# Recursive Forward Dynamics for Serial Kinematic Chains using Conformal Geometric Algebra

Tobias Löw[1,2][0000−0003−4001−2770] and Sylvain Calinon[1,2][0000−0002−9036−6799]

[1] Idiap Research Institute, Martigny Switzerland.
{tobias.loew, sylvain.calinon}@idiap.ch
[2] EPFL, Lausanne, Switzerland

**Abstract.** The computation of the forward dynamics plays an important role in simulating the motion of interconnected rigid bodies while considering the physical properties and constraints of each part. The applications in robotics, graphics and animation usually require fast computation, which leads to the usage of fast recursive algorithms. In this paper, we present a formulation of the recursive forward dynamics of serial kinematic chains that is rooted in geometry, which allows coordinate-free view and geometrically meaningful interpretations of the involved quantities. The mathematical framework is called conformal geometric algebra (CGA) and it extends classical vector algebra by introducting a unified representation of a large array of geometric operations, transformations and mathematical objects, such as points, lines, and planes, in a rigorous yet intuitive manner. We validate the computation numerically and provide an implementation of the results in an open-source library, making it immediately available in practice.

**Keywords:** Geometric Algebra, Dynamics

## 1 Introduction

Simulation is an integral tool for robotic engineering since it allows for the safe testing and evaluation of algorithms. Simulating the motion of robots from forces using their dynamics is utilized in many different fields in robotics and it is an important part in the development of new algorithms for robotics. The problem of mapping forces to motion is solved by computing the forward dynamics of the robot, i.e. mapping the applied joint torques to joint accelerations for a given joint position and velocity. Consequently, there are many off-the-shelf robot dynamics simulators available, such as Raisim [1], Isaac Sim [2], Gazebo [3],

This work was supported by the State Secretariat for Education, Research and Innovation in Switzerland for participation in the European Commission's Horizon Europe Program through the INTELLIMAN project (https://intelliman-project.eu/, HORIZON-CL4-Digital-Emerging Grant 101070136) and the SESTOSENSO project (http://sestosenso.eu/, HORIZON-CL4-Digital-Emerging Grant 101070310).

Mujoco [4], Bullet [5] or Coppelia Sim [6]. 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 [7], KDL [8] and RBDL [9].

There are many applications where the forward simulation of robotic systems has become an integral part of algorithm. Often these forward simulations are run in parallel to obtain trajectory rollouts, which requires very efficient algorithms for the forward dynamics, such as in optimal control [10, 11] . Sampling-based model predictive control, that is built upon gradient-free optimizers such as model predictive path integral control [12], also heavily leverages parallel rollouts provided by simulators [13]. Rolling out the system dynamics is also necessary for dynamic programming approaches, including model predictive control [14], [15]. The same applies to motion planning where the dynamics are simulated forward in time to determine the effects of the planned control command [16]. Apart from robotics, the forward dynamics also play an important role in physics-based animation where they are used for high-quality character animations [17].

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 [18]. The manifold can be predefined, such as learning orientations [19], and leveraged in a dictionary of different manifolds [20]. In [21], 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 [22]. Riemannian motion policies are another geometric framework for reactive control [23]. 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 [24].

Apart from these geometric techniques, a mathematical framework called geometric algebra is also gaining traction in robotics research. In engineering, geometric algebra provides 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 [25]. In robotics, geometric algebra has been used for differential kinematics [26]. Inverse kinematics also serves as a good example of the geometric significance of geometric algebra objects, since the problem can be solved iteratively by using the intersection operations of the geometric primitives in the algebra [27]. Other applications of these geometric primitives are tools for medical robotics to provide geometric intuitive techniques for planning surgical paths [28]. There also already exists a link of

the previously mentioned use-cases of the forward dynamics simulation and geometric algebra in the form of modeling optimal control problems using geometric primitives [29]. Albeit, in that work, the dynamics were included after solving the optimal control problem in the form of Lagrangian inverse dynamics in order to obtain torque commands. Another interesting way to use the geometric primitives is for object manipulation from stereoscopic images [30]. Mathematically, geometric algebra presents fresh insights on screw theory [31], which is often used in robotics to express kinematic relationships.

Despite this growing popularity of geometric algebra in the field of robotics, there are still many algorithms that need to be derived in their geometric algebra version in order for it to become a single tool for computation in robotics. One of them is the efficient computation of the forward dynamics, which we are addressing in this paper. 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*[3] that we first presented in [29].

The rest of the paper is organized as follows: Section 2 presents related work, Section 3 introduces the mathematical background of geometric algebra and robot dynamics, in Section 4 we show our formulation of the inertia tensor in CGA, in Section 5 we then introduce the recursive dynamics algorithms in CGA, finally in Section 6 we discuss some mathematical implications.

## 2 Related Work

This paper describes the recursive computation of the forward dynamics in conformal geometric algebra. Since this is an important concept in robotics, naturally, there have been been many algorithms and implementations tackling this problem. Mathematically it can be solved by deriving the Lagrangian formulation of the robot dynamics and solving for the joint accelerations. This approach, however requires the inversion of the generalized mass matrix, which is usually not the most efficient way of computation [32]. Since these dynamics simulations are frequently used in applications that require a high computational efficiency, such as control, it is important that these accelerations are computed as fast as possible. The recursive formulation of solving the forward dynamics problem called the Articulated Body Algorithm (ABA) was first presented in Featherstone's seminal work [33] and has been shown to have complexity $\mathcal{O}(n)$. Our algorithms build on this work and we will explain in the mathematical discussions how the approaches differ, and where the advantages of CGA lie.

Researchers have realized before that mathematical tools from geometry can lead to simplified and unified treatments of the robot dynamics. By treating

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

$\boldsymbol{SE}(3)$ as a Lie group and adopting basic ideas from Riemannian geometry, a geometric variant of the robot dynamics was derived in [34]. 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 [35], 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 [36]. Our formulation in CGA not only retains this coordinate invariance, but also, as we will explain later, 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 [37]. A notable insight from that work is that by combining the geometric perspective from screw theory, the thorougness 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 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 [38] and multicopters [39]. 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 $\mathbb{G}_{0,6,2}$ [40]. 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.

## 3 Background

In this section we are briefly introducing robot dynamics and geometric algebra. We will use the following notation throughout the paper: $x$ to denote scalars, $\boldsymbol{x}$ for vectors, $\boldsymbol{X}$ for matrices, $X$ for multivectors and $\boldsymbol{\mathcal{X}}$ for matrices of multivectors.

### 3.1 Robot Dynamics

The manipulator equation for computing the dynamics of the system is

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

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 (1) for the joint accelerations $\ddot{\boldsymbol{q}}$, i.e.

$$\ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}(\boldsymbol{q})\Big(\boldsymbol{\tau} - \boldsymbol{\tau}_{\text{ext}} - \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \boldsymbol{g}(\boldsymbol{q})\Big). \tag{2}$$

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 Geometric Algebra

Geometric algebras, also known as Clifford algebras, are actually a family of algebras that can be found as the quotient algebras of tensor algebras over quadratic spaces. Historically, geometric algebra is a unification of Hamilton's quaternion algebra and Grassmann's exterior algebra. It is a single algebra for geometric reasoning, alleviating the need of utilizing multiple algebras to express geometric relations. Geometric algebras are defined for vector spaces $\mathbb{R}^{p,q,r}$ of dimension $n = p + q + r$, where $p, q$ and $r$ are the number of basis vectors that square to 1, -1 and 0, respectively.

The fundamental multiplication operation is called the geometric product

$$\boldsymbol{ab} = \boldsymbol{a} \cdot \boldsymbol{b} + \boldsymbol{a} \wedge \boldsymbol{b}, \tag{3}$$

and is the sum of an inner $\cdot$ and an outer $\wedge$ product. The outer product is equivalent to the exterior product of Grassman algebra and is an operation that spans subspaces. Therefore, unlike the vector algebra of $\mathbb{R}^{p,q,r}$, the associated geometric algebra $\mathbb{G}_{p,q,r}$ uses subspaces of the associated vector space as elements of computations. The basic elements, called blades, are found as the outer product of $k$ linearly independent vectors, which leads to $2^n = 2^{p+q+r}$ blades for a geometric algebra. The linear combination of basis blades is then called a multivector. The number $k$ is then called grade of the multivector.

The computation with subspaces leads to nullspace representations of geometric primitves w.r.t either the inner or the outer product. In CGA, these primitives include points, lines, planes, circles and spheres. Since they are not

within the scope of this paper, we refer to [41] for their defintion and construction and to [29] for their usage in optimal control applications in robot manipulation.

In this paper, we are using CGA to formulate the forward dynamics, which is $\mathbb{G}_{4,1}$. As it has four basis vectors that square to 1 and one that squares to -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. $\mathbb{R}^{4,1} = \mathbb{R}^3 \oplus \mathbb{R}^{1,1}$. This leads to several transformations beyond the rigid body transformations presented in the following section. We will discuss those transformations later in Section 6.4.

**Rigid Body Transformations** Rigid body transformations in CGA are achieved using a representation of the Lie group $Spin(3) \ltimes \mathbb{R}^3$ called motors, which is a double cover of the matrix Lie group of rigid body transformations in 3-dimensional Euclidean space $\boldsymbol{SE}(3)$, i.e. there exists a surjective two-to-one homomorphism from $Spin(3) \ltimes \mathbb{R}^3$ to $\boldsymbol{SE}(3)$. The associated Lie algebra of the motor group is a bivector algebra and is connected to the motor manifold by the exponential map and its inverse, the logarithmic map. The bivectors of the Lie algebra are essentially (dual) lines in CGA, i.e. they define the screw axis of the motor. Motors are also isomorphic to dual quaternions. The difference between them is that CGA is a larger algebra which introduces more geometric primitives and alleviates the need of special adjoint operations to transform them.

Motors can be applied to arbitrary multivectors in the algebra by using a sandwich product operation (similar to how quaternions rotate vectors)

$$Y = MX\widetilde{M}, \tag{4}$$

where $\widetilde{M}$ stands for the reverse of a motor, which can thought of as being similar to a conjugate quaternion. 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 CGA, as it extends linear transformations naturally from vectors to the entire algebra, i.e. motors 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.

Motors are used to represent the forward kinematics of a serial kinematic chain, i.e. the motor $M(\boldsymbol{q})$, given the configuration $\boldsymbol{q}$, can be computed with

$$M(\boldsymbol{q}) = \prod_{j=1}^{N} M_j(q_j). \tag{5}$$

The motors $M_j(q_j)$ are the current joint motors of the $j$-th joint given the joint position. They are found as

$$M_j(q_j) = \exp\left(q_j B_j\right), \tag{6}$$

where $B_j$ is the bivector describing the screw axis of the $j$-th joint. In the following we will be using $M_j$ instead of $M_j(q_j)$ for a shorter notation.

**Twists and Wrenches** Utilizing the terminology of screw theory, the screws that carry velocity and force information are called twists and wrenches, respectively. In CGA, they are represented as bivectors. The twists are identified with the bivectors that generate rigid body motions, i.e. the bivector that result from the logarithmic mapping of motors. Hence, their space is found as

$$\mathcal{V} \in \text{span}\{\boldsymbol{e}_{23}, \boldsymbol{e}_{13}, \boldsymbol{e}_{12}, \boldsymbol{e}_{1\infty}, \boldsymbol{e}_{2\infty}, \boldsymbol{e}_{3\infty}\}, \tag{7}$$

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 coscrews, 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 pseudoscalar $I_c = I\boldsymbol{e}_0$ [42], where $I$ is the standard pseudoscalar of CGA, i.e. $I = \boldsymbol{e}_{0123\infty}$. Multiplication of $I_c$ with a twist yields the space of wrenches as

$$\mathcal{W} \in \text{span}\{\boldsymbol{e}_{23}, \boldsymbol{e}_{13}, \boldsymbol{e}_{12}, \boldsymbol{e}_{01}, \boldsymbol{e}_{02}, \boldsymbol{e}_{03}\}. \tag{8}$$

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.

$$p = -\mathcal{V} \cdot \mathcal{W}. \tag{9}$$

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 (4) 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

$$\mathcal{V}' = \mathcal{V}_1 \times \mathcal{V}_2 \quad \text{and} \quad \mathcal{W}' = \mathcal{V} \times \mathcal{W}, \tag{10}$$

where $\times$ is the commutator product that is defined as

$$X \times Y = \frac{1}{2}(XY - YX). \tag{11}$$

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.

## 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 [37]. 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

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

Note that an inertia element uses the same basis blades as a wrench that was defined in Equation (8). 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.

$$\boldsymbol{\mathcal{I}} = \{\mathcal{I}^{\boldsymbol{e}_k}\}_{\boldsymbol{e}_k \in \mathbb{I}_{\mathcal{I}}}, \tag{13}$$

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

$$\mathbb{I}_{\mathcal{I}} = \{\boldsymbol{e}_{23}, \boldsymbol{e}_{13}, \boldsymbol{e}_{12}, \boldsymbol{e}_{01}, \boldsymbol{e}_{02}, \boldsymbol{e}_{03}\}. \tag{14}$$

### 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.

$$\mathcal{W} = \boldsymbol{\mathcal{I}}[\mathcal{V}] = -\sum_{\boldsymbol{e}_k \in \mathbb{I}_{\mathcal{I}}} (\mathcal{I}^{\boldsymbol{e}_k} \cdot \mathcal{V})\boldsymbol{e}_k. \tag{15}$$

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 (7) and (8), 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.

### 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.

$$M * \boldsymbol{\mathcal{I}} = \left\{ M \left( \sum_{\boldsymbol{e}_k \in \mathbb{I}_{\mathcal{I}}} \langle M \mathcal{I}^{\boldsymbol{e}_k} \widetilde{M} \rangle_{\boldsymbol{e}_j} \boldsymbol{e}_k \right) \widetilde{M} \right\}_{\boldsymbol{e}_j \in \mathbb{I}_{\mathcal{I}}}, \tag{16}$$

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

With equations (15) and (16), we can find the equivariance relationship

$$M \boldsymbol{\mathcal{I}} [\mathcal{V}] \widetilde{M} = (M * \boldsymbol{\mathcal{I}}) \left[ M \mathcal{V} \widetilde{M} \right]. \tag{17}$$

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.

### 4.3 Spatial Inertia

Traditionally, the spatial inertia is a $6 \times 6$ 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 the general inertia tensor $\boldsymbol{\mathcal{I}}$ as a function of $m$ and $\boldsymbol{I}$, i.e. the elements of $\boldsymbol{\mathcal{I}}$ are

$$\begin{aligned} \mathcal{I}^{\boldsymbol{e}_{23}} &= \ \ i_{xx}\boldsymbol{e}_{23} - i_{xy}\boldsymbol{e}_{13} + i_{xz}\boldsymbol{e}_{12}, & \mathcal{I}^{\boldsymbol{e}_{01}} &= m\boldsymbol{e}_{01}, \\ \mathcal{I}^{\boldsymbol{e}_{13}} &= -i_{xy}\boldsymbol{e}_{23} + i_{yy}\boldsymbol{e}_{13} - i_{yz}\boldsymbol{e}_{12}, & \mathcal{I}^{\boldsymbol{e}_{02}} &= m\boldsymbol{e}_{02}, \\ \mathcal{I}^{\boldsymbol{e}_{12}} &= \ \ i_{xz}\boldsymbol{e}_{23} - i_{yz}\boldsymbol{e}_{13} + i_{zz}\boldsymbol{e}_{12}, & \mathcal{I}^{\boldsymbol{e}_{03}} &= m\boldsymbol{e}_{03}. \end{aligned} \tag{18}$$

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 $M^{CoM}$

$$\boldsymbol{\mathcal{I}}^{CoM} = M^{CoM} * \boldsymbol{\mathcal{I}}. \tag{19}$$

### 4.4 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 multibody system [32]. 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 [43].

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**
  $\mathcal{W}_{B_{j+1}} = -\hat{\boldsymbol{\mathcal{I}}}_{j+1}[B_{j+1}](B_{j+1} \cdot \hat{\boldsymbol{\mathcal{I}}}_{j+1}[B_{j+1}])^{-1}$
  $\bar{\boldsymbol{\mathcal{I}}}_j = \left\{ \hat{\mathcal{I}}_{j+1}^{\boldsymbol{e}_k} + (B_{j+1} \cdot \hat{\mathcal{I}}_{j+1}^{\boldsymbol{e}_k}) \mathcal{W}_{B_{j+1}} \right\}_{\boldsymbol{e}_k \in \mathbb{I}_{\mathcal{I}}}$
  $\hat{\boldsymbol{\mathcal{I}}}_j = M_j^{CoM} * \boldsymbol{\mathcal{I}}_j + M_{j+1} * \bar{\boldsymbol{\mathcal{I}}}_j$
**end**

## 5 Recursive Dynamics Algorithms

In this section, we are presenting the recursive forward dynamics algorithm for serial kinematic chains using CGA. The recursive computation of the inverse dynamics of serial kinematic chains has already been presented in [44], 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.

### 5.1 Recursive Inverse Dynamics

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 (15). The joint torques $\tau_j$ 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**

### 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 $\tau_j$ 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**

### 5.3 Numerical Validation

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* [45], 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 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 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 |

## 6 Discussion

### 6.1 Computational Efficiency of the Algorithm

Since this algorithm is 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 [29]. 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.

### 6.2 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 [46]. 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 [46] 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 graphical 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 $4 \times 4$ 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 paper.

### 6.3 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 [47]. Instead of an inner product, the algebra defines a scalar product between the two spaces that, like the inner product of Equation (9), yields the power of the motion. The equivalent of the Lie bracket that we introduced for CGA in Equations (10) and (11) 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.

### 6.4 Clifford Algebra and the Conformal Model of Geometry

In this section, we want to emphasize the importance of Clifford algebra in physics and in particular with respect to Lie groups. To this end, in a first step we use the general geometric algebra $\mathbb{G}_{p,q}$ to introduce some basic concepts and afterwards we explain the implications w.r.t. CGA $\mathbb{G}_{4,1}$. First, we want to introduce versors, which are the geometric product of $n$ non-null vectors of $\mathbb{R}^{p,q}$, i.e. $V = \prod_i^n \boldsymbol{x}_i$ with $\boldsymbol{x} \in \mathbb{R}^{p,q}$. The set of versors together with the geometric product form the Clifford group $\Gamma(p,q)$. Versors that fullfill $V\widetilde{V} = \pm 1$, where $\widetilde{V}$ is the reverse operation that we previously introduced for motors, are known as unitary versors and form the pin group $Pin(p,q)$. The additional restriction to $V\widetilde{V} = 1$ yields the spin group $Spin(p,q)$ as a subgroup. The spin group actually is a representation of spinors, which are a fundamental concept in physics and more information can be found here [48]. In the context of this paper, it is relevant to note that $Pin(p,q)$ and $Spin(p,q)$ are double covers of the orthogonal group $\boldsymbol{O}(p,q)$ and the special orthogonal group $\boldsymbol{SO}(p,q)$, respectively. In the specific case of CGA as $\mathbb{G}_{4,1}$, the motors as a group form a subgroup of $Pin(4,1)$ and represent rotations and translations in $\mathbb{R}^3$. We also know that the conformal group $Conf(p)$, which includes orthogonal transformations, i.e. reflections and rotations, translations, dilations and special conformal transformations, i.e. multiplicative inversions, is isomorphic to the orthogonal group $\boldsymbol{O}(p+1,1)$. Thus $Conf(3) = \boldsymbol{O}(4,1)$ and the covering group $Pin(4,1)$ actually represents conformal transformations on $\mathbb{R}^3$. Hence, using CGA, we can perform conformal transformations on $\mathbb{R}^3$ which usually are highly non-linear and require slow algorithms to apply. So here, CGA effectively allows to express them using linear operators and via the versor representation reduces composition to multiplication of vectors. The conformal group has ten unique generators, which in CGA correspond to the ten bivectors of the algebra. Six of those are forming the space of twists, which are the Lie algebra to the motor subgroup. The three bivectors carrying the force information from the wrenches form the generators for hyperbolic rotations. Here, an interesting future research directions would be to uncover the implications of this connection. The last bivector $\boldsymbol{e}_{0\infty}$ is responsible for dilations, which means that CGA also contains the group of direct similarities

$Sim(3)$ as a subgroup. The fact that CGA contains non-rigid transformations could mean that the presented dynamics algorithms could be extended to more complex dynamics such as soft robotics or to model deformable objects.

## 7 Conclusion

In this paper, 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.

## References


1. Hwangbo, J., Lee, J. & Hutter, M. Per-Contact Iteration Method for Solving Contact Dynamics. *IEEE Robot. Autom. Lett.* **3,** 895–902 (Apr. 2018).
2. Liang, J. *et al. GPU-Accelerated Robotic Simulation for Distributed Reinforcement Learning* `http://arxiv.org/abs/1810.05762` (2024). Pre-published.
3. Koenig, N. & Howard, A. *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)* 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566). **3** (IEEE, Sendai, Japan, 2004), 2149–2154.
4. Todorov, E., Erez, T. & Tassa, Y. *MuJoCo: A Physics Engine for Model-Based Control* in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (Oct. 2012), 5026–5033.
5. Coumans, E. & Bai, Y. *PyBullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning* 2016–2021.
6. *V-REP: A Versatile and Scalable Robot Simulation Framework | IEEE Conference Publication | IEEE Xplore* `https://ieeexplore.ieee.org/document/6696520` (2024).
7. Carpentier, J. *et al. The Pinocchio C++ Library : A Fast and Flexible Implementation of Rigid Body Dynamics Algorithms and Their Analytical Derivatives* in *2019 IEEE/SICE International Symposium on System Integration (SII)* 2019 IEEE/SICE International Symposium on System Integration (SII) (IEEE, Paris, France, Jan. 2019), 614–619.
8. Smits, R. *KDL: Kinematics and Dynamics Library* http://www.orocos.org/kdl.
9. Felis, M. L. RBDL: An Efficient Rigid-Body Dynamics Library Using Recursive Algorithms. *Auton Robot* **41,** 495–511 (Feb. 1, 2017).
10. Carius, J., Ranftl, R., Farshidian, F. & Hutter, M. Constrained Stochastic Optimal Control with Learned Importance Sampling: A Path Integral Approach. *The International Journal of Robotics Research* **41,** 189–209 (Feb. 2022).
11. Mastalli, C. *et al. Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control* `http://arxiv.org/abs/1909.04947` (2022). Pre-published.
12. Williams, G., Aldrich, A. & Theodorou, E. A. Model Predictive Path Integral Control: From Theory to Parallel Computation. *Journal of Guidance, Control, and Dynamics* **40,** 344–357 (Feb. 2017).
13. Pezzato, C. *et al. Sampling-Based Model Predictive Control Leveraging Parallelizable Physics Simulations* `http://arxiv.org/abs/2307.09105` (2024). Pre-published.
14. Sleiman, J.-P., Farshidian, F. & Hutter, M. *Constraint Handling in Continuous-Time DDP-Based Model Predictive Control* in *2021 IEEE International Conference on Robotics and Automation (ICRA)* (May 30, 2021), 8209–8215.
15. Bjelonic, M. *et al.* Offline Motion Libraries and Online MPC for Advanced Mobility Skills. *The International Journal of Robotics Research,* 22.
16. Plaku, E., Kavraki, L. E. & Vardi, M. Y. Motion Planning With Dynamics by a Synergistic Combination of Layers of Planning. *IEEE Trans. Robot.* **26,** 469–482 (June 2010).
17. Liu, L., Yin, K., van de Panne, M., Shao, T. & Xu, W. Sampling-Based Contact-rich Motion Control. *ACM Transactions on Graphics (TOG)* **29,** 1–10 (2010).

18. Calinon, S. Gaussians on Riemannian Manifolds: Applications for Robot Learning and Adaptive Control. *IEEE Robotics Automation Magazine* **27,** 33–45 (June 2020).
19. Saveriano, M., Abu-Dakka, F. J. & Kyrki, V. Learning Stable Robotic Skills on Riemannian Manifolds. *Robotics and Autonomous Systems* **169,** 104510 (Nov. 1, 2023).
20. Ti, B., Razmjoo, A., Gao, Y., Zhao, J. & Calinon, S. A Geometric Optimal Control Approach for Imitation and Generalization of Manipulation Skills. *Robotics and Autonomous Systems* **164,** 104413 (June 1, 2023).
21. Jaquier, N., Rozo, L., Caldwell, D. G. & Calinon, S. Geometry-Aware Manipulability Learning, Tracking, and Transfer. *The International Journal of Robotics Research* **40,** 624–650 (Feb. 2021).
22. Beik-Mohammadi, H., Hauberg, S., Arvanitidis, G., Neumann, G. & Rozo, L. Reactive Motion Generation on Learned Riemannian Manifolds. *The International Journal of Robotics Research,* 02783649231193046 (Aug. 28, 2023).
23. Cheng, C.-A. *et al.* RMPflow: A Geometric Framework for Generation of Multitask Motion Policies. *IEEE Transactions on Automation Science and Engineering* **18,** 968–987 (July 2021).
24. Marić, F. *et al.* Riemannian Optimization for Distance-Geometric Inverse Kinematics. *IEEE Trans. Robot.* **38,** 1703–1722 (June 2022).
25. Baylis, W. E. in Ablamowicz, R. *et al. Lectures on Clifford (Geometric) Algebras and Applications* (eds Abłamowicz, R. & Sobczyk, G.) 91–133 (Birkhäuser Boston, Boston, MA, 2004).
26. Bayro-Corrochano, E. & Zamora-Esquivel, J. Differential and Inverse Kinematics of Robot Devices Using Conformal Geometric Algebra. *Robotica* **25,** 43–61 (Jan. 2007).
27. Aristidou, A. & Lasenby, J. FABRIK: A Fast, Iterative Solver for the Inverse Kinematics Problem. *Graphical Models* **73,** 243–260 (Sept. 2011).
28. Bayro-Corrochano, E., Garza-Burgos, A. M. & Del-Valle-Padilla, J. L. Geometric Intuitive Techniques for Human Machine Interaction in Medical Robotics. *Int J of Soc Robotics* **12,** 91–112 (Jan. 2020).
29. Löw, T. & Calinon, S. Geometric Algebra for Optimal Control With Applications in Manipulation Tasks. *IEEE Transactions on Robotics* **39,** 3586–3600 (2023).
30. Zamora-Esquivel, J. & Bayro-Corrochano, E. Robot Object Manipulation Using Stereoscopic Vision and Conformal Geometric Algebra. *Applied Bionics and Biomechanics* **8,** 411–428 (2011).
31. Delafosse, L. *A New Approach to Screw Theory Using Geometric Algebra* `https://hal.science/hal-04177875v3`. Pre-published.
32. Featherstone, R. & Orin, D. *Robot Dynamics: Equations and Algorithms* in *IEEE International Conference on Robotics and Automation (ICRA)* **1** (2000), 826–834.
33. Featherstone, R. *Robot Dynamics Algorithms* (Springer US, Boston, MA, 1987).
34. Park, F., Bobrow, J. & Ploen, S. A Lie Group Formulation of Robot Dynamics. *The International Journal of Robotics Research* **14,** 609–618 (Dec. 1995).
35. Ploen, S. & Bobrow, J. *An O(n) Geometric Algorithm for Manipulator Forward Dynamics* in *International Conference on Advanced Robotics (ICAR)* (July 1997), 563–568.
36. Müller, A. Screw and Lie Group Theory in Multibody Dynamics: Recursive Algorithms and Equations of Motion of Tree-Topology Systems. *Multibody Syst Dyn* **42,** 219–248 (Feb. 2018).

37. Afonso Silva, F. F., José Quiroz-Omaña, J. & Vilhena Adorno, B. Dynamics of Mobile Manipulators Using Dual Quaternion Algebra. *Journal of Mechanisms and Robotics* **14** (Sept. 14, 2022).
38. Bayro-Corrochano, E., Medrano-Hermosillo, J., Osuna-González, G. & Uriostegui-Legorreta, U. Newton–Euler Modeling and Hamiltonians for Robot Control in the Geometric Algebra. *Robotica* **40,** 4031–4055 (Nov. 2022).
39. Arellano-Muro, C. A., Osuna-González, G., Castillo-Toledo, B. & Bayro-Corrochano, E. Newton–Euler Modeling and Control of a Multi-copter Using Motor Algebra. *Adv. Appl. Clifford Algebras* **30,** 19 (Apr. 2020).
40. Selig, J. M. & Bayro-Corrochano, E. Rigid Body Dynamics Using Clifford Algebra. *Advances in Applied Clifford Algebras (ACAA)* **20,** 141–154 (Mar. 2010).
41. Perwass, C. *Geometric Algebra with Applications in Engineering Geometry and Computing* **4** (Springer, Berlin, 2009).
42. Hestenes, D. in *Geometric Algebra Computing: In Engineering and Computer Science* (eds Bayro-Corrochano, E. & Scheuermann, G.) 3–33 (Springer, London, 2010).
43. Featherstone, R. *Rigid Body Dynamics Algorithms* (Springer US, Boston, MA, 2008).
44. Bayro-Corrochano, E. *Robot Modeling and Control Using the Motor Algebra Framework* in *International Workshop on Robot Motion and Control (RoMoCo)* (July 2019), 1–8.
45. Ansel, J. *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. 27, 2024), 929–947.
46. Dantam, N. T. Robust and Efficient Forward, Differential, and Inverse Kinematics Using Dual Quaternions. *The International Journal of Robotics Research* **40,** 1087–1105 (Sept. 2021).
47. Featherstone, R. A Beginner's Guide to 6-D Vectors (Part 1). *IEEE Robot. Automat. Mag.* **17,** 83–94 (Sept. 2010).
48. Vaz Jr., J. & Rocha Jr., R. da. *An Introduction to Clifford Algebras and Spinors* First edition. 242 pp. (Oxford University Press, Oxford, 2016).