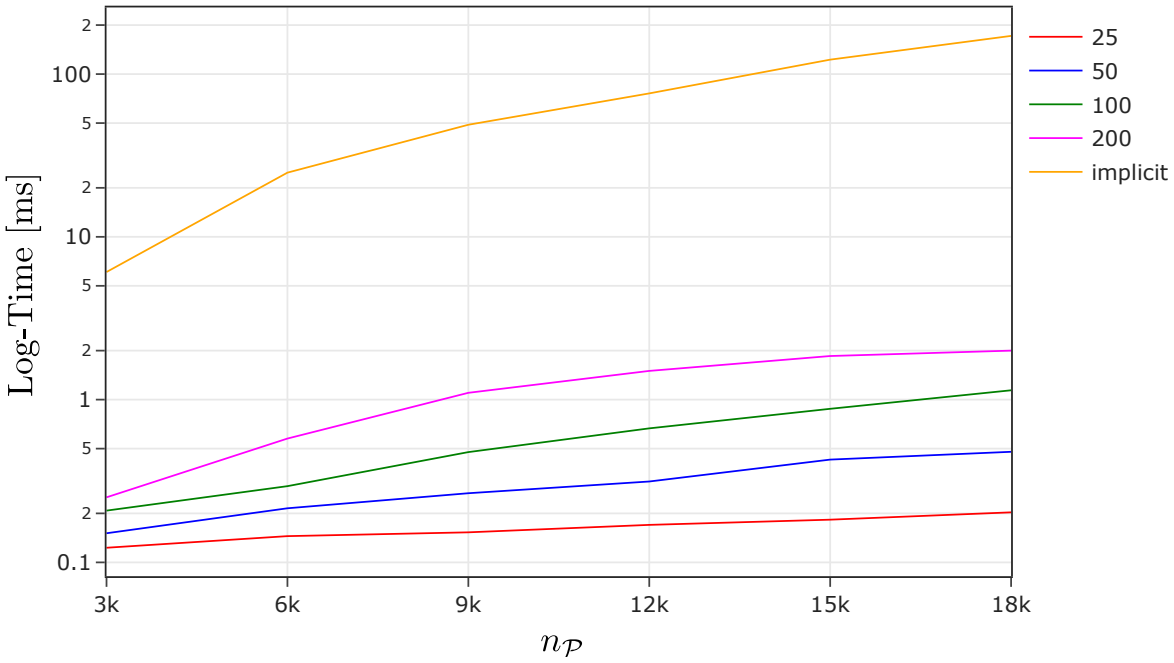


Figure 10.5.: Computational complexity of integrating the diffusion equation at runtime for different $n_{\mathcal{P}}$ and n_M. The time axis is logarithmic and the legend shows n_M values.

distributions (shapes, RLI, X). For these experiments, we fixed the objects to the table so that they could not move when the robot was in contact. At the beginning of the experiments, we moved the robot to a predefined joint configuration that fully captured the target distribution. Since we collected the point cloud data from a single image frame, our method only had access to a partial and noisy point cloud. We summarize the results of the real-world experiments in Figure 10.10 and share all the recorded experiment data and the videos on the accompanying website.

10.5.4. Comparisons

We present the first tactile ergodic coverage method in the literature that works on curved surfaces. Therefore, there are no methods that we can directly compare to quantitatively. For this reason, we selected three related state-of-the-art methods and compared them to our method qualitatively. As the first method, we selected the finite element based HEDAC planner [107], since it is the only other ergodic control approach working on curved surfaces. For the tactile interaction aspect, we selected two methods, the unified force-impedance control [119] and the sampling-based informative path planner [115]. We specified six criteria for comparison and summarized the results in Table 10.1.

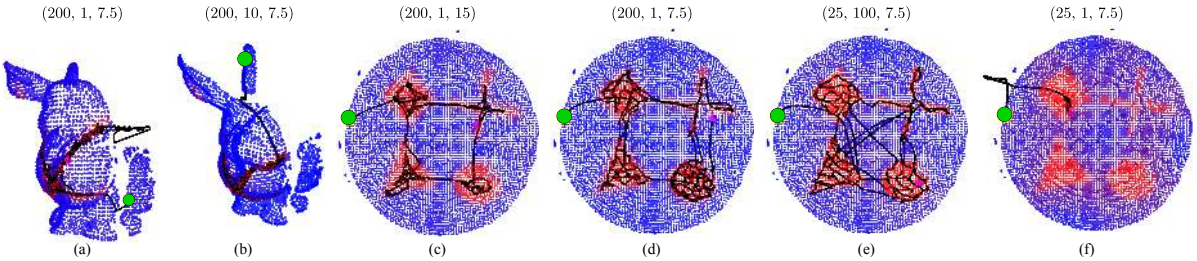


Figure 10.6.: Qualitative results of the coverage experiments showcasing the effect of different parameters. The red points designate the spatial target distribution $p_i > 0$. The agent starts at the green point, the trajectory is shown in black, and the final position after 1000 timesteps is shown with the purple point. The tuples given on top of the figures show the parameters $n_{\mathcal{K}}$, α, and r_a of the experiments. We provide the interactive point clouds and the experiment data on our website.

10.6. Discussion

10.6.1. Computational Performance

We investigated the computational performance of our method for the preprocessing and for the runtime.

The preprocessing step is only required, when the robot sees an object for the first time or when the object undergoes a non-isometric transformation. First thing to note from Figure 10.4 is that computing the eigenbasis is significantly faster than inverting the large sparse matrix. Secondly, the advantage of the spectral approach becomes more significant as the number of points increases. This is because the computational complexity of the spectral approach is linear $\mathcal{O}(n_{\mathcal{P}} n_M)$ with the number of points, whereas the matrix inversion of the implicit solution has quadratic complexity $\mathcal{O}(n_{\mathcal{P}}^2)$.

If we compare our method with the state-of-the-art in ergodic coverage on curved surfaces [107], our preprocessing step is significantly faster. They reported a computation time of 19.7 s for a mesh with 2315 points using a finite-element-based method. In contrast, our method takes 278 ms for a point cloud with ≈ 3000 points with $n_M = 100$. Therefore, in comparison, our method promises an increase in computation speed of more than 90 times. Note that, as the number of points increases, our gains in computation time become even more significant due to the difference in the computational complexity of the spectral and implicit formulations as mentioned above.

As Figure 10.5 shows the spectral approach also results in a significant performance increase at runtime. The implicit solution is also efficient in runtime, since it reduces to matrix-vector multiplication after inverting the sparse matrix at the preprocessing step. Nevertheless, the spectral formulation is still significantly faster than the implicit formulation, especially for large point clouds.

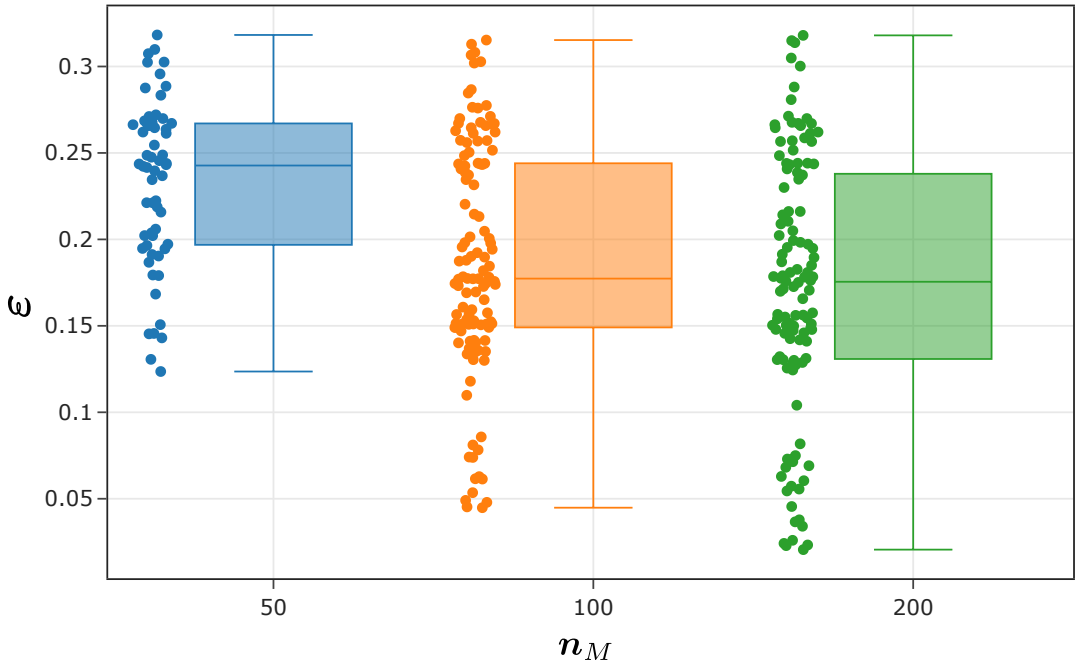


Figure 10.7.: Coverage performance measured by the ergodic metric ε_t (10.46) with respect to n_M used in the spectral formulation (10.15).

Obviously, an unnecessarily large eigenbasis for small point clouds, i.e. $n_M \to n_{\mathcal{P}}$, would cause the spectral approach to be slower than the implicit one.

10.6.2. Coverage Performance

A close investigation of the failure scenarios in Figure 10.6 revealed that they stem from the bad coupling of the parameters and from an initialization of the agent far away from the source. If the agent is not far away from the source, setting low values for α might actually lead to desirable properties such as prioritizing local coverage which would in turn minimize the distance traveled during coverage. Hence, for getting the best behavior, α can be set adaptively or sequentially. For instance, it is better to use high α values at the start for robustness to bad initializations and to decrease it as the coverage advances to prioritize local coverage and to increase the performance.

We measured the effect of our method parameters on the coverage performance in Figures 10.7 and 10.8. Interestingly, the parameters influencing the agent's speed, i.e. $\dot{\boldsymbol{x}}_{max}$, n_M and α, have a coupled effect on the coverage performance in some of the scenarios. The first thing to note here is that the value of the α is lower-bounded by the speed of the coverage agent $\dot{\boldsymbol{x}}_{max}$. Otherwise, the method cannot guide the agent since it moves faster than the diffusion. For instance, we observe from Figure 10.6 a) and f) that with a diffusion coefficient $\alpha = 1$, the source information does not propagate fast

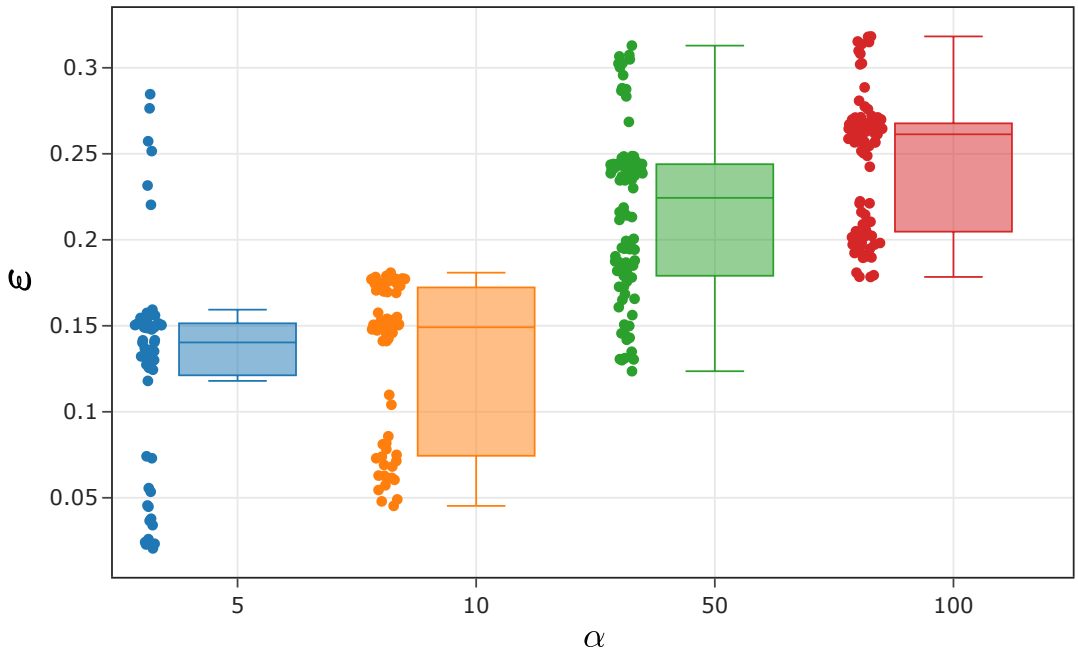


Figure 10.8.: Coverage performance measured by the normalized ergodic metric ε_t (10.46) with respect to the parameter α.

enough to the agent if it is too far from the source. Even for a small eigenbasis $n_M \leq 50$ and moderate diffusion coefficient values $1 < \alpha \leq 10$, it still results in a low coverage performance $\varepsilon_t > 0.5$. On the contrary, if the eigenbasis is chosen to be sufficiently large $n_M \geq 100$, we have more freedom in choosing α.

With this in mind, we removed the infeasible parameter combinations ($n_M = 50, \alpha = \{5, 10\}$) from the experiment results in Figures 10.7 and 10.8 to better observe the performance trend for n_M and α. It is easy to see that increasing n_M results in increased performance and higher freedom in choosing α. However, this benefit becomes marginal after $n_M \geq 100$. Therefore, choosing $n_M = 100$ becomes a good trade-off between coverage performance and computational complexity. This observation is in line with the value of $n_M = 128$ reported in [190].

In Figure 10.8, however, we observed minor differences in performance for different α. Considering the spread and the mean, choosing $\alpha = 10$ would be a good fit for most scenarios. Nevertheless, we must admit that the ergodic metric falls short in distinguishing the most significant differences between α values. Hence, the qualitative performance shown in Figure 10.6 becomes much more explanatory. The first thing to note here is that the lower values of α result in more local coverage, whereas higher values lead to prioritizing global coverage. Accordingly, the tuning of this parameter depends on the task itself. For example, suppose the goal is to collect measurements from different modes of a target distribution as quickly as possible, in which case we

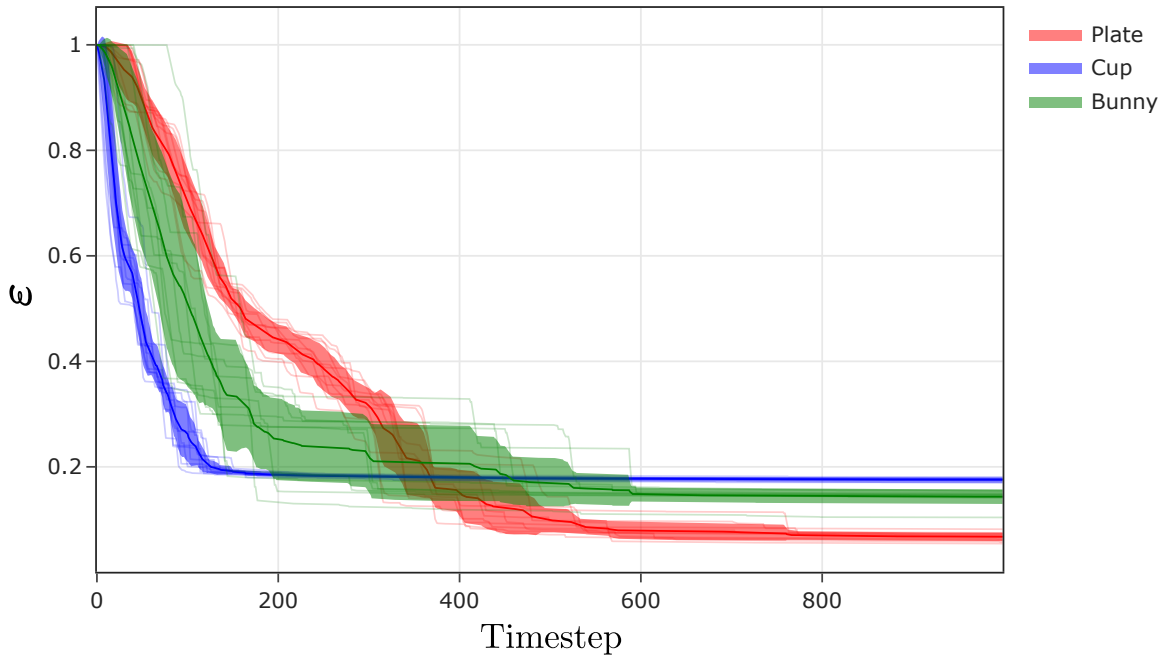


Figure 10.9.: Time evolution of the ergodic metric (10.46) for three different objects with $n_M = 200$ and $\alpha = 10$. The semi-transparent lines show ten different experiment runs, the center line shows the mean, and the shaded regions correspond to the standard deviation.

would recommend using $\alpha > 50$. On the other hand, if the surface motion is costly, because for example, the surface is prone to damage, moving less frequently between the modes can be achieved by setting $5 < \alpha < 50$.

In scenarios where the physical interactions are complex, stopping the coverage prematurely and observing the actual coverage might be preferable instead of continuing the coverage. To decide when to actually pause and measure the current coverage, we investigated the time evolution of the coverage performance in Figure 10.9. For the cup and the bunny, we see that the coverage reaches a steady state around the 200-th timestep, while for the plate, this occurs around the 500-th timestep. Still, we can identify the steepest increase in the coverage occurring until the 150-th timestep. Accordingly, we recommend the strategy to pause the coverage at roughly 200 timesteps, measure the actual coverage, and continue the coverage. This would potentially help in the cases where we have unconnected regions (various modes), because discontinuous jumps between the disjoint regions might be quicker and easier than following the surface. All that said, these claims require further testing and experimentation, which are left to be investigated in future work.

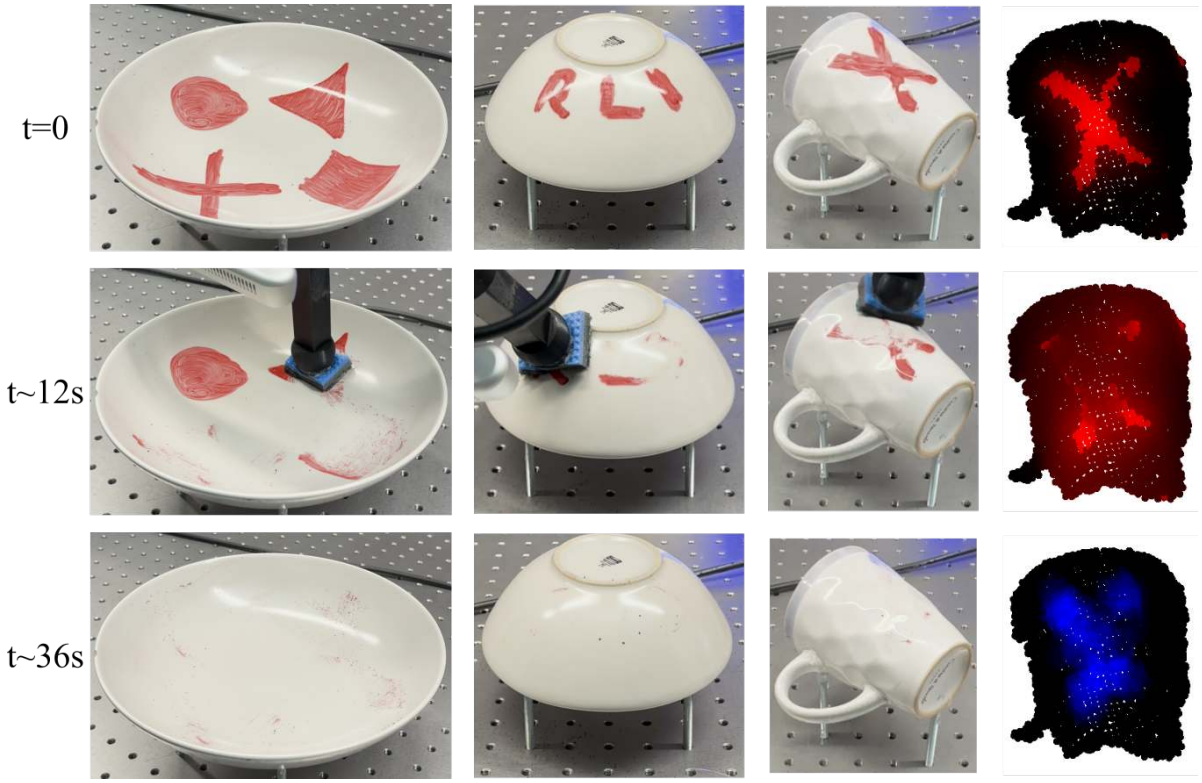


Figure 10.10.: Real-world experiment of the robot cleaning a plate, a bowl, and a cup. For the first three columns we give snapshots from the initial, intermediate, and final states from top to the bottom. In the last column, we show the target distribution $\boldsymbol{p}$, the simulated potential field $\boldsymbol{u}_t$ and the coverage $\boldsymbol{c}_t$ from top to the bottom.

10.6.3. Force Control

We demonstrated that the proposed method can perform closed-loop tactile ergodic control in the real world with unknown objects and target distributions, as depicted in Figure 10.10. The primary challenge, however, is to be keeping in contact with the surface without applying excessive force. This is mainly due to the insufficient depth accuracy of the camera, and uncertain dimensions of the mechanical system. A suboptimal solution is to use a compliant controller and adjust the penetration depth of the impedance target. A too compliant controller would, however, reduce the tracking precision and the uncertainty in the penetration depth could lead to unnecessarily high contact forces that might damage the object. More importantly, high contact forces result in high friction that further reduces the reference tracking performance.

Our solution to this problem was to introduce tactile feedback from the wrist-mounted force and torque sensor and closed-loop tracking of a reference contact force. In general, the commands generated by the force controllers conflict with the position controllers

Table 10.1.: Comparison of the proposed method with state-of-the-art methods: Finite element-based HEDAC [107], Sampling-based Planner [115], Unified Force-Impedance Control [119], and Tactile Ergodic Control (Ours). Acronyms: VI (Visual Inspection), TC (Tactile Coverage), SE (Surface Exploration).

Method	[107]	[115]	[119]	Ours
Domain	Mesh	Mesh	None	Point Cloud [a]
Approach	Planning	Planning	Control	Control
Online	No	Yes	Yes	Yes
Purpose	VI	TC	SE	TC
Multiscale	Yes	No	No	Yes
Multisetup [b]	No	Yes	No	No

[a] Since point clouds are the most general representation, our methoud can seamlessly be used on grids/meshes with only minor changes to the computation of the discrete Laplacian.

[b] Multisetup used by [115] refers to planning the configuration of the target object to reach otherwise unreachable regions.

and result in competing objectives. We overcome this problem by posing the objective as line tracking instead of position tracking. This forces the agent to be on the line but free to move along the line. Accordingly, the force and the line controller can simultaneously be active without conflicting objectives or rigorous parameter tuning.

10.6.4. Comparisons

We compared our method with state-of-the-art approaches in Table 10.1. Since the methods are not comparable in all aspects, we discuss the advantages and disadvantages of our method in three parts: (i) ergodic coverage; (ii) tactile interactions; and (iii) tactile coverage.

Ergodic Coverage

In the literature, the only other ergodic coverage method on curved surfaces is the finite element-based HEDAC [107]. This work presents an offline planning method on meshes for visual inspection using multiple aerial vehicles. Accordingly, our method extends the state of the art in ergodic coverage on curved surfaces by being the first formulation (i) working on point clouds, (ii) providing closed-loop coverage using vision, and (iii) performing tactile coverage. Furthermore, as we showed in the experiments in Section 10.5.2, our approach vastly outscales the finite element-based HEDAC in terms of computation time for the preprocessing step. It is also important to note that, due

to the generality of the underlying ergodic control formulation that we are using, our method could be applied to their use-case as well.

Tactile Interactions

To ensure contact during tactile exploration, the usage of a unified force-impedance control scheme was proposed [119]. The general idea is similar to ours, in the sense that the controller is required to track a given reference while exerting a force on the surface. The main difference stems from the formulation of the reference for the impedance behavior. While their method tracks a full Cartesian pose, our impedance controller tracks a line. The main difference here is that our method imposes fewer constraints on the reference tracking, which leaves more degrees of freedom for secondary tasks, such as tracking the force objective. Hence, we require no additional tuning to integrate these objectives, whereas their method uses a passivity-based design to ensure the stability of the combined controller.

Tactile Coverage

Concerning the problem of using a manipulator for tactile coverage on curved surfaces, we compare our method to the online sampling-based planner presented in [115]. Unlike the more general point cloud representation that we are using, this method operates on meshes. However, it includes the planning of the configuration of the target object. This is currently a limitation of our approach, since we assume the object to be fixed and consider only a single viewpoint. Although this configuration planner is considered to be independent of the coverage at a given configuration, it could be easily combined with our method. In contrast to our myopic feedback controller, they use trajectory planning, which requires a predefined planning horizon using a number of passes for covering discrete patches. For tactile coverage tasks, this can be extremely challenging to estimate beforehand. Our method does not suffer from this limitation, since ergodicity guarantees revisiting continuous areas according to the target distribution over an infinite time horizon. In addition, their approach is based on generating splines that connect the waypoints. This has two issues: if the points are not densely sampled, there is no guarantee that the resulting spline would be on the surface; and conversely, if the points are densely sampled, then the spline would be very complex and not smooth. Accordingly, this approach would not scale to complex surfaces and target distributions. Our approach, on the other hand, uses a feedback controller to stay in contact with the surface, where the local references are coming from the surface-constrained ergodic controller. Hence, our approach is mainly limited by the robot's geometry with respect to the complexity of the object, which could also be mitigated by changing its configuration online.

10.7. Conclusion

In this chapter, we presented the first closed-loop ergodic coverage method on point clouds to address the tactile coverage tasks on curved surfaces. Tactile coverage tasks are challenging to model due to complex physical interactions. We use vision to jointly capture the surface geometry and the target distribution as a point cloud and directly use this representation as input. Then, we propagate the information regarding the coverage target to our robot using a diffusion process on the point cloud. Here, we use ergodicity to relate the spatial distribution to the number of visits required for coverage in an infinite-horizon formulation. We leverage a spectral formulation to trade-off the accuracy of the diffusion computation with its computational complexity. To find a favorable compromise between the two, we tested the dependency of the coverage performance to the hyperparameters in kinematic simulation experiments. Next, we demonstrated the method in a real-world setting by cleaning previously unknown curved surfaces with arbitrary human-drawn distributions. We observed that our method can indeed adapt and generalize to different object shapes and distributions on the fly.

Our method assumes that it is possible to

Additionally, in some tactile coverage scenarios it is not straightforward to measure the actual coverage using an RGB-D camera such as surface inspection, sanding, or mechanical palpation. Still, we can use cleaning as a proxy task such that a human expert can mark the regions that need to be inspected with an easy-to-remove marker. Then, the robot's progress would be detectable by a camera. Accordingly, our method provides an interesting human-robot interaction modality using annotations and markings of an expert for tactile robotics tasks.

As discussed in Section 10.6.4, a practical limitation of our setup is fixing the object pose during the operation. Therefore, we plan to extend our method to scenarios where the object is grasped by a second manipulator and can be reconfigured for covering regions that otherwise would be unreachable due to either collisions or joint limits. Although this problem is easy to address by sampling discrete configurations, as previously done in [115], our goal is to extend our method to handle this problem in a continuous manner using a control approach.

Another promising extension of our method is automating the collection of visuotactile datasets. In this setting, one can combine our method with a vision-based active learning module such as [122], which estimate high tactile-information regions on the surface. Then, our controller could be used to collect data from these regions with a multi-modal tactile sensor.

11. Conclusion

This thesis proposed using geometric algebra as a unified geometric framework for control and optimization in robot manipulation. We demonstrated the potential of GA in robotics through three interconnected parts: mathematical modeling, optimization, and practical control. In the first part, we provided the mathematical foundations. We showed how GA provides a unified framework to represent geometric primitives (points, lines, spheres) and transformations (rotations, translations, dilations) in robotics. We then showed how these can be applied to the modeling of robot kinematics and dynamics, and proposed a recursive forward dynamics algorithm in GA. Part two then tackled the geometrically coherent formulation of optimization problems for manipulation tasks. Here, we demonstrated how GA simplifies the symbolic complexity and ensures coordinate invariance. We proposed formulations based on the outer product and on similarity transformations. Both have inherent nullspace structures that allow the integration of secondary control objectives. We the extended these formulations to include probabilistic geometric primitives for robust uncertainty-aware manipulation. In the last part, we tackled the practical integration of our propositions. To this end, we provided the gafro library that facilitates the integration of GA into real robotic control systems. We used our theoretical findings in conjunction with this implementation to solve tasks around dual-arm admittance control and tactile ergodic surface coverage. While we drew conclusions for each chapter, this final chapter aims at connecting them and putting them into a broader context by mentioning limitations and possible future research directions.

11.1. Limitations

This thesis presented contributions towards capturing the intrinsic geometric structure of robotic tasks by utilizing the unifying representation of geometric algebra.- Although we demonstrated in this thesis how the modeling of robotic manipulation tasks is simplified by geometric algebra, there remain still unsolved challenges stemming from the assumptions we made. We discuss these limitations in this section.

11.1.1. End-Effector Reference

As we already briefly mentioned in the discussion in Section 6.4.4, we mainly consider the end-effector points of the robot arms as reference for modeling manipulation tasks. Using

the robot end-effector as the reference point for control is standard practice in robot manipulation. Therefore, this is not only a limitation of this thesis, but we regard it as one of the most fundamental limitations in robotics research. From a mechanical design point of view, end-effectors are made for interaction and offer attachments for tools or sensors for specific tasks. While these are valid practical considerations motivating the choice, it restricts the robot's ability to leverage its full physical structure for complex manipulation challenges. For instance, tasks like carrying big and bulky objects, where contact distribution and load-sharing across the robot's body are required. In this case, the geometric primitives should be extended to not only consider the end-effector, but to encompass the entire surface geometry of the robot arm to enable dexterous manipulation. Representations of the geometry as continuous distance functions try to address this issue by [138], and future work therefore tackle the integration of geometric primitives from GA to enable whole-body constraints for motion optimization.

11.1.2. Integration of Perception

While this thesis has advanced the application of geometric algebra to the mathematical modeling and optimization of robotic arm movements, a notable limitation lies in its partial integration of perception within the broader control framework. For example, in the experiments where vision was used, we either used Aruco markers to define the exact location of geometric primtivers, as in Chapter 4, or assumed static scenarios, where discrete feedback at irregular intervals was sufficient, as in Chapter 10. Albeit in Chapters 9 and 10 wrist-mounted force-torque sensors were used, the system's capacity to handle contact-rich tasks is limited. These sensors excel at measuring interaction forces, but lack the spatial resolution of tactile sensor or the contextual awareness vision-based perception. Neglecting, perception technologies, such as vision, LiDAR, or tactile sensing, assumes operation under idealized environment conditions. This limits the applicability of our proposed control models to controlled laboratories and prevents deployment in unstructured or dynamically changing settings. Environments where object positions, shapes, or obstacles are not predefined require visual feedback to determine the geometric primitives that approximate the task geometry. Furthermore, noisy perception and environmental variability introduce uncertainties on the task geometries. Chapter 7 provided a first step towards the integration of uncertainty into the GA-based modeling of manipulation tasks. Future work should therefore extend this approach for a better integration of perception into the proposed control pipelines. This would further demonstrate GA's potential to unify perceptual and control variables within a single mathematical language.

11.2. Future Work

In the previous section, we highlighted the limitations of the work presented and this thesis. We then proposed future work for addressing these limitations. We want to discuss possible future research direction more generally. Hence, the topics in this section contain ideas that build directly on our proposed methods.

11.2.1. Object-Centric Articulation Models

Many objects that humans interact with on a daily basis, such as doors, drawers and bottles, have articulations that define their functionality. While humans are capable of naturally discovering how to manipulate these objects in order to use them, obtaining these skills poses a challenging problem for robots. Understanding the underlying kinematics is, however, paramount for robots to be able to manipulate them and not have unsafe forceful interactions in human environments. Here, geometric algebra could be used for uniformly modeling the involved geometric primitives that determine the motion of the articulation. One approach could involve the orbit of twists [69], i.e. the shape of the surface that a point is moving on given a constant twist. Both the twist and the orbit are algebraic objects within CGA and can thus be used for mathematical computations and directly be integrated into our presented control methods.

11.2.2. Robot Learning

This thesis has lain the foundations for the mathematical modeling with geometric algebra for optimization and control in robot manipulation tasks. Future work should aim at integrating the flexibility of learning approaches into this framework. Learning approaches in general are highly connected to statistics. In Section 7.5.1, we already hinted at the integration of learning from demonstration methods via representing the variations through the covariance propagated to the geometric primitives. Here, the desired geometric primitives could be directly learned from variations in task demonstrations. Robust control for compliant manipulation could then be ensured via adapting impedance parameters based on the covariance. Learning from demonstration is not the only learning modality that offers interesting research opportunities for geometric algebra. Incorporating the presented GA approaches into deep learning could enable neural networks to inherently respect geometric symmetries. For instance, the similarity transformations presented in Chapter 6 could be further exploited to generalize skills by learning equivariant networks, e.g. $SIM(3)$-equivariant models [218]. $E(3)$-equivariance using geometric algebra transformers has already been shown to outperform baselines [36]. Merging these approaches to enable visuomotor control using GA to encode points, lines, and spheres as algebraic entities, simplifying sensorimotor transformations pro-

vides a promising research direction. Another promising learning methodology are diffusion models, where GA could naturally incorporate geometric relationships, e.g. screw motions for dexterous manipulation or latent constraints as geometric primitives, for efficient manipulation skill synthesis. Connecting this with reinforcement learning could then provide lifelong adaptation, where skill acquisition is treated as stochastic exploration of multivector spaces.

11.3. Vision

My vision is that robots will infer generalized skills from geometric first principles, rather than engineering customized solutions for every task. Geometric structures are the unifying mathematical foundation for learning, perception, control, and optimization. They will enable robots to generalize skills seamlessly across tasks, environments, and physical embodiments. The key lies in recognizing the correct embedding spaces and non-linear manifolds that properly represent and explain the desired skills. For instance, three-dimensional objects cannot be explained from two dimensions, since they will change their shape or even disappear depending on the projection plane. Therefore, I believe that also in robotics, higher dimensional manifolds allow perspectives that will reveal hidden connections for generalizing robot skills. Similar to how screws and twists provide invariant representations for classical robotics, modeling these skill manifolds can systematize skill transfer, composition, and adaptation. Dimension-agnostic geometry is essential to understanding these connections and to unlocking their full potential. This is why I believe that geometric algebra provides the ideal mathematical language for uncovering this understanding. Its geometric, coordinate-free formulations seamlessly generalize to arbitrary dimensions.

Bibliography

[1] Ian Abraham, Ahalya Prabhakar, Mitra J. Z. Hartmann, and Todd D. Murphey. "Ergodic Exploration Using Binary Sensing for Non-Parametric Shape Estimation". In: *IEEE Robotics and Automation Letters* 2.2 (Apr. 2017), pp. 827–834. DOI: 10.1109/LRA.2017.2654542.

[2] E.U. Acar, H. Choset, and Ji Yeong Lee. "Sensor-Based Coverage with Extended Range Detectors". In: *IEEE Transactions on Robotics* 22.1 (Feb. 2006), pp. 189–198. DOI: 10.1109/TRO.2005.861455.

[3] Ercan U. Acar, Howie Choset, Alfred A. Rizzi, Prasad N. Atkar, and Douglas Hull. "Morse Decompositions for Coverage Tasks". In: *The International Journal of Robotics Research* 21.4 (Apr. 2002), pp. 331–344. DOI: 10.1177/027836402320556359.

[4] Bruno Vilhena Adorno, Philippe Fraisse, and Sébastien Druon. "Dual Position Control Strategies Using the Cooperative Dual Task-Space Framework". In: *IEEE International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2010, pp. 3955–3960. DOI: 10.1109/IROS.2010.5650218.

[5] Bruno Vilhena Adorno and Murilo Marques Marinho. "DQ Robotics: A Library for Robot Modeling and Control". In: *IEEE Robotics Automation Magazine* 28.3 (Sept. 2021), pp. 102–116. DOI: 10.1109/MRA.2020.2997920.

[6] Frederico Fernandes Afonso Silva, Juan José Quiroz-Omaña, and Bruno Vilhena Adorno. "Dynamics of Mobile Manipulators Using Dual Quaternion Algebra". In: *Journal of Mechanisms and Robotics* 14.6 (Sept. 2022). DOI: 10.1115/1.4054320.

[7] Victor Aladele, Carlos R. De Cos, Dimos V. Dimarogonas, and Seth Hutchinson. "An Adaptive Cooperative Manipulation Control Framework for Multi-Agent Disturbance Rejection". In: *IEEE Conference on Decision and Control (CDC)*. Dec. 2022, pp. 100–106. DOI: 10.1109/CDC51059.2022.9992478.

[8] Diogo Almeida and Yiannis Karayiannidis. "Cooperative Manipulation and Identification of a 2-DOF Articulated Object by a Dual-Arm Robot". In: *IEEE International Conference on Robotics and Automation (ICRA)*. May 2018, pp. 5445–5451. DOI: 10.1109/ICRA.2018.8460511.

[9] Walid Amanhoud, Mahdi Khoramshahi, and Aude Billard, eds. *A Dynamical System Approach to Motion and Force Generation in Contact Tasks*. Robotics: Science and Systems (RSS), 2019.

[10] Jason Ansel et al. "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation". In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. ACM, Apr. 2024, pp. 929–947. DOI: 10.1145/3620665.3640366.

[11] Carlos A. Arellano-Muro, Guillermo Osuna-González, Bernardino Castillo-Toledo, and Eduardo Bayro-Corrochano. "Newton–Euler Modeling and Control of a Multicopter Using Motor Algebra". In: *Advances in Applied Clifford Algebras* 30.2 (Apr. 2020), p. 19. DOI: 10.1007/s00006-020-1045-1.

[12] Andreas Aristidou, Yiorgos Chrysanthou, and Joan Lasenby. "Extending FABRIK with Model Constraints". In: *Computer Animation and Virtual Worlds* 27.1 (Jan. 2016), pp. 35–57. DOI: 10.1002/cav.1630.

[13] Andreas Aristidou and Joan Lasenby. "Inverse Kinematics Solutions Using Conformal Geometric Algebra". In: *Guide to Geometric Algebra in Practice*. Ed. by Leo Dorst and Joan Lasenby. London: Springer London, 2011, pp. 47–62. DOI: 10.1007/978-0-85729-811-9_3.

[14] P.N. Atkar, D.C. Conner, A. Greenfield, H. Choset, and A.A. Rizzi. "Hierarchical Segmentation of Piecewise Pseudoextruded Surfaces for Uniform Coverage". In: *IEEE Transactions on Automation Science and Engineering* 6.1 (Jan. 2009), pp. 107–120. DOI: 10.1109/TASE.2008.916768.

[15] Alp Aydinoglu and Michael Posa. "Real-Time Multi-Contact Model Predictive Control via ADMM". In: *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*. May 2022, pp. 3414–3421. DOI: 10.1109/ICRA46639.2022.9811957.

[16] Elif Ayvali, Rangaprasad Arun Srivatsan, Long Wang, Rajarshi Roy, Nabil Simaan, and Howie Choset. "Using Bayesian Optimization to Guide Probing of a Flexible Environment for Simultaneous Registration and Stiffness Mapping". In: *IEEE International Conference on Robotics and Automation (ICRA)*. May 2016, pp. 931–936. DOI: 10.1109/ICRA.2016.7487225.

[17] Robert S. (Robert Stawell) Ball. *A Treatise on the Theory of Screws*. Cambridge : University Press, 1900.

[18] Eduardo Bayro-Corrochano. "A Survey on Quaternion Algebra and Geometric Algebra Applications in Engineering and Computer Science 1995–2020". In: *IEEE Access* 9 (2021), pp. 104326–104355. DOI: 10.1109/ACCESS.2021.3097756.

[19] Eduardo Bayro-Corrochano. *Geometric Algebra Applications Vol. II: Robot Modelling and Control*. Cham: Springer International Publishing, 2020. DOI: 10.1007/978-3-030-34978-3.

[20] Eduardo Bayro-Corrochano. "Robot Modeling and Control Using the Motor Algebra Framework". In: *International Workshop on Robot Motion and Control (RoMoCo)*. July 2019, pp. 1–8. DOI: 10.1109/RoMoCo.2019.8787386.

[21] Eduardo Bayro-Corrochano, Ana M. Garza-Burgos, and Juan L. Del-Valle-Padilla. "Geometric Intuitive Techniques for Human Machine Interaction in Medical Robotics". In: *International Journal of Social Robotics* 12.1 (Jan. 2020), pp. 91–112. DOI: 10.1007/s12369-019-00545-8.

[22] Eduardo Bayro-Corrochano and Detlef Kähler. "Motor Algebra Approach for Computing the Kinematics of Robot Manipulators". In: *Journal of Robotic Systems* 17.9 (2000), pp. 495–516. DOI: 10.1002/1097-4563(200009)17:9<495::AID-ROB4>3.0.CO;2-S.

[23] Eduardo Bayro-Corrochano, Jesus Medrano-Hermosillo, Guillermo Osuna-González, and Ulises Uriostegui-Legorreta. "Newton–Euler Modeling and Hamiltonians for Robot Control in the Geometric Algebra". In: *Robotica* 40.11 (Nov. 2022), pp. 4031–4055. DOI: 10.1017/S0263574722000741.

[24] Eduardo Bayro-Corrochano and Julio Zamora-Esquivel. "Differential and Inverse Kinematics of Robot Devices Using Conformal Geometric Algebra". In: *Robotica* 25.1 (Jan. 2007), pp. 43–61. DOI: 10.1017/S0263574706002980.

[25] Patrick Beeson and Barrett Ames. "TRAC-IK: An Open-Source Library for Improved Solving of Generic Inverse Kinematics". In: *IEEE International Conference on Humanoid Robots (Humanoids)*. Seoul, South Korea, Nov. 2015, pp. 928–935. DOI: 10.1109/HUMANOIDS.2015.7363472.

[26] Hadi Beik-Mohammadi, Søren Hauberg, Georgios Arvanitidis, Gerhard Neumann, and Leonel Rozo. "Reactive Motion Generation on Learned Riemannian Manifolds". In: *The International Journal of Robotics Research* (Aug. 2023), p. 02783649231193046. DOI: 10.1177/02783649231193046.

[27] Mikhail Belkin, Jian Sun, and Yusu Wang. "Constructing Laplace Operator from Point Clouds in R^d". In: *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, Jan. 2009, pp. 1031–1040. DOI: 10.1137/1.9781611973068.112.

[28] Mauricio Cele Lopez Belon. "Applications of Conformal Geometric Algebra in Mesh Deformation". In: *2013 XXVI Conference on Graphics, Patterns and Images*. Arequipa, Peru: IEEE, Aug. 2013, pp. 39–46. DOI: 10.1109/SIBGRAPI.2013.15.

[29] Werner Benger and Wolfgang Dobler. "Massive Geometric Algebra: Visions for C++ Implementations of Geometric Algebra to Scale into the Big Data Era". In: *Advances in Applied Clifford Algebras* 27.3 (Sept. 2017), pp. 2153–2174. DOI: 10.1007/s00006-017-0780-4.

[30] Thomas A. Berrueta, Allison Pinosky, and Todd D. Murphey. *Maximum Diffusion Reinforcement Learning*. Sept. 2023. DOI: 10.48550/arXiv.2309.15293.

[31] Cem Bilaloglu, Tobias Löw, and Sylvain Calinon. "Tactile Ergodic Coverage on Curved Surfaces". In: *IEEE Transactions on Robotics* (2025).

[32] Cem Bilaloglu, Tobias Löw, and Sylvain Calinon. "Whole-Body Ergodic Exploration with a Manipulator Using Diffusion". In: *IEEE Robotics and Automation Letters* (2023), pp. 1–7. DOI: 10.1109/LRA.2023.3329755.

[33] Walter G. Bircher, Andrew S. Morgan, and Aaron M. Dollar. "Complex Manipulation with a Simple Robotic Hand through Contact Breaking and Caging". In: *Science Robotics* 6.54 (May 2021), eabd2666. DOI: 10.1126/scirobotics.abd2666.

[34] Riccardo Bonalli, Abhishek Cauligi, Andrew Bylard, Thomas Lew, and Marco Pavone. "Trajectory Optimization on Manifolds: A Theoretically-Guaranteed Embedded Sequential Convex Programming Approach". In: *Robotics: Science and Systems XV*. Robotics: Science and Systems Foundation, June 2019. DOI: 10.15607/RSS.2019.XV.078.

[35] Silvere Bonnabel, Philippe Martin, and Erwan Salaun. "Invariant Extended Kalman Filter: Theory and Application to a Velocity-Aided Attitude Estimation Problem". In: *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) Held Jointly with 2009 28th Chinese Control Conference*. Shanghai: IEEE, Dec. 2009, pp. 1297–1304. DOI: 10.1109/CDC.2009.5400372.

[36] Johann Brehmer, Pim de Haan, Sönke Behrends, and Taco S. Cohen. "Geometric Algebra Transformer". In: *Advances in Neural Information Processing Systems* 36 (Dec. 2023), pp. 35472–35496.

[37] Stephane Breuils, Vincent Nozick, and Eckhard Hitzer. "Quadric Conformal Geometric Algebra of R9,6". In: (), p. 15.

[38] Stephane Breuils, Kanta Tachibana, and Eckhard Hitzer. "New Applications of Clifford's Geometric Algebra". In: *Advances in Applied Clifford Algebras* 32.2 (Apr. 2022), p. 17. DOI: 10.1007/s00006-021-01196-7.

[39] Stéphane Breuils, Vincent Nozick, and Laurent Fuchs. "Garamon: A Geometric Algebra Library Generator". In: *Advances in Applied Clifford Algebras* 29.4 (July 2019), p. 69. DOI: 10.1007/s00006-019-0987-7.

[40] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. "Geometric Deep Learning: Going beyond Euclidean Data". In: *IEEE Signal Processing Magazine* 34.4 (July 2017), pp. 18–42. DOI: 10.1109/MSP.2017.2693418.

[41] Francesco Bullo and Andrew D. Lewis. *Geometric Control of Mechanical Systems: Modeling, Analysis, and Design for Simple Mechanical Control Systems*. Vol. 49. Texts in Applied Mathematics. New York, NY: Springer, 2005. DOI: 10.1007/978-1-4899-7276-7.

[42] Fabrizio Caccavale, Ciro Natale, Bruno Siciliano, and Luigi Villani. "Six-DOF Impedance Control Based on Angle/Axis Representations". In: *IEEE Transactions on Robotics and Automation* 15.2 (Apr. 1999), pp. 289–300. DOI: 10.1109/70.760350.

[43] Fabrizo Caccavale, Pasquale Chiacchio, Alessandro Marino, and Luigi Villani. "Six-DOF Impedance Control of Dual-Arm Cooperative Manipulators". In: *IEEE/ASME Transactions on Mechatronics* 13.5 (Oct. 2008), pp. 576–586. DOI: 10.1109/TMECH.2008.2002816.

[44] Maya Cakmak and Leila Takayama. "Towards a Comprehensive Chore List for Domestic Robots". In: *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. Tokyo, Japan: IEEE, Mar. 2013, pp. 93–94. DOI: 10.1109/HRI.2013.6483517.

[45] Sylvain Calinon. "Gaussians on Riemannian Manifolds: Applications for Robot Learning and Adaptive Control". In: *IEEE Robotics Automation Magazine* 27.2 (June 2020), pp. 33–45. DOI: 10.1109/MRA.2020.2980548.

[46] Daniel Canedo, Pedro Fonseca, Petia Georgieva, and António J. R. Neves. "A Deep Learning-Based Dirt Detection Computer Vision System for Floor-Cleaning Robots with Improved Data Collection". In: *Technologies* 9.4 (Dec. 2021), p. 94. DOI: 10.3390/technologies9040094.

[47] Junjie Cao, Andrea Tagliasacchi, Matt Olson, Hao Zhang, and Zhinxun Su. "Point Cloud Skeletons via Laplacian Based Contraction". In: *2010 Shape Modeling International Conference*. Aix-en-Provence, France: IEEE, June 2010, pp. 187–197. DOI: 10.1109/SMI.2010.25.

[48] Nicole E. Carey and Justin Werfel. "A Force-Mediated Controller for Cooperative Object Manipulation with Independent Autonomous Robots". In: *Distributed Autonomous Robotic Systems*. Ed. by Julien Bourgeois et al. Vol. 28. Cham: Springer Nature Switzerland, 2024, pp. 140–155. DOI: 10.1007/978-3-031-51497-5_11.

[49] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. *Emerging Properties in Self-Supervised Vision Transformers*. May 2021.

[50] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. "The Pinocchio C++ Library : A Fast and Flexible Implementation of Rigid Body Dynamics Algorithms and Their Analytical Derivatives". In: *2019 IEEE/SICE International Symposium on System Integration (SII)*. Paris, France: IEEE, Jan. 2019, pp. 614–619. DOI: 10.1109/SII.2019.8700380.

[51] Preetham Chalasani, Long Wang, Rajarshi Roy, Nabil Simaan, Russell H. Taylor, and Marin Kobilarov. "Concurrent Nonparametric Estimation of Organ Geometry and Tissue Stiffness Using Continuous Adaptive Palpation". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. May 2016, pp. 4164–4171. DOI: 10.1109/ICRA.2016.7487609.

[52] Ching-An Cheng, Mustafa Mukadam, Jan Issac, Stan Birchfield, Dieter Fox, Byron Boots, and Nathan Ratliff. "RMPflow: A Geometric Framework for Generation of Multitask Motion Policies". In: *IEEE Transactions on Automation Science and Engineering* 18.3 (July 2021), pp. 968–987. DOI: 10.1109/TASE.2021.3053422.

[53] Chih-Hung King, T L Chen, A Jain, and C C Kemp. "Towards an Assistive Robot That Autonomously Performs Bed Baths for Patient Hygiene". In: *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*. Taipei: IEEE, Oct. 2010, pp. 319–324. DOI: 10.1109/IROS.2010.5649101.

[54] Sachin Chitta, Jürgen Sturm, Matthew Piccoli, and Wolfram Burgard. "Tactile Sensing for Mobile Manipulation". In: *IEEE Transactions on Robotics* 27.3 (June 2011), pp. 558–568. DOI: 10.1109/TRO.2011.2134130.

[55] Howie Choset. "Coverage for Robotics – A Survey of Recent Results". In: ().

[56] Howie Choset and Philippe Pignon. "Coverage Path Planning: The Boustrophedon Cellular Decomposition". In: *Field and Service Robotics*. Ed. by Alexander Zelinsky. London: Springer London, 1998, pp. 203–209. DOI: 10.1007/978-1-4471-1273-0_32.

[57] Pablo Colapinto. "Versor: Spatial Computing with Conformal Geometric Algebra". PhD thesis. University of California at Santa Barbara, 2011.

[58] Erwin Coumans and Yunfei Bai. *PyBullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning*. 2016/2021.

[59] John Craig. *Introduction to Robotics*. Pearson, 2021.

[60] Keenan Crane, Fernando De Goes, Mathieu Desbrun, and Peter Schröder. "Digital Geometry Processing with Discrete Exterior Calculus". In: *ACM SIGGRAPH 2013 Courses*. Anaheim California: ACM, July 2013, pp. 1–126. DOI: 10.1145/2504435.2504442.

[61] Bojan Crnković, Stefan Ivić, and Mila Zovko. *Fast Algorithm for Centralized Multi-Agent Maze Exploration*. Oct. 2023.

[62] Neil T Dantam. "Robust and Efficient Forward, Differential, and Inverse Kinematics Using Dual Quaternions". In: *The International Journal of Robotics Research* 40.10-11 (Sept. 2021), pp. 1087–1105. DOI: 10.1177/0278364920931948.

[63] Cristiana Miranda de Farias, Yuri Gonçalves Rocha, Luis Felipe Cruz Figueredo, and Mariana Costa Bernardes. "Design of Singularity-Robust and Task-Priority Primitive Controllers for Cooperative Manipulation Using Dual Quaternion Representation". In: *2017 IEEE Conference on Control Technology and Applications (CCTA)*. Aug. 2017, pp. 740–745. DOI: 10.1109/CCTA.2017.8062550.

[64] Luca De Pascali, Sebastian Erhart, Luca Zaccarian, Biral Francesco, and Sandra Hirche. "A Decoupling Scheme for Force Control in Cooperative Multi-Robot Manipulation Tasks". In: *2022 IEEE 17th International Conference on Advanced Motion Control (AMC)*. Feb. 2022, pp. 243–249. DOI: 10.1109/AMC51637.2022.9729263.

[65] Loris Delafosse. *A New Approach to Screw Theory Using Geometric Algebra*. 2023.

[66] Van-Thach Do and Quang-Cuong Pham. "Geometry-Aware Coverage Path Planning for Depowdering on Complex 3D Surfaces". In: *IEEE Robotics and Automation Letters* 8.9 (Sept. 2023), pp. 5552–5559. DOI: 10.1109/LRA.2023.3296943.

[67] Dayi Dong, Henry Berger, and Ian Abraham. "Time Optimal Ergodic Search". In: *Robotics: Science and Systems XIX*. Robotics: Science and Systems Foundation, July 2023. DOI: 10.15607/RSS.2023.XIX.082.

[68] Chris Doran and Anthony Lasenby. *Geometric Algebra for Physicists*. 1st ed. Cambridge University Press, May 2003. DOI: 10.1017/CBO9780511807497.

[69] Leo Dorst. "The Construction of 3D Conformal Motions". In: *Mathematics in Computer Science* 10.1 (Mar. 2016), pp. 97–113. DOI: 10.1007/s11786-016-0250-8.

[70] Weitao Du, He Zhang, Yuanqi Du, Qi Meng, Wei Chen, Nanning Zheng, Bin Shao, and Tie-Yan Liu. "SE(3) Equivariant Graph Neural Networks with Complete Local Frames". In: *Proceedings of the 39th International Conference on Machine Learning* ().

[71] Sebastian Erhart, Dominik Sieber, and Sandra Hirche. "An Impedance-Based Control Architecture for Multi-Robot Cooperative Dual-Arm Mobile Manipulation". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Nov. 2013, pp. 315–322. DOI: 10.1109/IROS.2013.6696370.

[72] Hamed Farivarnejad and Spring Berman. "Multirobot Control Strategies for Collective Transport". In: *Annual Review of Control, Robotics, and Autonomous Systems* 5.1 (May 2022), pp. 205–219. DOI: 10.1146/annurev-control-042920-095844.

[73] R. Featherstone and D. Orin. "Robot Dynamics: Equations and Algorithms". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 1. 2000, pp. 826–834. DOI: 10.1109/ROBOT.2000.844153.

[74] Roy Featherstone. "A Beginner's Guide to 6-D Vectors (Part 1)". In: *IEEE Robotics & Automation Magazine* 17.3 (Sept. 2010), pp. 83–94. DOI: 10.1109/MRA.2010.937853.

[75] Roy Featherstone. *Rigid Body Dynamics Algorithms*. Boston, MA: Springer US, 2008. DOI: 10.1007/978-1-4899-7560-7.

[76] Roy Featherstone. *Robot Dynamics Algorithms*. Boston, MA: Springer US, 1987. DOI: 10.1007/978-0-387-74315-8.

[77] Martin L. Felis. "RBDL: An Efficient Rigid-Body Dynamics Library Using Recursive Algorithms". In: *Autonomous Robots* 41.2 (Feb. 2017), pp. 495–511. DOI: 10.1007/s10514-016-9574-0.

[78] Leandro A. F. Fernandes. "Exploring Lazy Evaluation and Compile-Time Simplifications for Efficient Geometric Algebra Computations". In: *Systems, Patterns and Data Engineering with Geometric Calculi*. Ed. by Sebastià Xambó-Descamps. Vol. 13. Cham: Springer International Publishing, 2021, pp. 111–131. DOI: 10.1007/978-3-030-74486-1_6.

[79] Mariana de Paula Assis Fonseca. "Closed-Loop Admittance and Motion Control Strategies for Safe Robotic Manipulation Tasks Subject to Contacts". PhD thesis. Universidade Federal de Minas Gerais, 2021.

[80] Mariana de Paula Assis Fonseca, Bruno Vilhena Adorno, and Philippe Fraisse. "Coupled Task-Space Admittance Controller Using Dual Quaternion Logarithmic Mapping". In: *IEEE Robotics and Automation Letters* 5.4 (Oct. 2020), pp. 6057–6064. DOI: 10.1109/LRA.2020.3010458.

[81] Mariana de Paula Assis Fonseca, Bruno Vilhena Adorno, and Philippe Fraisse. "Coupled Task-Space Admittance Controller Using Dual Quaternion Logarithmic Mapping". In: *IEEE Robotics and Automation Letters* 5.4 (Oct. 2020), pp. 6057–6064. DOI: 10.1109/LRA.2020.3010458.

[82] Daniel Fontijne. "Gaigen 2: A Geometric Algebra Implementation Generator". In: *Proceedings of the 5th International Conference on Generative Programming and Component Engineering - GPCE '06*. Portland, Oregon, USA: ACM Press, 2006, p. 141. DOI: 10.1145/1173706.1173728.

[83] Junling Fu, Ilaria Burzo, Elisa Iovene, Jianzhuang Zhao, Giancarlo Ferrigno, and Elena De Momi. "Optimization-Based Variable Impedance Control of Robotic Manipulator for Medical Contact Tasks". In: *IEEE Transactions on Instrumentation and Measurement* 73 (2024), pp. 1–8. DOI: 10.1109/TIM.2024.3372209.

[84] Jaime Gallardo-Alvarado. *Kinematic Analysis of Parallel Manipulators by Algebraic Screw Theory*. Cham: Springer International Publishing, 2016. DOI: 10.1007/978-3-319-31126-5.

[85] Jianfeng Gao, Xiaoshu Jin, Franziska Krebs, Noémie Jaquier, and Tamim Asfour. "Bi-KVIL: Keypoints-based Visual Imitation Learning of Bimanual Manipulation Tasks". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. May 2024, pp. 16850–16857. DOI: 10.1109/ICRA57147.2024.10610763.

[86] Jianfeng Gao, Zhi Tao, Noémie Jaquier, and Tamim Asfour. "K-VIL: Keypoints-Based Visual Imitation Learning". In: *IEEE Transactions on Robotics* 39.5 (Oct. 2023), pp. 3888–3908. DOI: 10.1109/TRO.2023.3286074.

[87] Wei Gao and Russ Tedrake. "kPAM 2.0: Feedback Control for Category-Level Robotic Manipulation". In: *IEEE Robotics and Automation Letters* 6.2 (Apr. 2021), pp. 2962–2969. DOI: 10.1109/LRA.2021.3062315.

[88] Francesco Giovinazzo, Francesco Grella, Marco Sartore, Manuela Adami, Riccardo Galletti, and Giorgio Cannata. "From CySkin to ProxySKIN: Design, Implementation and Testing of a Multi-Modal Robotic Skin for Human–Robot Interaction". In: *Sensors* 24.1334 (2024). DOI: 10.3390/s24041334.

[89] L. González-Jiménez, O. Carbajal-Espinosa, A. Loukianov, and E. Bayro-Corrochano. "Robust Pose Control of Robot Manipulators Using Conformal Geometric Algebra". In: *Advances in Applied Clifford Algebras* 24.2 (June 2014), pp. 533–552. DOI: 10.1007/s00006-014-0448-2.

[90] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. "A Survey of Deep Learning Techniques for Autonomous Driving". In: *Journal of Field Robotics* 37.3 (2020), pp. 362–386. DOI: 10.1002/rob.21918.

[91] Charles Gunn. "Geometric Algebras for Euclidean Geometry". In: *Advances in Applied Clifford Algebras* 27.1 (Mar. 2017), pp. 185–208. DOI: 10.1007/s00006-016-0647-0.

[92] Charles G. Gunn. *Projective Geometric Algebra: A New Framework for Doing Euclidean Geometry*. Aug. 2020.

[93] Hugo Hadfield, Sushant Achawal, Joan Lasenby, Anthony Lasenby, and Benjamin Young. "Exploring Novel Surface Representations via an Experimental Ray-Tracer in CGA". In: *Advances in Applied Clifford Algebras* 31.2 (Apr. 2021), p. 16. DOI: 10.1007/s00006-021-01117-8.

[94] Hugo Hadfield and Joan Lasenby. "Constrained Dynamics in Conformal and Projective Geometric Algebra". In: *Advances in Computer Graphics*. Ed. by Nadia Magnenat-Thalmann, Constantine Stephanidis, Enhua Wu, Daniel Thalmann, Bin Sheng, Jinman Kim, George Papagiannakis, and Marina Gavrilova. Vol. 12221. Cham: Springer International Publishing, 2020, pp. 459–471. DOI: 10.1007/978-3-030-61864-3_39.

[95] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2. edition, 17.printing. Cambridge: Cambridge Univ. Press, 2018.

[96] Yanhao He, Min Wu, and Steven Liu. "A Distributed Optimal Control Framework for Multi-Robot Cooperative Manipulation in Dynamic Environments". In: *Journal of Intelligent & Robotic Systems* 105.1 (May 2022), p. 8. DOI: 10.1007/s10846-022-01621-4.

[97] Yanhao He, Min Wu, and Steven Liu. "An Optimisation-Based Distributed Cooperative Control for Multi-Robot Manipulation with Obstacle Avoidance". In: *IFAC-PapersOnLine*. 21st IFAC World Congress 53.2 (Jan. 2020), pp. 9859–9864. DOI: 10.1016/j.ifacol.2020.12.2691.

[98] Yinmei He, Rui Wang, Xiangyang Wang, Jian Zhou, and Yi Yan. "Novel Adaptive Filtering Algorithms Based on Higher-Order Statistics and Geometric Algebra". In: *IEEE Access* 8 (2020), pp. 73767–73779. DOI: 10.1109/ACCESS.2020.2988521.

[99] Susan Hert and Sanjay Tiwari. "A Terrain-Covering Algorithm for an AUV". In: *Autonomous Robots* (1996).

[100] David Hestenes. "New Tools for Computational Geometry and Rejuvenation of Screw Theory". In: *Geometric Algebra Computing: In Engineering and Computer Science*. Ed. by Eduardo Bayro-Corrochano and Gerik Scheuermann. London: Springer, 2010, pp. 3–33. DOI: 10.1007/978-1-84996-108-0_1.

[101] David Hestenes, Garret Sobczyk, and James S. Marsh. "*Clifford Algebra to Geometric Calculus. A Unified Language for Mathematics and Physics*". In: *American Journal of Physics* 53.5 (May 1985), pp. 510–511. DOI: 10.1119/1.14223.

[102] Dietmar Hildenbrand. *Foundations of Geometric Algebra Computing*. Vol. 8. Geometry and Computing. Berlin, Heidelberg: Springer, 2013. DOI: 10.1007/978-3-642-31794-1.

[103] Dietmar Hildenbrand, Joachim Pitt, and Andreas Koch. "Gaalop—High Performance Parallel Computing Based on Conformal Geometric Algebra". In: *Geometric Algebra Computing: In Engineering and Computer Science*. Ed. by Eduardo Bayro-Corrochano and Gerik Scheuermann. London: Springer London, 2010, pp. 477–494. DOI: 10.1007/978-1-84996-108-0_22.

[104] Eckhard Hitzer, Manos Kamarianakis, George Papagiannakis, and Petr Vašík. "Survey of New Applications of Geometric Algebra". In: *Mathematical Methods in the Applied Sciences* (2023). DOI: 10.1002/mma.9575.

[105] Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. "ReKep: Spatio-Temporal Reasoning of Relational Keypoint Constraints for Robotic Manipulation". In: *Proceedings of The 8th Conference on Robot Learning*. PMLR, Jan. 2025, pp. 4573–4602.

[106] Jemin Hwangbo, Joonho Lee, and Marco Hutter. "Per-Contact Iteration Method for Solving Contact Dynamics". In: *IEEE Robotics and Automation Letters* 3.2 (Apr. 2018), pp. 895–902. DOI: 10.1109/LRA.2018.2792536.

[107] Stefan Ivić, Bojan Crnković, Luka Grbčić, and Lea Matleković. "Multi-UAV Trajectory Planning for 3D Visual Inspection of Complex Structures". In: *Automation in Construction* 147 (Mar. 2023), p. 104709. DOI: 10.1016/j.autcon.2022.104709.

[108] Stefan Ivić, Bojan Crnković, and Igor Mezić. "Ergodicity-Based Cooperative Multiagent Area Coverage via a Potential Field". In: *IEEE Transactions on Cybernetics* 47.8 (Aug. 2017), pp. 1983–1993. DOI: 10.1109/TCYB.2016.2634400.

[109] Stefan Ivić, Ante Sikirica, and Bojan Crnković. "Constrained Multi-Agent Ergodic Area Surveying Control Based on Finite Element Approximation of the Potential Field". In: *Engineering Applications of Artificial Intelligence* 116 (Nov. 2022), p. 105441. DOI: 10.1016/j.engappai.2022.105441.

[110] Henry Jacobs, Sujit Nair, and Jerrold Marsden. "Multiscale Surveillance of Riemannian Manifolds". In: *Proceedings of the 2010 American Control Conference*. June 2010, pp. 5732–5737. DOI: 10.1109/ACC.2010.5531152.

[111] Noemie Jaquier, Leonel Rozo, and Tamim Asfour. "Unraveling the Single Tangent Space Fallacy: An Analysis and Clarification for Applying Riemannian Geometry in Robot Learning". In: ().

[112] Noémie Jaquier, Leonel Rozo, Darwin G Caldwell, and Sylvain Calinon. "Geometry-Aware Manipulability Learning, Tracking, and Transfer". In: *The International Journal of Robotics Research* 40.2-3 (Feb. 2021), pp. 624–650. DOI: 10.1177/0278364920946815.

[113] Zhongliang Jiang, Nehil Danis, Yuan Bi, Mingchuan Zhou, Markus Kroenke, Thomas Wendler, and Nassir Navab. "Precise Repositioning of Robotic Ultrasound: Improving Registration-based Motion Compensation Using Ultrasound Confidence Optimization". In: *IEEE Transactions on Instrumentation and Measurement* 71 (2022), pp. 1–11. DOI: 10.1109/TIM.2022.3200360.

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

[115] Ariyan M. Kabir, Krishnanand N. Kaipa, Jeremy Marvel, and Satyandra K. Gupta. "Automated Planning for Robotic Cleaning Using Multiple Setups and Oscillatory Tool Motions". In: *IEEE Transactions on Automation Science and Engineering* 14.3 (July 2017), pp. 1364–1377. DOI: 10.1109/TASE.2017.2665460.

[116] Suhas Kadalagere Sampath, Ning Wang, Hao Wu, and Chenguang Yang. "Review on Human-like Robot Manipulation Using Dexterous Hands". In: *Cognitive Computation and Systems* 5.1 (2023), pp. 14–29. DOI: 10.1049/ccs2.12073.

[117] Aleksandra Kalinowska, Ahalya Prabhakar, Kathleen Fitzsimons, and Todd Murphey. "Ergodic Imitation: Learning from What to Do and What Not to Do". In: *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*. May 2021, pp. 3648–3654. DOI: 10.1109/ICRA48506.2021.9561746.

[118] Kubra Karacan, Hamid Sadeghian, Robin Kirschner, and Sami Haddadin. "Passivity-Based Skill Motion Learning in Stiffness-Adaptive Unified Force-Impedance Control". In: *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*. Kyoto, Japan: IEEE, Oct. 2022, pp. 9604–9611. DOI: 10.1109/IROS47612.2022.9981728.

[119] Kübra Karacan, Divij Grover, Hamid Sadeghian, Fan Wu, and Sami Haddadin. "Tactile Exploration Using Unified Force-Impedance Control". In: *IFAC-PapersOnLine* 56.2 (2023), pp. 5015–5020. DOI: 10.1016/j.ifacol.2023.10.1279.

[120] Ladislav Kavan, Steven Collins, Carol O'Sullivan, and Jiri Zara. *Dual Quaternions for Rigid Transformation Blending*. Technical Report. Dublin, Ireland: Trinity College, 2006, p. 11.

[121] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. "Geometric Skinning with Approximate Dual Quaternion Blending". In: *ACM Trans. Graph.* 27.4 (Nov. 2008), 105:1–105:23. DOI: 10.1145/1409625.1409627.

[122] Jake Ketchum, Ahalya Prabhakar, and Todd D. Murphey. *Active Exploration for Real-Time Haptic Training*. May 2024.

[123] O. Khatib. "A Unified Approach for Motion and Force Control of Robot Manipulators: The Operational Space Formulation". In: *IEEE Journal on Robotics and Automation* 3.1 (Feb. 1987), pp. 43–53. DOI: 10.1109/JRA.1987.1087068.

[124] Mahdi Khoramshahi, Gustav Henriks, Aileen Naef, Seyed Sina Mirrazavi Salehian, Joonyoung Kim, and Aude Billard. "Arm-Hand Motion-Force Coordination for Physical Interactions with Non-Flat Surfaces Using Dynamical Systems: Toward Compliant Robotic Massage". In: *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*. Paris, France: IEEE, May 2020, pp. 4724–4730. DOI: 10.1109/ICRA40945.2020.9196593.

[125] Jaeseok Kim, Anand Kumar Mishra, Raffaele Limosani, Marco Scafuro, Nino Cauli, Jose Santos-Victor, Barbara Mazzolai, and Filippo Cavallo. "Control Strategies for Cleaning Robots in Domestic Applications: A Comprehensive Review". In: *International Journal of Advanced Robotic Systems* 16.4 (July 2019), p. 1729881419857432. DOI: 10.1177/1729881419857432.

[126] Holger Klein, Noémie Jaquier, Andre Meixner, and Tamim Asfour. "On the Design of Region-Avoiding Metrics for Collision-Safe Motion Generation on Riemannian Manifolds". In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Detroit, MI, USA: IEEE, Oct. 2023, pp. 2346–2353. DOI: 10.1109/IROS55552.2023.10341266.

[127] N. Koenig and A. Howard. "Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3. Sendai, Japan: IEEE, 2004, pp. 2149–2154. DOI: 10.1109/IROS.2004.1389727.

[128] Dmitrii Kolpashchikov, Olga Gerget, and Viacheslav Danilov. "FABRIKx: Tackling the Inverse Kinematics Problem of Continuum Robots with Variable Curvature". In: *Robotics* 11.6 (Dec. 2022), p. 128. DOI: 10.3390/robotics11060128.

[129] Riddhiman Laha, Luis F.C. Figueredo, Juraj Vrabel, Abdalla Swikir, and Sami Haddadin. "Reactive Cooperative Manipulation Based on Set Primitives and Circular Fields". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. May 2021, pp. 6577–6584. DOI: 10.1109/ICRA48506.2021.9561985.

[130] J. Lasenby, H. Hadfield, and A. Lasenby. "Calculating the Rotor Between Conformal Objects". In: *Advances in Applied Clifford Algebras* 29.5 (Oct. 2019), p. 102. DOI: 10.1007/s00006-019-1014-8.

[131] Jean-Claude Latombe. *Robot Motion Planning*. Boston, MA: Springer US, 1991. DOI: 10.1007/978-1-4615-4022-9.

[132] 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". In: *Meccanica* 57.6 (June 2022), pp. 1413–1428. DOI: 10.1007/s11012-022-01512-w.

[133] Guillaume Leclercq, Philippe Lefèvre, and Gunnar Blohm. "3D Kinematics Using Dual Quaternions: Theory and Applications in Neuroscience". In: *Frontiers in Behavioral Neuroscience* 7 (2013). DOI: 10.3389/fnbeh.2013.00007.

[134] S. Lee and S. Kim. "A Self-Reconfigurable Dual-Arm System". In: *1991 IEEE International Conference on Robotics and Automation Proceedings*. Apr. 1991, 164–169 vol.1. DOI: 10.1109/ROBOT.1991.131573.

[135] Cameron Lerch, Dayi Dong, and Ian Abraham. "Safety-Critical Ergodic Exploration in Cluttered Environments via Control Barrier Functions". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. May 2023, pp. 10205–10211. DOI: 10.1109/ICRA48891.2023.10161032.

[136] Thomas Lew et al. "Robotic Table Wiping via Reinforcement Learning and Whole-body Trajectory Optimization". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. May 2023, pp. 7184–7190. DOI: 10.1109/ICRA48891.2023.10161283.

[137] Hongbo Li, David Hestenes, and Alyn Rockwood. "Generalized Homogeneous Coordinates for Computational Geometry". In: *Geometric Computing with Clifford Algebras*. Ed. by Gerald Sommer. 2001, pp. 27–59. DOI: 10.1007/978-3-662-04621-0_2.

[138] Yiming Li, Yan Zhang, Amirreza Razmjoo, and Sylvain Calinon. "Representing Robot Geometry as Distance Fields: Applications to Whole-body Manipulation". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. May 2024, pp. 15351–15357. DOI: 10.1109/ICRA57147.2024.10611674.

[139] Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. *GPU-Accelerated Robotic Simulation for Distributed Reinforcement Learning*. Oct. 2018.

[140] Toru Lin, Yu Zhang, Qiyang Li, Haozhi Qi, Brent Yi, Sergey Levine, and Jitendra Malik. *Learning Visuotactile Skills with Two Multifingered Hands*. May 2024.

[141] Yang Liu, Balakrishnan Prabhakaran, and Xiaohu Guo. "Point-Based Manifold Harmonics". In: *IEEE Transactions on Visualization and Computer Graphics* 18.10 (Oct. 2012), pp. 1693–1703. DOI: 10.1109/TVCG.2011.152.

[142] Ruhizan Liza Ahmad Shauri and Kenzo Nonami. "Assembly Manipulation of Small Objects by Dual-arm Manipulator". In: *Assembly Automation* 31.3 (Jan. 2011), pp. 263–274. DOI: 10.1108/01445151111150604.

[143] Wilder Bezerra Lopes and Cassio Guimaraes Lopes. "Geometric-Algebra Adaptive Filters". In: *IEEE Transactions on Signal Processing* 67.14 (July 2019), pp. 3649–3662. DOI: 10.1109/TSP.2019.2916028.

[144] Pertti Lounesto. *Clifford Algebras and Spinors*. 2. ed., repr., transferred to digital print. 2006. London Mathematical Society Lecture Note Series 286. Cambridge: Cambridge University Press, 2006.

[145] Tobias Löw, Philip Abbet, and Sylvain Calinon. "Gafro: Geometric Algebra for Robotics". In: *IEEE Robotics & Automation Magazine* (2024). DOI: 10.1109/MRA.2024.3433109.

[146] Tobias Löw, Cem Bilaloglu, and Sylvain Calinon. "Cooperative Geometric Primitives for Multi-Arm Manipulation Control". In: *under review* (2025).

[147] Tobias Löw and Sylvain Calinon. "Extending the Cooperative Dual-Task Space in Conformal Geometric Algebra". In: *IEEE International Conference on Robotics and Automation*. 2024.

[148] Tobias Löw and Sylvain Calinon. "Geometric Algebra for Optimal Control With Applications in Manipulation Tasks". In: *IEEE Transactions on Robotics* 39.5 (2023), pp. 3586–3600. DOI: 10.1109/TRO.2023.3277282.

[149] Tobias Löw and Sylvain Calinon. "Recursive Forward Dynamics of Serial Kinematic Chains Using Conformal Geometric Algebra". In: *Workshop on Algorithmic Foundations of Robotics (WAFR)*. 2024.

[150] Tobias Löw, Mariana de Paula Assis Fonseca, Vitalii Pruks, Graham Deacon, Jelizaveta Konstantinova, and Sylvain Calinon. "Dual-Arm Admittance Control Using Conformal Geometric Algebra". In: *in preparation* ().

[151] Ren C. Luo, Chin Po Tsai, and Kai Chun Hsieh. "Robot Assisted Tapping Control for Therapeutical Percussive Massage Applications". In: *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*. Singapore, Singapore: IEEE, May 2017, pp. 3606–3611. DOI: 10.1109/ICRA.2017.7989415.

[152] Rishabh Madan, Skyler Valdez, David Kim, Sujie Fang, Luoyan Zhong, Diego Virtue, and Tapomayukh Bhattacharjee. *RABBIT: A Robot-Assisted Bed Bathing System with Multimodal Perception and Integrated Compliance*. Jan. 2024.

[153] Lucas Manuelli, Wei Gao, Peter Florence, and Russ Tedrake. "KPAM: KeyPoint Affordances for Category-Level Robotic Manipulation". In: *Robotics Research*. Ed. by Tamim Asfour, Eiichi Yoshida, Jaeheung Park, Henrik Christensen, and Oussama Khatib. Cham: Springer International Publishing, 2022, pp. 132–157. DOI: 10.1007/978-3-030-95459-8_9.

[154] Bruno Maric, Alan Mutka, and Matko Orsag. "Collaborative Human-Robot Framework for Delicate Sanding of Complex Shape Surfaces". In: *IEEE Robotics and Automation Letters* 5.2 (Apr. 2020), pp. 2848–2855. DOI: 10.1109/LRA.2020.2969951.

[155] Filip Marić, Matthew Giamou, Adam W. Hall, Soroush Khoubyarian, Ivan Petrovic, and Jonathan Kelly. "Riemannian Optimization for Distance-Geometric Inverse Kinematics". In: *IEEE Transactions on Robotics* 38.3 (June 2022), pp. 1703–1722. DOI: 10.1109/TRO.2021.3123841.

[156] Filip Marić, Luka Petrović, Marko Guberina, Jonathan Kelly, and Ivan Petrović. "A Riemannian Metric for Geometry-Aware Singularity Avoidance by Articulated Robots". In: *Robotics and Autonomous Systems* 145 (Nov. 2021), p. 103865. DOI: 10.1016/j.robot.2021.103865.

[157] M. M. Marinho, L. F. C. Figueredo, and B. V. Adorno. "A Dual Quaternion Linear-Quadratic Optimal Controller for Trajectory Tracking". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2015, pp. 4047–4052. DOI: 10.1109/IROS.2015.7353948.

[158] Murilo Marques Marinho, Bruno Vilhena Adorno, Kanako Harada, and Mamoru Mitsuishi. "Dynamic Active Constraints for Surgical Robots Using Vector-Field Inequalities". In: *IEEE Transactions on Robotics* 35.5 (Oct. 2019), pp. 1166–1185. DOI: 10.1109/TRO.2019.2920078.

[159] Uriel Martinez-Hernandez, Tony J. Dodd, Mathew H. Evans, Tony J. Prescott, and Nathan F. Lepora. "Active Sensorimotor Control for Tactile Exploration". In: *Robotics and Autonomous Systems* 87 (Jan. 2017), pp. 15–27. DOI: 10.1016/j.robot.2016.09.014.

[160] Jonathan Masci, Davide Boscaini, Michael M. Bronstein, and Pierre Vandergheynst. "Geodesic Convolutional Neural Networks on Riemannian Manifolds". In: *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*. Santiago, Chile: IEEE, Dec. 2015, pp. 832–840. DOI: 10.1109/ICCVW.2015.112.

[161] George Mathew and Igor Mezic. "Spectral Multiscale Coverage: A Uniform Coverage Algorithm for Mobile Sensor Networks". In: *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) Held Jointly with 2009 28th Chinese Control Conference*. Dec. 2009, pp. 7872–7877. DOI: 10.1109/CDC.2009.5400401.

[162] George Mathew and Igor Mezić. "Metrics for Ergodicity and Design of Ergodic Dynamics for Multi-Agent Systems". In: *Physica D: Nonlinear Phenomena* 240.4-5 (Feb. 2011), pp. 432–442. DOI: 10.1016/j.physd.2010.10.010.

[163] K. Mertens, B. De Ketelaere, B. Kamers, F.R. Bamelis, B.J. Kemps, E.M. Verhoelst, J.G. De Baerdemaeker, and E. M Decuypere. "Dirt Detection on Brown Eggs by Means of Color Computer Vision". In: *Poultry Science* 84.10 (Oct. 2005), pp. 1653–1659. DOI: 10.1093/ps/84.10.1653.

[164] Andreas Müller. "Screw and Lie Group Theory in Multibody Dynamics: Recursive Algorithms and Equations of Motion of Tree-Topology Systems". In: *Multibody System Dynamics* 42.2 (Feb. 2018), pp. 219–248. DOI: 10.1007/s11044-017-9583-6.

[165] Robin R. Murphy, Satoshi Tadokoro, Daniele Nardi, Adam Jacoff, Paolo Fiorini, Howie Choset, and Aydan M. Erkmen. "Search and Rescue Robotics". In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Berlin, Heidelberg: Springer, 2008, pp. 1151–1173. DOI: 10.1007/978-3-540-30301-5_51.

[166] Amal Nanavati, Vinitha Ranganeni, and Maya Cakmak. "Physically Assistive Robots: A Systematic Review of Mobile and Manipulator Robots That Physically Assist People with Disabilities". In: *Annual Review of Control, Robotics, and Autonomous Systems* 7.1 (July 2024), pp. 123–147. DOI: 10.1146/annurev-control-062823-024352.

[167] Andrew Nealen and TU Darmstadt. "An As-Short-As-Possible Introduction to the Least Squares, Weighted Least Squares and Moving Least Squares Methods for Scattered Data Approximation and Interpolation". In: ().

[168] Rachele Nebbia Colomba, Riddhiman Laha, Luis F.C. Figueredo, and Sami Haddadin. "Adaptive Admittance Control for Cooperative Manipulation Using Dual Quaternion Representation and Logarithmic Mapping". In: *2022 IEEE 61st Conference on Decision and Control (CDC)*. Cancun, Mexico: IEEE, Dec. 2022, pp. 107–144. DOI: 10.1109/CDC51059.2022.9992402.

[169] Maximilian Neidhardt, Robin Mieling, Marcel Bengs, and Alexander Schlaefer. "Optical Force Estimation for Interactions between Tool and Soft Tissues". In: *Scientific Reports* 13.1 (Jan. 2023), p. 506. DOI: 10.1038/s41598-022-27036-7.

[170] Wu Xin Ng, Hau Kong Chan, Wee Kin Teo, and I-Ming Chen. "Programming a Robot for Conformance Grinding of Complex Shapes by Capturing the Tacit Knowledge of a Skilled Operator". In: *IEEE Transactions on Automation Science and Engineering* 14.2 (Apr. 2017), pp. 1020–1030. DOI: 10.1109/TASE.2015.2474708.

[171] Petter Ögren, Christian Smith, Yiannis Karayiannidis, and Danica Kragic. "A Multi Objective Control Approach to Online Dual Arm Manipulation1". In: *IFAC Proceedings Volumes*. 10th IFAC Symposium on Robot Control 45.22 (Jan. 2012), pp. 747–752. DOI: 10.3182/20120905-3-HR-2030.00032.

[172] M. Ollis and A. Stentz. "First Results in Vision-Based Crop Line Tracking". In: *Proceedings of IEEE International Conference on Robotics and Automation*. Vol. 1. Minneapolis, MN, USA: IEEE, 1996, pp. 951–956. DOI: 10.1109/ROBOT.1996.503895.

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

[174] Ingrid Fjordheim Onstein, Oleksandr Semeniuta, and Magnus Bjerkeng. "Deburring Using Robot Manipulators: A Review". In: *2020 3rd International Symposium on Small-scale Intelligent Manufacturing Systems (SIMS)*. Gjovik, Norway: IEEE, June 2020, pp. 1–7. DOI: 10.1109/SIMS49386.2020.9121490.

[175] *ORB-SLAM: A Versatile and Accurate Monocular SLAM System | IEEE Journals & Magazine | IEEE Xplore*.

[176] F.C. Park, J.E. Bobrow, and S.R. Ploen. "A Lie Group Formulation of Robot Dynamics". In: *The International Journal of Robotics Research* 14.6 (Dec. 1995), pp. 609–618. DOI: 10.1177/027836499501400606.

[177] H. Andy Park and C. S. George Lee. "Extended Cooperative Task Space for Manipulation Tasks of Humanoid Robots". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. Seattle, WA, USA: IEEE, May 2015, pp. 6088–6093. DOI: 10.1109/ICRA.2015.7140053.

[178] Vatsal V. Patel and Aaron M. Dollar. "Robot Hand Based on a Spherical Parallel Mechanism for Within-Hand Rotations about a Fixed Point". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Prague, Czech Republic: IEEE, Sept. 2021, pp. 709–716. DOI: 10.1109/IROS51168.2021.9636704.

[179] Christian Perwass. *Geometric Algebra with Applications in Engineering*. Geometry and Computing 4. Berlin: Springer, 2009.

[180] P. Pfändler, K. Bodie, G. Crotta, M. Pantic, R. Siegwart, and U. Angst. "Non-Destructive Corrosion Inspection of Reinforced Concrete Structures Using an Autonomous Flying Robot". In: *Automation in Construction* 158 (Feb. 2024), p. 105241. DOI: 10.1016/j.autcon.2023.105241.

[181] Martin Pfanne, Maxime Chalon, Freek Stulp, Helge Ritter, and Alin Albu-Schäffer. "Object-Level Impedance Control for Dexterous In-Hand Manipulation". In: *IEEE Robotics and Automation Letters* 5.2 (Apr. 2020), pp. 2987–2994. DOI: 10.1109/LRA.2020.2974702.

[182] S.R. Ploen and J.E. Bobrow. "An O(n) Geometric Algorithm for Manipulator Forward Dynamics". In: *International Conference on Advanced Robotics (ICAR)*. July 1997, pp. 563–568. DOI: 10.1109/ICAR.1997.620238.

[183] Namiko Saito, Danyang Wang, Tetsuya Ogata, Hiroki Mori, and Shigeki Sugano. "Wiping 3D-objects Using Deep Learning Model Based on Image/Force/Joint Information". In: *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*. Las Vegas, NV, USA: IEEE, Oct. 2020, pp. 10152–10157. DOI: 10.1109/IROS45743.2020.9341275.

[184] Hadi Salman, Elif Ayvali, Rangaprasad Arun Srivatsan, Yifei Ma, Nicolas Zevallos, Rashid Yasin, Long Wang, Nabil Simaan, and Howie Choset. "Trajectory-Optimized Sensing for Active Search of Tissue Abnormalities in Robotic Surgery". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. Brisbane, QLD: IEEE, May 2018, pp. 5356–5363. DOI: 10.1109/ICRA.2018.8460936.

[185] Emre Sariyildiz and Hakan Temeltas. "A Comparison Study of Three Screw Theory Based Kinematic Solution Methods for the Industrial Robot Manipulators". In: *2011 IEEE International Conference on Mechatronics and Automation*. Beijing, China: IEEE, Aug. 2011, pp. 52–57. DOI: 10.1109/ICMA.2011.5985630.

[186] Matteo Saveriano, Fares J. Abu-Dakka, and Ville Kyrki. "Learning Stable Robotic Skills on Riemannian Manifolds". In: *Robotics and Autonomous Systems* 169 (Nov. 2023), p. 104510. DOI: 10.1016/j.robot.2023.104510.

[187] Adam Seewald, Cameron J. Lerch, Marvin Chancán, Aaron M. Dollar, and Ian Abraham. *Energy-Aware Ergodic Search: Continuous Exploration for Multi-Agent Systems with Battery Constraints*. Oct. 2023.

[188] J. M. Selig and E. Bayro-Corrochano. "Rigid Body Dynamics Using Clifford Algebra". In: *Advances in Applied Clifford Algebras (ACAA)* 20.1 (Mar. 2010), pp. 141–154. DOI: 10.1007/s00006-008-0144-1.

[189] Florian Seybold and Uwe Wössner. "Gaalet - a C++ Expression Template Library for Implementing Geometric Algebra". In: *High-End Visualization Workshop*. 2010.

[190] Nicholas Sharp, Souhaib Attaiki, Keenan Crane, and Maks Ovsjanikov. *DiffusionNet: Discretization Agnostic Learning on Surfaces*. Mar. 2022. DOI: 10.1145/3507905.

[191] Nicholas Sharp and Keenan Crane. "A Laplacian for Nonmanifold Triangle Meshes". In: *Computer Graphics Forum* 39.5 (2020), pp. 69–80. DOI: 10.1111/cgf.14069.

[192] Suhan Shetty, João Silvério, and Sylvain Calinon. "Ergodic Exploration Using Tensor Train: Applications in Insertion Tasks". In: *IEEE Transactions on Robotics* 38.2 (Apr. 2022), pp. 906–921. DOI: 10.1109/TRO.2021.3087317.

[193] Vikas Shivashankar, Rajiv Jain, Ugur Kuter, and Dana Nau. "Real-Time Planning for Covering an Initially-Unknown Spatial Environment". In: ().

[194] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. Berlin: Springer, 2008.

[195] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics*. Ed. by Michael J. Grimble and Michael A. Johnson. Advanced Textbooks in Control and Signal Processing. London: Springer London, 2009. DOI: 10.1007/978-1-84628-642-1.

[196] Dominik Sieber, Frederik Deroo, and Sandra Hirche. "Formation-Based Approach for Multi-Robot Cooperative Manipulation Based on Optimal Control Design". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Tokyo: IEEE, Nov. 2013, pp. 5227–5233. DOI: 10.1109/IROS.2013.6697112.

[197] Dominik Sieber and Sandra Hirche. "Human-Guided Multirobot Cooperative Manipulation". In: *IEEE Transactions on Control Systems Technology* 27.4 (July 2019), pp. 1492–1509. DOI: 10.1109/TCST.2018.2813323.

[198] Joao Silverio, Leonel Rozo, Sylvain Calinon, and Darwin G. Caldwell. "Learning Bimanual End-Effector Poses from Demonstrations Using Task-Parameterized Dynamical Systems". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Hamburg: IEEE, Sept. 2015, pp. 464–470. DOI: 10.1109/IROS.2015.7353413.

[199] Christian Smith, Yiannis Karayiannidis, Lazaros Nalpantidis, Xavi Gratal, Peng Qi, Dimos V. Dimarogonas, and Danica Kragic. "Dual Arm Manipulation—A Survey". In: *Robotics and Autonomous Systems* 60.10 (Oct. 2012), pp. 1340–1353. DOI: 10.1016/j.robot.2012.07.005.

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

[201] Eduardo Vera Sousa and Leandro A. F. Fernandes. "TbGAL: A Tensor-Based Library for Geometric Algebra". In: *Advances in Applied Clifford Algebras* 30.2 (Apr. 2020), p. 27. DOI: 10.1007/s00006-020-1053-1.

[202] Max Spahn, Martijn Wisse, and Javier Alonso-Mora. "Dynamic Optimization Fabrics for Motion Generation". In: *IEEE Transactions on Robotics* (2023), pp. 1–16. DOI: 10.1109/TRO.2023.3255587.

[203] J. Spletzer, A.K. Das, R. Fierro, C.J. Taylor, V. Kumar, and J.P. Ostrowski. "Cooperative Localization and Control for Multi-Robot Manipulation". In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*. Vol. 2. Oct. 2001, 631–636 vol.2. DOI: 10.1109/IROS.2001.976240.

[204] Muchen Sun, Ayush Gaggar, Peter Trautman, and Todd Murphey. *Fast Ergodic Search with Kernel Functions*. Mar. 2024.

[205] Yinshuai Sun, Zhongliang Jing, Peng Dong, Jianzhe Huang, Wujun Chen, and Henry Leung. "A Switchable Unmanned Aerial Manipulator System for Window-Cleaning Robot Installation". In: *IEEE Robotics and Automation Letters* 6.2 (Apr. 2021), pp. 3483–3490. DOI: 10.1109/LRA.2021.3062795.

[206] Chee Sheng Tan, Rosmiwati Mohd-Mokhtar, and Mohd Rizal Arshad. "A Comprehensive Review of Coverage Path Planning in Robotics Using Classical and Heuristic Algorithms". In: *IEEE access : practical innovations, open solutions* 9 (2021), pp. 119310–119342. DOI: 10.1109/ACCESS.2021.3108177.

[207] Yuval Tassa, Tom Erez, and Emanuel Todorov. "Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2012, pp. 4906–4913. DOI: 10.1109/IROS.2012.6386025.

[208] R.H. Taylor and D. Stoianovici. "Medical Robotics in Computer-Integrated Surgery". In: *IEEE Transactions on Robotics and Automation* 19.5 (Oct. 2003), pp. 765–781. DOI: 10.1109/TRA.2003.817058.

[209] Boyang Ti, Amirreza Razmjoo, Yongsheng Gao, Jie Zhao, and Sylvain Calinon. "A Geometric Optimal Control Approach for Imitation and Generalization of Manipulation Skills". In: *Robotics and Autonomous Systems* 164 (June 2023), p. 104413. DOI: 10.1016/j.robot.2023.104413.

[210] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A Physics Engine for Model-Based Control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[211] Julen Urain, Davide Tateo, and Jan Peters. *Learning Stable Vector Fields on Lie Groups*. Oct. 2022.

[212] *V-REP: A Versatile and Scalable Robot Simulation Framework | IEEE Conference Publication | IEEE Xplore*.

[213] Karl Van Wyk et al. "Geometric Fabrics: Generalizing Classical Mechanics to Capture the Physics of Behavior". In: *IEEE Robotics and Automation Letters* 7.2 (Apr. 2022), pp. 3202–3209. DOI: 10.1109/LRA.2022.3143311.

[214] Prabakaran Veerajagadheswar, Shi Yuyao, Prathap Kandasamy, Mohan R. Elara, and Abdullah A. Hayat. "S-Sacrr: A Staircase and Slope Accessing Reconfigurable Cleaning Robot and Its Validation". In: *IEEE Robotics and Automation Letters* 7.2 (Apr. 2022), pp. 4558–4565. DOI: 10.1109/LRA.2022.3151572.

[215] Emanuele Vignali, Emanuele Gasparotti, Katia Capellini, Benigno Marco Fanni, Luigi Landini, Vincenzo Positano, and Simona Celi. "Modeling Biomechanical Interaction between Soft Tissue and Soft Robotic Instruments: Importance of Constitutive Anisotropic Hyperelastic Formulations". In: *The International Journal of Robotics Research* 40.1 (Jan. 2021), pp. 224–235. DOI: 10.1177/0278364920927476.

[216] Jonathan Vorndamme, Joao Carvalho, Riddhiman Laha, Dorothea Koert, Luis Figueredo, Jan Peters, and Sami Haddadin. "Integrated Bi-Manual Motion Generation and Control Shaped for Probabilistic Movement Primitives". In: *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*. Ginowan, Japan: IEEE, Nov. 2022, pp. 202–209. DOI: 10.1109/Humanoids53995.2022.10000149.

[217] Youcan Yan and Jia Pan. "Fast Localization and Segmentation of Tissue Abnormalities by Autonomous Robotic Palpation". In: *IEEE Robotics and Automation Letters* 6.2 (Apr. 2021), pp. 1707–1714. DOI: 10.1109/LRA.2021.3058870.

[218] Jingyun Yang, Congyue Deng, Jimmy Wu, Rika Antonova, Leonidas Guibas, and Jeannette Bohg. "EquivAct: SIM(3)-Equivariant Visuomotor Policies beyond Rigid Object Manipulation". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)* (May 2024), pp. 9249–9255. DOI: 10.1109/ICRA57147.2024.10611491.

[219] Kunpeng Yao and Aude Billard. "Exploiting Kinematic Redundancy for Robotic Grasping of Multiple Objects". In: 39.3 (2023), pp. 1982–2002. DOI: 10.1109/TRO.2023.3253249.

[220] Julio Zamora-Esquivel. "G 6,3 Geometric Algebra; Description and Implementation". In: *Advances in Applied Clifford Algebras* 24.2 (June 2014), pp. 493–514. DOI: 10.1007/s00006-014-0442-8.

[221] Julio Zamora-Esquivel and Eduardo Bayro-Corrochano. "Robot Object Manipulation Using Stereoscopic Vision and Conformal Geometric Algebra". In: *Applied Bionics and Biomechanics* 8.3-4 (2011), pp. 411–428. DOI: 10.1155/2011/728132.

[222] Isiah Zaplana, Hugo Hadfield, and Joan Lasenby. "Closed-Form Solutions for the Inverse Kinematics of Serial Robots Using Conformal Geometric Algebra". In: *Mechanism and Machine Theory* 173 (2022). DOI: 10.1016/j.mechmachtheory.2022.104835.

[223] Hai Zhu, Nicholas R. J. Lawrance, Roland Siegwart, and Javier Alonso-Mora. *Online Informative Path Planning for Active Information Gathering of a 3D Surface*. Mar. 2021. DOI: 10.48550/arXiv.2103.09556.

[224] Jihong Zhu, Benjamin Navarro, Philippe Fraisse, Andre Crosnier, and Andrea Cherubini. "Dual-Arm Robotic Manipulation of Flexible Cables". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid: IEEE, Oct. 2018, pp. 479–484. DOI: 10.1109/IROS.2018.8593780.

[225] Daniel Zwillinger. *Standard Mathematical Tables and Formulae*. 31st ed. CRC Press, 2003.

A. Derivation of the Jacobians

A.1. Derivation of the Jacobian of the Motor Logarithmic Map

From Equation (2.41) we know that $B = \log(M)$. We define the parameters of the motor and bivector as follows

$$M = m_1 + m_2\boldsymbol{e}_{23} + m_3\boldsymbol{e}_{13} + m_4\boldsymbol{e}_{12} + m_5\boldsymbol{e}_{1\infty} + m_6\boldsymbol{e}_{2\infty} + m_7\boldsymbol{e}_{3\infty} + m_8\boldsymbol{e}_{123\infty}, \quad \text{(A.1)}$$

and

$$B = b_1\boldsymbol{e}_{23} + b_2\boldsymbol{e}_{13} + b_3\boldsymbol{e}_{12} + b_4\boldsymbol{e}_{1\infty} + b_5\boldsymbol{e}_{2\infty} + b_6\boldsymbol{e}_{3\infty}. \quad \text{(A.2)}$$

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

$$\begin{aligned} R &= -\boldsymbol{e}_0 \cdot M\boldsymbol{e}_\infty, & \text{(A.3)}\\ T &= M\widetilde{R}. & \text{(A.4)} \end{aligned}$$

Using the Equations (2.30) and (2.32) it becomes straightforward to derive the bivector components b_i in function of the motor components m_i

$$\begin{aligned} b_1 &= -m_2\frac{2\cos^{-1}(m_1)}{\sin(\cos^{-1}(m_1))}, & \text{(A.5)}\\ b_2 &= -m_3\frac{2\cos^{-1}(m_1)}{\sin(\cos^{-1}(m_1))}, & \text{(A.6)}\\ b_3 &= -m_4\frac{2\cos^{-1}(m_1)}{\sin(\cos^{-1}(m_1))}, & \text{(A.7)}\\ b_4 &= -2(m_1m_5 + m_4m_6 + m_3m_7 + m_2m_8), & \text{(A.8)}\\ b_5 &= -2(-m_4m_5 + m_1m_6 + m_2m_7 - m_3m_8), & \text{(A.9)}\\ b_6 &= -2(-m_3m_5 - m_2m_6 + m_1m_7 + m_4m_8). & \text{(A.10)} \end{aligned}$$

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.

$$\boldsymbol{J}_{\mathcal{M}\to\mathbb{B}}(M) = \begin{bmatrix} \frac{\partial b_1}{\partial m_1} & \cdots & \frac{\partial b_1}{\partial m_8} \\ \vdots & \ddots & \vdots \\ \frac{\partial b_6}{\partial m_1} & \cdots & \frac{\partial b_6}{\partial m_8} \end{bmatrix}. \tag{A.11}$$

Using the Equations (A.5) the non-trivial partial derivatives can be found to be

$$\frac{\partial b_1}{\partial m_1} = -2m_2 \left(\frac{1}{m_1^2 - 1} + m_1 \cos^{-1}(m_1) \right), \tag{A.12}$$

$$\frac{\partial b_2}{\partial m_1} = -2m_3 \left(\frac{1}{m_1^2 - 1} + m_1 \cos^{-1}(m_1) \right), \tag{A.13}$$

$$\frac{\partial b_3}{\partial m_1} = -2m_4 \left(\frac{1}{m_1^2 - 1} + m_1 \cos^{-1}(m_1) \right), \tag{A.14}$$

$$\frac{\partial b_1}{\partial m_2} = \frac{\partial b_2}{\partial m_3} = \frac{\partial b_3}{\partial m_4} = \frac{-2\cos^{-1}(m_1)}{\sin\left(\cos^{-1}(m_1)\right)}, \tag{A.15}$$

$$\frac{\partial b_4}{\partial m_1} = -\frac{\partial b_5}{\partial m_4} = -\frac{\partial b_6}{\partial m_3} = -2m_5, \tag{A.16}$$

$$\frac{\partial b_4}{\partial m_2} = -\frac{\partial b_5}{\partial m_3} = \frac{\partial b_6}{\partial m_4} = -2m_8, \tag{A.17}$$

$$\frac{\partial b_4}{\partial m_3} = \frac{\partial b_5}{\partial m_2} = \frac{\partial b_6}{\partial m_1} = -2m_7, \tag{A.18}$$

$$\frac{\partial b_4}{\partial m_4} = \frac{\partial b_5}{\partial m_1} = -\frac{\partial b_6}{\partial m_2} = -2m_6, \tag{A.19}$$

$$\frac{\partial b_4}{\partial m_5} = \frac{\partial b_5}{\partial m_6} = \frac{\partial b_6}{\partial m_7} = -2m_1, \tag{A.20}$$

$$\frac{\partial b_4}{\partial m_6} = -\frac{\partial b_5}{\partial m_5} = \frac{\partial b_6}{\partial m_8} = -2m_4, \tag{A.21}$$

$$\frac{\partial b_4}{\partial m_7} = -\frac{\partial b_5}{\partial m_8} = -\frac{\partial b_6}{\partial m_5} = -2m_3, \tag{A.22}$$

$$\frac{\partial b_4}{\partial m_8} = \frac{\partial b_5}{\partial m_7} = -\frac{\partial b_6}{\partial m_6} = -2m_2, \tag{A.23}$$

which concludes the derivation.

A.2. Derivation of the Jacobian of the exponential map

The exponential map is a mapping from bivectors to motors, i.e. $B = \log(M)$. The parameters of the motor and bivector are as follows

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

and

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

Since a motor is a product of a translator T and a rotor R, we can find the exponential map

$$M = TR,$$

$$T = \left(1 - \frac{1}{2}(b_4\boldsymbol{e}_{1\infty} + b_5\boldsymbol{e}_{2\infty} + b_6\boldsymbol{e}_{3\infty})\right),$$

$$R = \frac{1}{\theta}\left(\cos(\frac{1}{2}\theta) - \sin(\frac{1}{2}\theta)\,(b_1\boldsymbol{e}_{23} + b_2\boldsymbol{e}_{13} + b_3\boldsymbol{e}_{12})\right),$$

where

$$\theta = \sqrt{b_1^2 + b_2^2 + b_3^2}.$$

In terms of parameters the m_i and b_i this becomes

$$\begin{aligned}
m_1 &= \frac{1}{\sqrt{b_1^2 + b_2^2 + b_3^2}} \cos\left(\frac{1}{2}\sqrt{b_1^2 + b_2^2 + b_3^2}\right), \\
m_2 &= \frac{-b_1}{b_1^2 + b_2^2 + b_3^2} \sin\left(\frac{1}{2}\sqrt{b_1^2 + b_2^2 + b_3^2}\right), \\
m_3 &= \frac{-b_2}{b_1^2 + b_2^2 + b_3^2} \sin\left(\frac{1}{2}\sqrt{b_1^2 + b_2^2 + b_3^2}\right), \\
m_4 &= \frac{-b_3}{b_1^2 + b_2^2 + b_3^2} \sin\left(\frac{1}{2}\sqrt{b_1^2 + b_2^2 + b_3^2}\right), \\
m_5 &= \frac{1}{2}\left(m_1 b_4 + m_3 b_6 + m_4 b_5\right), \\
m_6 &= \frac{1}{2}\left(m_1 b_5 + m_2 b_6 - m_4 b_4\right), \\
m_7 &= \frac{1}{2}\left(m_1 b_5 - m_2 b_5 - m_3 b_4\right), \\
m_8 &= \frac{1}{2}\left(m_2 b_4 - m_3 b_5 + m_4 b_6\right).
\end{aligned}$$

The non-trivial partial derivatives can then be found as

$$
\begin{aligned}
\frac{\partial m_1}{\partial b_1} &= -\frac{1}{2}\frac{b_1}{\theta}\sin\left(\frac{1}{2}\theta\right), \\
\frac{\partial m_1}{\partial b_2} &= -\frac{1}{2}\frac{b_2}{\theta}\sin\left(\frac{1}{2}\theta\right), \\
\frac{\partial m_1}{\partial b_3} &= -\frac{1}{2}\frac{b_3}{\theta}\sin\left(\frac{1}{2}\theta\right), \\
\frac{\partial m_2}{\partial b_1} &= \sin\left(\frac{1}{2}\theta\right)\left(\frac{b_1^2}{\theta^3}-\frac{1}{\theta}\right)-\frac{1}{2}\frac{b_1^2}{\theta^2}\cos\left(\frac{1}{2}\theta\right), \\
\frac{\partial m_2}{\partial b_2} &= b_1\left(\frac{b_2}{\theta^3}\sin\left(\frac{1}{2}\theta\right)-\frac{1}{2}\frac{b_2}{\theta^2}\cos\left(\frac{1}{2}\theta\right)\right), \\
\frac{\partial m_2}{\partial b_3} &= b_1\left(\frac{b_3}{\theta^3}\sin\left(\frac{1}{2}\theta\right)-\frac{1}{2}\frac{b_3}{\theta^2}\cos\left(\frac{1}{2}\theta\right)\right), \\
\frac{\partial m_3}{\partial b_1} &= b_2\left(\frac{b_1}{\theta^3}\sin\left(\frac{1}{2}\theta\right)-\frac{1}{2}\frac{b_1}{\theta^2}\cos\left(\frac{1}{2}\theta\right)\right), \\
\frac{\partial m_3}{\partial b_2} &= \sin\left(\frac{1}{2}\theta\right)\left(\frac{b_2^2}{\theta^3}-\frac{1}{\theta}\right)-\frac{1}{2}\frac{b_2^2}{\theta^2}\cos\left(\frac{1}{2}\theta\right), \\
\frac{\partial m_3}{\partial b_3} &= b_2\left(\frac{b_3}{\theta^3}\sin\left(\frac{1}{2}\theta\right)-\frac{1}{2}\frac{b_3}{\theta^2}\cos\left(\frac{1}{2}\theta\right)\right), \\
\frac{\partial m_4}{\partial b_1} &= b_3\left(\frac{b_1}{\theta^3}\sin\left(\frac{1}{2}\theta\right)-\frac{1}{2}\frac{b_1}{\theta^2}\cos\left(\frac{1}{2}\theta\right)\right), \\
\frac{\partial m_4}{\partial b_2} &= b_3\left(\frac{b_2}{\theta^3}\sin\left(\frac{1}{2}\theta\right)-\frac{1}{2}\frac{b_2}{\theta^2}\cos\left(\frac{1}{2}\theta\right)\right), \\
\frac{\partial m_4}{\partial b_3} &= \sin\left(\frac{1}{2}\theta\right)\left(\frac{b_3^2}{\theta^3}-\frac{1}{\theta}\right)-\frac{1}{2}\frac{b_3^2}{\theta^2}\cos\left(\frac{1}{2}\theta\right), \\
\frac{\partial m_5}{\partial b_1} &= -\frac{1}{2}\left(b_4\frac{\partial m_1}{\partial b_1}-b_6\frac{\partial m_3}{\partial b_1}-b_5\frac{\partial m_4}{\partial b_1}\right), \\
\frac{\partial m_5}{\partial b_2} &= -\frac{1}{2}\left(b_4\frac{\partial m_1}{\partial b_2}-b_6\frac{\partial m_3}{\partial b_2}-b_5\frac{\partial m_4}{\partial b_2}\right), \\
\frac{\partial m_5}{\partial b_3} &= -\frac{1}{2}\left(b_4\frac{\partial m_1}{\partial b_3}-b_6\frac{\partial m_3}{\partial b_3}-b_5\frac{\partial m_4}{\partial b_3}\right), \\
\frac{\partial m_6}{\partial b_1} &= -\frac{1}{2}\left(b_5\frac{\partial m_1}{\partial b_1}-b_6\frac{\partial m_2}{\partial b_1}+b_4\frac{\partial m_4}{\partial b_1}\right), \\
\frac{\partial m_6}{\partial b_2} &= -\frac{1}{2}\left(b_5\frac{\partial m_1}{\partial b_2}-b_6\frac{\partial m_2}{\partial b_2}+b_4\frac{\partial m_4}{\partial b_2}\right), \\
\frac{\partial m_6}{\partial b_3} &= -\frac{1}{2}\left(b_5\frac{\partial m_1}{\partial b_3}-b_6\frac{\partial m_2}{\partial b_3}+b_4\frac{\partial m_4}{\partial b_3}\right),
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial m_7}{\partial b_1} &= -\frac{1}{2}\left(b_6\frac{\partial m_1}{\partial b_1} + b_5\frac{\partial m_2}{\partial b_1} + b_4\frac{\partial m_3}{\partial b_1}\right),\\
\frac{\partial m_7}{\partial b_2} &= -\frac{1}{2}\left(b_6\frac{\partial m_1}{\partial b_2} + b_5\frac{\partial m_2}{\partial b_2} + b_4\frac{\partial m_3}{\partial b_2}\right),\\
\frac{\partial m_7}{\partial b_3} &= -\frac{1}{2}\left(b_6\frac{\partial m_1}{\partial b_3} + b_5\frac{\partial m_2}{\partial b_3} + b_4\frac{\partial m_3}{\partial b_3}\right),\\
\frac{\partial m_8}{\partial b_1} &= -\frac{1}{2}\left(b_4\frac{\partial m_2}{\partial b_1} - b_5\frac{\partial m_3}{\partial b_1} + b_6\frac{\partial m_4}{\partial b_1}\right),\\
\frac{\partial m_8}{\partial b_2} &= -\frac{1}{2}\left(b_4\frac{\partial m_2}{\partial b_2} - b_5\frac{\partial m_3}{\partial b_2} + b_6\frac{\partial m_4}{\partial b_2}\right),\\
\frac{\partial m_8}{\partial b_3} &= -\frac{1}{2}\left(b_4\frac{\partial m_2}{\partial b_3} - b_5\frac{\partial m_3}{\partial b_3} + b_6\frac{\partial m_4}{\partial b_3}\right),\\
\frac{\partial m_5}{\partial b_4} &= -\frac{1}{2}m_1,\\
\frac{\partial m_5}{\partial b_5} &= \frac{1}{2}m_4,\\
\frac{\partial m_5}{\partial b_6} &= \frac{1}{2}m_3,\\
\frac{\partial m_6}{\partial b_4} &= -\frac{1}{2}m_4,\\
\frac{\partial m_6}{\partial b_5} &= -\frac{1}{2}m_1,\\
\frac{\partial m_6}{\partial b_6} &= \frac{1}{2}m_2,\\
\frac{\partial m_7}{\partial b_4} &= -\frac{1}{2}m_3,\\
\frac{\partial m_7}{\partial b_5} &= -\frac{1}{2}m_2,\\
\frac{\partial m_7}{\partial b_6} &= -\frac{1}{2}m_1,\\
\frac{\partial m_8}{\partial b_4} &= -\frac{1}{2}m_2,\\
\frac{\partial m_8}{\partial b_5} &= \frac{1}{2}m_3,\\
\frac{\partial m_8}{\partial b_6} &= -\frac{1}{2}m_4.
\end{aligned}
$$

A.3. Normalizing a Multivector

Given a multivector X, where $X\widetilde{X} \in \mathbb{R}$, it can be normalized as

$$\bar{X} = X \left| X\widetilde{X} \right|^{-\frac{1}{2}}. \tag{A.24}$$

For a multivector valued function $X = F(\boldsymbol{x})$, we can then find its derivative as

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{x}} \bar{F}(\boldsymbol{x}) = & \left| X\widetilde{X} \right|^{-\frac{1}{2}} \boldsymbol{\mathcal{J}}_F \\ & - \frac{1}{2} \left| X\widetilde{X} \right|^{-\frac{3}{2}} \left(X(\boldsymbol{\mathcal{J}}_F \widetilde{X} + X \widetilde{\boldsymbol{\mathcal{J}}}_F) \right). \end{aligned} \tag{A.25}$$

A.4. Inverting a Multivector

Given a multivector X, where $X\widetilde{X} \in \mathbb{R}$, its inverse can be found as

$$X^{-1} = \widetilde{X} \left(X\widetilde{X} \right)^{-1}. \tag{A.26}$$

For a multivector valued function $X = F(\boldsymbol{x})$, we can then find its derivative as

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{x}} F^{-1}(\boldsymbol{x}) = & \left(X\widetilde{X} \right)^{-1} \widetilde{\boldsymbol{\mathcal{J}}}_F \\ & - 2 \left(X\widetilde{X} \right)^{-2} \widetilde{X} \left\langle X \widetilde{\boldsymbol{\mathcal{J}}}_F \right\rangle_0. \end{aligned} \tag{A.27}$$

A.5. Analytic Circle Similarity Jacobian

Here, we derive the analytic similarity Jacobian $\boldsymbol{\mathcal{J}}^A_{S,c}(\boldsymbol{q})$, given the cooperative geometric primitive $X_c(\boldsymbol{q})$ and the corresponding Jacobian $\boldsymbol{\mathcal{J}}^A_c(\boldsymbol{q})$ from Equation (6.3). As shown in Equation (6.21), we can decompose the analytic similarity into the Jacobians for the translator $\boldsymbol{\mathcal{J}}^A_T(\boldsymbol{q})$, rotor $\boldsymbol{\mathcal{J}}^A_R(\boldsymbol{q})$, and dilator $\boldsymbol{\mathcal{J}}^A_D(\boldsymbol{q})$.

The analytic dilator Jacobian can be found as

$$\boldsymbol{\mathcal{J}}^A_D(\boldsymbol{q}) = -\frac{1}{4} (P_\infty \widetilde{P}_\infty)^{-1} \left\langle \boldsymbol{\mathcal{J}}_\infty \widetilde{P}_\infty + P_\infty \widetilde{\boldsymbol{\mathcal{J}}}_\infty \right\rangle_0 D, \tag{A.28}$$

where P_∞ is the projection of infinity from Equation (6.13)

$$P_\infty = (X \cdot \boldsymbol{e}_\infty) X^{-1}, \tag{A.29}$$

and $\boldsymbol{\mathcal{J}}_\infty$ its Jacobian

$$\boldsymbol{\mathcal{J}}_\infty = (\boldsymbol{e}_\infty \cdot \boldsymbol{\mathcal{J}}_c^A)C^{-1} + (\boldsymbol{e}_\infty \cdot C)\hat{\boldsymbol{\mathcal{J}}}_c^A. \tag{A.30}$$

The analytic rotor Jacobian can be found as

$$\begin{aligned}\boldsymbol{\mathcal{J}}_R^A(\boldsymbol{q}) = &\left|R\widetilde{R}\right|^{-\frac{1}{2}} \boldsymbol{\mathcal{J}}_R \\ &- \frac{1}{2}\left|R\widetilde{R}\right|^{-\frac{3}{2}} R(R\widetilde{\boldsymbol{\mathcal{J}}}_R + \boldsymbol{\mathcal{J}}_R\widetilde{R}),\end{aligned} \tag{A.31}$$

where R is the rotor $R(\boldsymbol{q})$ before normalization according to Equation (6.15) and $\boldsymbol{\mathcal{J}}_R$ the corresponding Jacobian

$$\boldsymbol{\mathcal{J}}_R = \boldsymbol{e}_3 \cdot \bar{\boldsymbol{\mathcal{J}}}_N - \boldsymbol{e}_3 \wedge \bar{\boldsymbol{\mathcal{J}}}_N, \tag{A.32}$$

where $\bar{\boldsymbol{\mathcal{J}}}_N$ is the normalized Jacobian of the plane normal found in Equation (6.14)

$$\begin{aligned}\bar{\boldsymbol{\mathcal{J}}}_N = &\left|N\widetilde{N}\right|^{-\frac{1}{2}} \boldsymbol{\mathcal{J}}_N \\ &- \frac{1}{2}\left|N\widetilde{N}\right|^{-\frac{3}{2}} N(N\widetilde{\boldsymbol{\mathcal{J}}}_N + \boldsymbol{\mathcal{J}}_N\widetilde{N}),\end{aligned} \tag{A.33}$$

where

$$\boldsymbol{\mathcal{J}}_N = \boldsymbol{\mathcal{J}}_E^* - \frac{1}{2}(\boldsymbol{e}_0 \cdot \boldsymbol{\mathcal{J}}_E^*)\boldsymbol{e}_\infty, \tag{A.34}$$

and

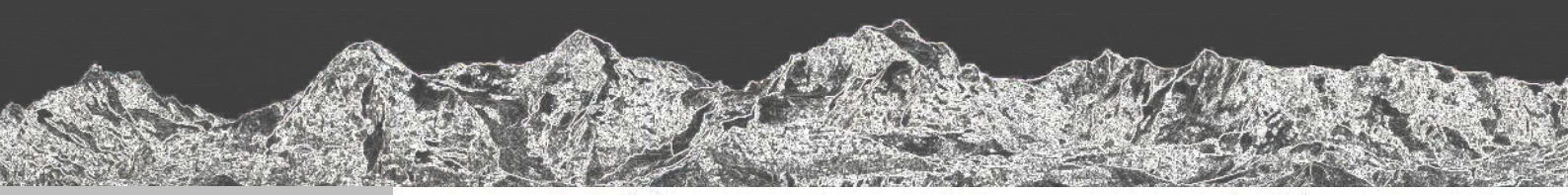
$$\boldsymbol{\mathcal{J}}_E = \boldsymbol{\mathcal{J}}_c^A(\boldsymbol{q}) \wedge \boldsymbol{e}_\infty. \tag{A.35}$$

The analytic translator Jacobian can be found as

$$\begin{aligned}\boldsymbol{\mathcal{J}}_T^A = &-\frac{1}{2}\langle P_c\rangle_0^{-1} \boldsymbol{\mathcal{J}}_P \\ &- \langle P_c\rangle_0^{-1} P_c(-\boldsymbol{e}_\infty \cdot \langle \boldsymbol{\mathcal{J}}_P\rangle_0) \wedge \boldsymbol{e}_\infty,\end{aligned} \tag{A.36}$$

where P_c is the center point and $\boldsymbol{\mathcal{J}}_P$ its Jacobian

$$\boldsymbol{\mathcal{J}}_P = \boldsymbol{\mathcal{J}}_c^A(\boldsymbol{q})\boldsymbol{e}_\infty X_c(\boldsymbol{q}) + X_c(\boldsymbol{q})\boldsymbol{e}_\infty \boldsymbol{\mathcal{J}}_c^A(\boldsymbol{q}). \tag{A.37}$$



Tobias Löw

Contact

tobi.loew@protonmail.ch
tobias-löw-982aa3179

Nationality

Switzerland
Germany

Languages

(Swiss) German
English
French

Website

tobiloew.ch

Coding

gitlab.com/tloew
gitlab.com/gafro
github.com/loewt

Education

PhD in Electrical Engineering 01/2021-06/2025
EPFL, Lausanne VD, Switzerland
Thesis Title: Robot Manipulation with Geometric Algebra: A Unified Geometric Framework for Control and Optimization

MSc in Mechanical Engineering 03/2018-09/2020
ETH Zürich, Zürich ZH, Switzerland
Thesis Title: Motion Planning with Motion Primitives

BSc in Mechanical Engineering 09/2013-07/2017
ETH Zürich, Zürich ZH, Switzerland
Thesis Title: Motion Control of a Fluidic Muscle – An Alternative Actuation for ATHLAS

Experience

Research Assistant 01/2021-06/2025
Idiap Research Institute, Martigny VS, Switzerland

Post-Graduate Student 10/2018-06/2020
CSIRO, Pullenvale, Brisbane, QLD, Australia

Laboratory Assistant 07/2013-12/2017
u-Blox AG, Thalwil ZH, Switzerland

Teaching Experience

Teaching Assistance

EE613 Machine Learning for Engineers 09/2023-01/2024
EPFL, Lausanne VD, Switzerland

MasterAI A04 - Robotics 09/2023-01/2024
UniDistance, Brig VS, Switzerland

MICRO-452 Basics of mobile robotics 09/2022-01/2023
EPFL, Lausanne VD, Switzerland

Thermodynamics I 09/2016-01/2017
ETH Zürich, Zürich ZH, Switzerland

Committees

Organizing Committee — Workshop on Geometric Representations: The Roles of Modern Screw Theory, Lie algebra, and Geometric Algebra in Robotics
`https://www.mirmi.tum.de/en/mirmi/veranstaltungen/icra-2023-workshop/program/`
IEEE Intl Conf. on Robotics and Automation (ICRA'2023), London, UK.

Awards

IEEE RAS, TC on Model-based Optimization for Robotics, Best Paper Award
Shetty et al.,
"Tensor Train for Global Optimization Problems in Robotics,"
The International Journal of Robotics Research, 2023

Idiap Best Paper Award
Löw and Calinon,
"Geometric Algebra for Optimal Control With Applications in Manipulation Tasks,"
IEEE Transactions on Robotics, 2023

Publications

[1] S. Shetty, T. Lembono, T. Löw, and S. Calinon, "Tensor train for global optimization problems in robotics," *The International Journal of Robotics Research*, p. 02 783 649 231 217 527, Nov. 30, 2023. DOI: 10.1177/02783649231217527.

[2] T. Löw and S. Calinon, "Geometric Algebra for Optimal Control With Applications in Manipulation Tasks," *IEEE Transactions on Robotics*, vol. 39, no. 5, pp. 3586–3600, 2023. DOI: 10.1109/TRO.2023.3277282.

[3] C. Bilaloglu, T. Löw, and S. Calinon, "Tactile Ergodic Coverage on Curved Surfaces," *IEEE Transaction on Robotics (T-RO)*, 2025.

[4] C. Bilaloglu, T. Löw, and S. Calinon, "Whole-Body Ergodic Exploration with a Manipulator Using Diffusion," *IEEE Robot. Autom. Lett.*, pp. 1–7, 2023. DOI: 10.1109/LRA.2023.3329755.

[5] R. Bowyer, T. Lowe, P. Borges, T. Bandyopadhyay, T. Löw, and D. Haddon, "PaintPath: Defining Path Directionality in Maps for Autonomous Ground Vehicles," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, NV, USA: IEEE, Oct. 24, 2020, pp. 2362–2369. DOI: 10.1109/IROS45743.2020.9341676.

[6] X. Chi, T. Löw, Y. Li, Z. Liu, and S. Calinon. "Geometric Projectors: Geometric Constraints based Optimization for Robot Behaviors," Accessed: Feb. 8, 2024. [Online]. Available: http://arxiv.org/abs/2309.08802, pre-published.

[7] H. Girgin, T. Löw, T. Xue, and S. Calinon. "Projection-based first-order constrained optimization solver for robotics," Accessed: Feb. 8, 2024. [Online]. Available: http://arxiv.org/abs/2306.17611, pre-published.

[8] T. Löw, J. Maceiras, and S. Calinon, "drozBot: Using Ergodic Control to Draw Portraits," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 728–11 734, 4 Oct. 2022. DOI: 10.1109/LRA.2022.3186735.

[9] T. Löw and S. Calinon, "Extending the Cooperative Dual-Task Space in Conformal Geometric Algebra," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[10] T. Löw, P. Abbet, and S. Calinon, "Gafro: Geometric Algebra for Robotics," *IEEE Robotics & Automation Magazine*, 2024. DOI: 10.1109/MRA.2024.3433109.

[11] T. Löw, T. Bandyopadhyay, and P. V. K. Borges, "Identification of Effective Motion Primitives for Ground Vehicles," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020, pp. 2027–2034. DOI: 10.1109/IROS45743.2020.9341708.

[12] T. Löw, T. Bandyopadhyay, J. Williams, and P. Borges, "PROMPT: Probabilistic Motion Primitives based Trajectory Planning," in *Robotics: Science and Systems XVII*, Robotics: Science and Systems Foundation, Jul. 12, 2021. DOI: 10.15607/RSS.2021.XVII.058.

[13] T. Löw and S. Calinon, "Recursive Forward Dynamics of Serial Kinematic Chains using Conformal Geometric Algebra," in *Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2024.

[14] B. Stolz et al., "An Adaptive Landing Gear for Extending the Operational Range of Helicopters," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid: IEEE, Oct. 2018, pp. 1757–1763. DOI: 10.1109/IROS.2018.8594062.

[15] G. G. Waibel, T. Löw, M. Nass, D. Howard, T. Bandyopadhyay, and P. V. K. Borges, "How Rough Is the Path? Terrain Traversability Estimation for Local and Global Path Planning," *IEEE Trans. Intell. Transport. Syst.*, vol. 23, no. 9, pp. 16 462–16 473, Sep. 2022. DOI: 10.1109/TITS.2022.3150328.