Geometric Structures for Learning and Optimization in Robotics

Sylvain Calinon

Idiap Research Institute, Martigny, Switzerland; email: sylvain.calinon@idiap.ch

Published in the Annual Review of Control, Robotics, and Autonomous Systems, Volume 9, pp. 123–146.

https://doi.org/10.1146/annurev-control-022624-013742

Keywords

signed distance fields, implicit shape representations, geometric algebra, Riemannian manifolds, skill transfer, robot learning, model-based optimization

Abstract

This article presents an overview of geometric approaches to facilitate the acquisition and transfer of robot skills. It focuses on three complementary geometric frameworks: signed distance fields, geometric algebra and Riemannian geometry, which provide representations facilitating learning, planning, control and optimization problems in robotics. The first consists of representing shapes in an implicit manner through the use of a distance function, where different approaches can be used to encode and learn this function. The second, geometric algebra, is linked to Clifford algebra and allows basic geometric primitives to be treated in a unified manner, including 6D poses, planes, lines, circles, and spheres, which can represent various forms of constraints in robot applications. The third leverages the use of Riemannian manifolds to extend models and algorithms originally developed for standard Euclidean data to curved spaces. These manifolds can represent a variety of geometric objects in robotics, not only for structured objects such as spheres, matrices and subspaces, but also for more generic smooth manifolds described by a Riemannian metric to measure distances. The article discusses the distinctions and connections between these geometric approaches and shows how they can contribute to various problems in robotics, with a focus on manipulation tasks.

Contents

1. INTRODUCTION 2
2. REPRESENTATIONS OF SHAPES 2
3. DISTANCE FIELDS 3
   3.1. Modeling approaches for signed distance fields 5
   3.2. Extensions of Distance Functions Beyond 3D Object Shape Modeling 7
4. GEOMETRIC ALGEBRA 8
   4.1. Motivation 9
   4.2. Mathematical overview 10
   4.3. Practical usage and applications 12
5. RIEMANNIAN MANIFOLDS 14
   5.1. Mathematical overview 16
   5.2. Homogeneous manifolds 17
   5.3. Inhomogeneous manifolds 20
6. CONCLUSION 22

1. INTRODUCTION

Despite significant advances in AI, robots still struggle with tasks involving physical interaction. Robots can easily beat humans at board games such as chess or Go but are mostly incapable of skillfully moving the game pieces by themselves—the part of the task that humans subconsciously succeed in. Learning and transferring manipulation skills is hard because the movement behaviors that the robots need to acquire are tightly connected to our physical world and to embodied forms of intelligence.

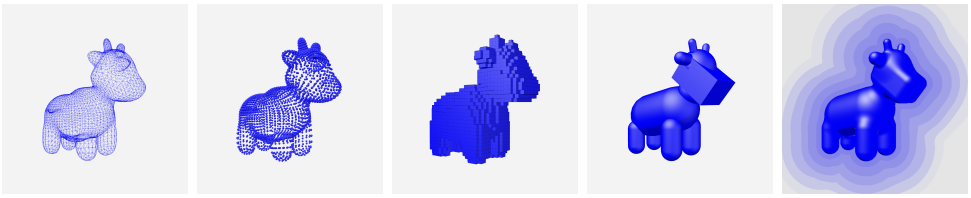
Acquiring manipulation tasks encompasses a wide range of learning techniques, from foundational models with large-scale datasets (1, 2, 3) to model-based optimization and frugal learning techniques that rely on a handful of demonstrations or exploration trials (4). Despite the difference in scale, these models can benefit from a clever use of the underlying model and algorithmic structures. Different terms are used in the literature to refer to such guidance, including prior knowledge, inductive biases, models, and representations. The research challenge is to design these representations without limiting the generalization, adaptation and processing speed capabilities of the system.

This article focuses on geometric priors, which can be viewed as a subset of physics-informed models. It also concentrates principally on manipulation skills, providing an overview of geometric representations and associated learning approaches to help robots acquire skills by imitation and self-refinement.

2. REPRESENTATIONS OF SHAPES

Geometry is a branch of mathematics concerned with properties of space such as shapes, distances, sizes, relative locations, symmetries and projections. The most fundamental objects for building geometry include points, lines, planes, curves and surfaces such as spheres. Several approaches can be used to describe more complex objects, including meshes, point clouds, occupancy grids, and compositions using shape primitives (see **Figure 1**).

Meshes are typically used as models of the robot and in situations where the objects, tools or elements of the surrounding environments are known and can be provided explicitly



(a) Mesh (b) Point cloud (c) Occupancy (d) Primitives (e) Distance field

Figure 1: Diverse representations of shapes used in robotics applications.

in this format (e.g., from CAD models). Point clouds are typically used when depth cameras or laser sensors are used to model the robot's surroundings. Meshes and point clouds differ mainly in the connectivity information between the points, which is absent in point clouds. As an alternative, a 2D or 3D occupancy grid can also be used as a set of regularly organized voxels to determine the presence or absence of an object at each voxel location. This representation is often used with mobile robots by defining the 2D or 3D grid in a moving local region in front of the robot. This information is also often compressed by relying on quadtree/octree representations (5).

The combination of shape primitives is another useful representation that is often very efficient computationally when the application allows object shapes to be coarsely represented as a composition of geometric primitives such as spheres, boxes or capsules (6). This is, for example, the case for obstacle avoidance, where the volume of the primitives can be deliberately increased so that the composition defines a virtual barrier function that should not be entered. It is also the case for the manipulation of objects that are deformable and/or whose 6D pose is hard to estimate, meaning that the inaccuracy of approximating the shape with a limited set of primitives has limited impact on the overall task. Shape primitives can also be used to model subparts of the objects (e.g., to represent the local contacts with the object surfaces in a grasping task). While the use of shape primitives is appealing from a computation perspective, the hard problem is to automate the modeling of this composition, which requires a system to estimate how many primitives are required, which primitives to use and where these primitive should be placed (6D poses and scaling factors). Moreover, while the composition of shape primitives is often additive, other Boolean operations can also be considered (e.g. by modeling a volume as the intersection or subtraction of two shape primitives), which increases the representation capability (expressiveness) but also increases the difficulty for automatic shape approximation. In **Figure 1d**, the shape is approximated by an additive composition of capsules, a sphere, and a box.

In contrast to the representations described above, shapes can also be implicitly represented by a distance function, which is discussed in the next section. The general idea is to describe the shape as a function that takes 3D coordinates as input and provides a scalar value as output, describing the distance to the shape (or other geometric objects), see **Figure 1e**. This distance can optionally be signed to distinguish between points inside and outside a closed shape.

3. DISTANCE FIELDS

The use of implicit shape representations based on distances is widespread in computer graphics and robotics to represent objects. It is defined as a distance function $d = f(\boldsymbol{x})$

representing a distance field, which typically takes as input the 3D coordinates $\boldsymbol{x}$ of a point and returns a scalar distance d, which corresponds to the distance between $\boldsymbol{x}$ and the closest point on the surface of the object. The ensemble of points returning the same value is called a level set, where the zero level set correspond to the surface of the modeled object.

In computer graphics, the concept of assigning a distance value to the pixels of an image can be traced back to the 1960s (7) and was later extended to implicit surface modeling (8) and robotics (9). For computer graphics applications, the distance function is used principally for rendering, with the main goal of visualizing the zero level set of the function within the entire scene (namely, by displaying the whole set of points at distance zero forming the surface to be rendered).

In robotics, the goals are often different and more varied. For manipulation, we can, for example, be interested in calling the distance function for a subset of points, such as testing whether the end effector of the robot is inside an object or far from an obstacle. Often, fewer points need to be tested, often iteratively as the robot moves, without having to render the whole scene, and without caring specifically about zero level sets. For example, an impedance controller can be designed by setting a negative level set as the virtual distance target to reach, effectively resulting in a compliant controller that will make the robot apply a constant pressure along the surface (10). The resulting behavior can be interpreted as a virtual spring where one attachment point is on the robot gripper and the other can slide on the surface of a negative level set (i.e., at a given distance inside of the object), so that the virtual spring produces a given force when the gripper is in contact with the surface of the object. The distance field can similarly be used to generate virtual guides in control and teleoperation applications, such as moving while maintaining contact with the surface, staying a desired distance from the surface, or staying within a desired distance range from the surface.

Since many learning, control, planning and estimation problems in robotics are based on cost functions measuring distances, it is not surprising that the use of a distance field as an underlying geometric representation of shapes has been widely adopted by the robotics community. Distance fields have consequently been used in applications such as motion planning (11), collision avoidance (12), reactive control (13, 14), manipulation (15), grasping (16), and mapping and odometry (17). The implicit representation of shapes as a distance function also provides easy mathematical operations, such as changes of coordinate systems and Boolean compositions (union, subtraction, etc.) based on *min* and *max* operators (18).

A *signed distance field* (SDF) can be estimated from CAD models or from contour points extracted by vision. In both cases, the vertices form a limited set of data as they only correspond to the zero level set of the SDF. To cope with this limited set, eikonal and tension terms can be added in the cost function as prior knowledge (19, 20, 21). The eikonal equation $\|\boldsymbol{\nabla} f\| = 1$ is based on the gradient of the distance function f. It constrains the norm of the gradient to be 1, which intuitively means that at a given point at some distance to the shape, if we move a bit away from the shape, we expect the distance to increase by the same amount. Similarly, the SDF can be built by a diffusion process based on Laplace–Beltrami operators, which is compatible with various geometric representations (22, 23, 24).

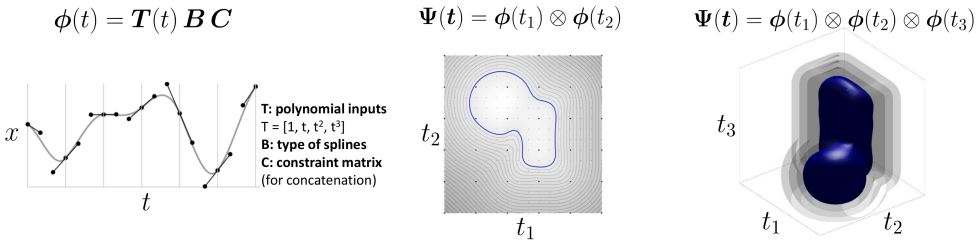


Figure 2: Signed distance field (SDF) constructed from analytic basis functions. *Left:* Concatenation of 6 cubic Bézier curves with 1D inputs (time variable t). $\boldsymbol{\phi}(t)$ is a row vector containing the basis functions evaluated at time t. If $\boldsymbol{w}$ is a vector containing the superposition weights, the signal is constructed as $x(t) = \boldsymbol{\phi}(t)\boldsymbol{w}$. *Center:* Concatenation of 5×5 cubic Bézier curves with 2D inputs. Several equidistant contours are displayed as closed paths, with the one corresponding to the object contour (distance zero) represented in blue. Encoding each dimension with basis functions defined by $\boldsymbol{\phi}(t_i)$ allows the signal $x(t)$ (i.e., the distance function) to be constructed as $\boldsymbol{x}(t) = \boldsymbol{\Psi}(t)\boldsymbol{w}$, with $\boldsymbol{\Psi}(\boldsymbol{t}) = \boldsymbol{\phi}(t_1) \otimes \boldsymbol{\phi}(t_2)$, where $\otimes$ is the Kronecker product. *Right:* Concatenation of $5 \times 5 \times 5$ cubic Bézier curves with 3D inputs. Several isosurfaces are displayed as 3D shapes, with the one corresponding to the object surface (distance zero) represented in blue. Each cubic Bézier curve corresponds to the superposition of four Bernstein basis functions, with superposition weights acting as control points (displayed as black points in the 1D example), and constraints on the control points ensuring continuity.

3.1. Modeling approaches for signed distance fields

Several encoding strategies have been proposed to model the distance functions, which are described in this section.

3.1.1. Grid-based encoding of signed distance fields. A first approach is to consider a discretization of the space, similarly to the occupancy grid in **Figure 1c**, with each voxel encoding a distance value instead of a binary value. As with occupancy grids, the distance information can be compressed. Several approaches have been proposed, including voxel hashing (25, 12) and tensor decomposition (26). The first category typically involves truncating the SDF around the surface of the shapes to encode (where the SDF is then called a truncated SFD), so that the majority of data stored in the regular voxel grid can be marked as free or unobserved space. This truncation satisfies applications in which we only care about distances in close proximity to the shapes (such as obstacle avoidance), but it can be limiting in situations that require estimating distances in the whole space (such as reaching tasks).

3.1.2. Polynomial encoding of signed distance fields. Another approach consists of approximating the distance function as a polynomial function that instead takes a position variable and outputs a scalar distance value (27, 20). Instead of specifying a high-order polynomial function, the encoding most often relies on piecewise polynomial functions, where the desired degree of continuity can be easily specified.

One advantage of such an analytic expression is that the derivatives can also be computed analytically, which can be very useful for control and planning (20). Indeed, the first-order derivative (gradient) provides the direction vector to move closer to or away from objects, which can also be used to orient the robot end effector to be locally aligned with the surface of the object (e.g., to drill a hole perpendicular to the surface). The

second-order derivative provides a Hessian matrix that can be used directly in Newton's optimization methods.

Several choices of polynomial exist. For trajectories (characterized by 1D time input and multivariate outputs), this choice includes, for example, Bézier curves (Bernstein polynomials) or B-splines, which can be formally described as a weighted superposition of basis functions specified by a constrained matrix. **Figure 2** shows an example with Bernstein basis functions.

3.1.3. Gaussian process encoding of signed distance fields. *Gaussian process implicit surface* (GPIS) is a regression technique that was originally proposed in the field of computer graphics to model shapes in a probabilistic manner (28). The implicit surface is built from a set of data points annotated with the values 0, 1, or -1 to define whether each point is on the border, inside the shape, or outside the shape, respectively (or alternatively, with surface normals associated with points on the contour). The model can then be used to solve a *Gaussian process regression* problem, by estimating the proximity of a new given point to the surface of the shape, together with a measure of uncertainty about this proximity that can be exploited for robust planning/control (29) and active learning (30). Note that this estimate provides a proximity information rather than a real distance, since it takes into account that in GPIS, the training points are typically given as 0, 1 and -1 rather than as real distances. Reformulations of GPIS have been proposed to provide more faithful Euclidean distance fields, such as considering a log-Gaussian process in the computation (17).

Similarly to polynomial encoding, GPIS can be exploited to analytically estimate the direction to move toward or away from the surface (31, 32). In addition, the Gaussian process can be exploited to define prior information about the shape, which is useful when only few points are available to model the shape or when data points representing a portion of the shape are missing (33). GPIS has been employed in various robot applications, including grasp control (31, 32), grasp planning (29), navigation (17), object detection, and terrain classification (30). Similar regression approaches have also been used, such as support vector regression, for polishing and bimanual lifting tasks (34).

GPIS can be extended in various ways by exploiting the underlying Gaussian process. For example, the distance matrix, at the core of the process, can be factorized as an eigendecomposition (35), which can be exploited to sort shape information with an increasing amount of detail.

3.1.4. Neural Network encoding of signed distance fields. A large body of work has explored the use of neural networks as distance function approximators. Applications include learning shapes (36), reactive motion generation, whole-body control, and safe human–robot interaction in shared workspaces (14), manipulation planning by modeling pair collisions between objects (15), reactive control for manipulation (13), joint grasp and motion planning (16), manipulation using contact points on the whole body of the robot (21), and manipulation planning in the configuration space (37).

In comparison with the other encoding schemes presented above, the advantages of the neural network approximation concern the inference speed (which allows fast evaluation of distances) as well as the larger number of input variables that can be considered in the network (which enables many possible extensions, some of which are discussed in the next section).

3.2. Extensions of Distance Functions Beyond 3D Object Shape Modeling

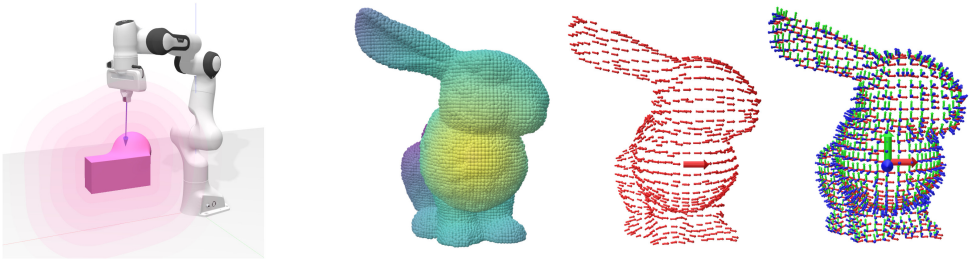


Figure 3: *(a)* Learning and control with an implicit shape representation of the surrounding environment, formulated as a signed distance function (SDF), depicted as pink level sets. *(b)* Distance field modeled on the surface of an object instead of a standard Euclidean space, used to measure the distance and geodesic paths between two points on the surface of the object. *(c)* Extension to a vector field. *(d)* Extension to a multivector field (a geometric algebra object), exploited to define a smoothly varying reference frame depicted as a coordinate system with three basis vectors (for details, see (38)).

Representing shapes in the form of a distance function facilitates the Boolean composition of multiple shapes by using simple *min* or *max* operators. Combining this property with an explicit rotation and translation transform at the level of the inputs allows an articulated chain to be easily presented as an SDF, with an automatic adaptation to the joint angle configuration (21). This can be used to model robot or human bodies as well as articulated objects or furniture (doors, windows, drawers, etc.). Moreover, neural networks can go beyond 3D shape modeling by considering distance fields in configuration spaces of higher dimensions, such as joint angle state spaces (13, 37), which bring a representation with tight links to planning and control problems acting directly in the robot configuration space. Koptev *et al.* (13) exploited this approach to enable fast reactions to perturbations in a manipulation task, and Li *et al.* (37) used it for rapid computation of whole-body manipulation skills, demonstrated in bimanual lifting of bulky objects and in interception behaviors (for a robotic goalkeeper).

Distance fields can also be used with open surfaces (39) and curves (40). In this case, the distance field takes an unsigned form (called an *unsigned distance field*). In robotics, open surfaces can, for example, represent clothes, textiles; or flat, flexible objects. Curves can, for example, represent motion trajectories, the shape of entangled cables, or the pose of a continuum robot.

Another extension consists of specifying a subset of points for the robotic task to achieve and implicitly modeling the state of the system or the movement to perform this task as pairwise distances (41, 42). This relative distance space aims to capture the important relations between the points of interest, which can, for example, correspond to the different articulations of the robot or to multiple objects used in the task. Similarly to using kernel functions in GPIS, this implicit representation defines distance matrices that can be processed to extract the principal relations between the points or to compress the information to extract the most invariant relationships.

It is also possible to go beyond scalar Euclidean distances by relying on a similar computation pipeline to extend the notion of distance to other manifolds, and/or to transport information other than distances in the diffusion process. **Figure 3** illustrates how the

computation of a distance field by diffusion (through a Laplace–Beltrami operator) can be extended to the diffusion on other manifolds and for other sources of information, including vectors and more elaborated descriptors (quaternions, multivectors in geometric algebra, orientation matrices, or ellipsoids) (38).

Similarly, models such as *neural radiance fields* (NeRFs) (43) and *Gaussian splatting* (44) replace the distance information by jointly encoding volume density and color, offering another alternative in robotics for compact and continuous representations for shapes and scenes (45).

4. GEOMETRIC ALGEBRA

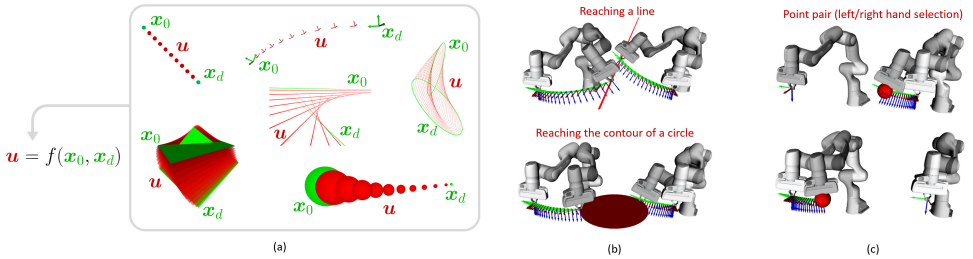


Figure 4: *(a)* Geometric algebra (GA) encodes geometric primitives in a uniform manner, including points, end-effector poses, lines, planes, circles or spheres, as well as the associated transformations $\boldsymbol{u}$ (called motors, shown in red) to move from an initial state $\boldsymbol{x}_0$ to a desired state $\boldsymbol{x}_d$ (shown in green). These geometric objects can be used to define constraints (e.g., staying on the surface of a sphere corresponds to the constraint of maintaining a given distance to the center). In practice, this means that the same function $\boldsymbol{u} = f(\boldsymbol{x}_0, \boldsymbol{x}_d)$ is used for these different geometric objects, which is useful in various manipulation contexts, such as the coordination of multiple end effectors, manipulation with contacts along the whole robot body, and human–robot collaboration. *(b)* GA can also be employed for bimanual skills (46), by providing a geometric representation for various forms of coordination (here, with two robot manipulators mounted on a table). The initial robot poses are depicted in white and the final poses are in gray. *(c)* In addition to classical geometric objects, GA can also be used to define point pairs, which is useful in the context of bimanual hands to automatically determine which hand is most appropriate to use in a manipulation task. Indeed, the use of point-pair primitives enables automatic decision-making on which arm should reach for a target without requiring the use of conditional statements (46).

Another candidate to efficiently handle geometric structures in robotics is the framework of **geometric algebra** (GA), whose roots can be found in Clifford algebra, which can be seen as a fusion of quaternion and Grassmann algebras. The problem of representing rotations in 3D was studied by several mathematicians in the nineteenth century, with Hamilton developing his quaternion algebra and Grassmann developing an algebra based on outer products, which later in the century was extended by Clifford, who combined aspects of both within a single structure, effectively combining the advantages of both quaternions and vector geometry. These results were largely hindered by quaternion algebra for interesting historical reasons that are discussed in (47).

The seminal work of Hestenes and colleagues (48, 49) introduced the modernization of Clifford algebra that was later termed geometric algebra. As an entry point to the field, Lasenby *et al.* (47) provides a general introduction to GA, as well as a historical perspective. A wider survey of GA in engineering applications is also provided in (50). Examples of applications in robotics are detailed in (51).

GA unifies multiple frameworks popular in robotics, such as screw theory, Lie algebra or dual quaternions, while offering excellent possibilities for generalization and extension. GA is part of an increasing prominence of geometric methods in robotics, by integrating geometric primitives (e.g. points, lines, and planes) and transformations (e.g. rotations and translations) into a single, high-level mathematical language for geometric reasoning. In robotics, this representation is thus an interesting candidate to provide a single algebra for geometric reasoning, alleviating the need to use multiple algebras to express geometric relations.

4.1. Motivation

This section aims to motivate the use of GA by providing a brief conceptual overview and informal guidance on its usage. More formal and practical guides to the use of GA in robotics are available in (52, 53).

For practitioners in robotics, the easiest way to motivate the use of GA is to start from the widely adopted representation of orientation as a unit quaternion. While a quaternion can commonly be stored on the computer as four variables, it is mathematically a number, similar to a complex number with a real and imaginary part stored as two variables. Quaternions come with an algebra as a set of operations and transformation rules. It is common in robotics to represent the orientation of a mobile robot or the end effector of a robot as a unit quaternion, so that the vector formed by the four variables has a unit norm. Measuring distances between unit quaternions, or computing the actions to move from one unit quaternion to another, requires the associated quaternion algebra, which explains why these operations are typically coded differently for the 3D Cartesian position of the robot (using standard linear algebra) and its orientation. With some shortcuts, a unit quaternion can be geometrically interpreted as a point on a sphere in a 4D space. This interpretation ignores the double coverage property of the quaternion used to represent orientation, with q and $-q$ representing the same orientation.

Similarly to quaternions, geometric algebra comes with geometric objects and an associated algebra to make operations on these objects. Before we examine this further, the dual quaternion is another relevant intermediary representation to introduce. Dual quaternions naturally arose in robotics, stemming from the same motivation of avoiding the use of different rules and algebras for translation and orientation information (54, 55, 56). Since most robotics applications require considering both position and orientation as a state space (e.g., the 6D pose of a robot gripper), the use of dual quaternions efficiently homogenizes computation by removing the need to compute operations with different algebras and convert geometric objects from one algebra to another.

While it is common in practice to store a unit quaternion as four variables, only three parameters are truly independent. Similarly, a dual quaternion stores the pose information as eight variables, where only six parameters are independent. The robotics applications of GA that we will describe next instead use objects stored with 32 variables. In the same way as dual quaternions can encode more elaborate objects than unit quaternions, GA extends this modeling capability one step further by also allowing lines, planes, circles and spheres to be encoded by following a similar principle.

Interestingly, GA does not require the notion of imaginary numbers in its development. While the theory may appear unfamiliar at first sight, its underlying concepts are actually simpler and more intuitive than the use of quaternions as an extension of complex numbers.

GA is based on a multiplication operation called the geometric product, composed of an inner product and an outer product. The latter describes an oriented plane/volume that extends and generalizes the cross product (which is restricted to only three dimensions).

The resulting elements are called *multivectors*. **Figure 4** illustrates how this representation can be used to encode geometric primitives in a uniform manner (e.g., as points, poses, lines, planes or spheres, depicted in green), as well as the associated transformations $\boldsymbol{u}$ (called motors, depicted in red) to move from an initial state $\boldsymbol{x}_0$ to a desired state $\boldsymbol{x}_d$. Similarly to dual quaternions, operations on these objects can be treated in the same way, without requiring us to switch between different algebras. In the depicted example, the practical result is that the interpolation function $\boldsymbol{u} = f(\boldsymbol{x}_0, \boldsymbol{x}_d)$ is coded only once and automatically generalizes to all geometric objects represented in the figure. These geometric operations yield compact codes and can be computed quickly (52).

Several variants of GA can be formulated, requiring different numbers of dimensions, by offering the opportunity to treat different categories of geometric problems while keeping the same base formulation. Using a *conformal geometric algebra* (CGA) representation enables data to be stored in a vector of 32 dimensions, which allows us to describe a rich variety of geometric objects, including constraints specified as spheres, circles, planes, lines or segments.

As a crude illustration, we can think first of the way we represent the position of data points in a Cartesian space as a 3D vector. If we set some of these entries to zero in the vector, then we can represent points in specific 2D planes of the 3D space (e.g, by representing a planar path in x–y and ignoring the z axis). In analogy to this 3D Cartesian vector, the non-zero values in this 32D vector determine which geometric object it represents. Interestingly, both states and actions use the same formulation, providing a generic and homogeneous formulation for various robotics problems (kinematic and dynamic control, learning, optimization, and planning).

In contrast to quaternion algebra whose theory relies on a generalization of complex numbers (the imaginary i, j, and k components of the quaternion), the theory behind geometric algebra relies on a simpler principle that uses basis vectors as a starting point to describe different directions, which are combined with the notion of inner and outer products. In standard linear algebra, the inner/dot product of two vectors gives a scalar, while the outer product of two vectors typically takes the form of a cross product for 3D vectors, which results in another 3D vector whose length is proportional to the surface swiped by the two vectors (in the plane formed by these two vectors). In geometric algebra, the notion of an outer product is generalized to any dimension. First, the result is not expressed as a vector: An oriented surface is instead defined so that the ordering of the two vectors matters. Then, multiple consecutive outer products can be defined (e.g., representing an oriented volume for the outer product of three vectors). This composition of the original basis vectors allows the formation of a higher-dimension space in which many different geometric objects can be defined in a uniform manner.

4.2. Mathematical overview

The geometric product $\boldsymbol{ab} = \boldsymbol{a} \cdot \boldsymbol{b} + \boldsymbol{a} \wedge \boldsymbol{b}$ is composed of an inner product $\boldsymbol{a} \cdot \boldsymbol{b}$ and an outer product $\boldsymbol{a} \wedge \boldsymbol{b}$ between two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. Grassmann's outer product $\boldsymbol{a} \wedge \boldsymbol{b}$ defines a new quantity called the bivector, representing the area swept by the two vectors as a directed/signed quantity that depends on the ordering (namely, $\boldsymbol{a} \wedge \boldsymbol{b} = -\boldsymbol{b} \wedge \boldsymbol{a}$), thus

encoding the notion of an oriented plane. By extension, a trivector $\boldsymbol{a} \wedge \boldsymbol{b} \wedge \boldsymbol{c}$ defines an oriented volume, and similarly for higher dimensions.

$\boldsymbol{a} \cdot \boldsymbol{b}$ and $\boldsymbol{a} \wedge \boldsymbol{b}$ thus carry different quantities, which are gathered as a geometric product $\boldsymbol{ab}$, in the same way that a complex number carries a real and an imaginary component that are gathered as a complex number, where the notation $x + y\,i$ does not mean that x can be added to $y\,i$ as a standard operation with real numbers. Similarly $\boldsymbol{a} \cdot \boldsymbol{b} + \boldsymbol{a} \wedge \boldsymbol{b}$ defines a geometric product, but the inner and outer products carry different quantities.

This linear combination of objects of any type (scalar, bivector, trivector, etc.) is generically called a *multivector*, which allows the representation of a wide range of concepts and quantities in physics and engineering. As a means of introducing GA, a famous example is the one from the field of fluid dynamics showing that with a multivector representation, all four of Maxwell's equations are reduced to a single equation in the language of geometric algebra (57).

4.2.1. Rotation operations. For unit vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, we can first notice that if $\boldsymbol{a}$ and $\boldsymbol{b}$ are the same vector, then $\boldsymbol{ab} = 1$. We can also observe that if $\boldsymbol{a}$ and $\boldsymbol{b}$ are perpendicular, then the expression simplifies to $\boldsymbol{ab} = \boldsymbol{a} \wedge \boldsymbol{b}$. Since $\boldsymbol{a} \wedge \boldsymbol{b} = -\boldsymbol{b} \wedge \boldsymbol{a}$, we can also observe in this case that $\boldsymbol{ab} = -\boldsymbol{ba}$ and that $(\boldsymbol{ab})(\boldsymbol{ab}) = \boldsymbol{a}(\boldsymbol{ba})\boldsymbol{b} = -\boldsymbol{a}(\boldsymbol{ab})\boldsymbol{b} = -(\boldsymbol{aa})(\boldsymbol{bb}) = -1$, which can be summarized as the squared expression $(\boldsymbol{ab})^2 = -1$, which is reminiscent of the expressions we know for complex numbers and quaternion algebra with the imaginary unit (i.e. $i^2 = -1$) but was here derived very simply with two orthonormal vectors, without any notion of imaginary units.

When applied to a set of basis vectors $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n\}$, the geometric product then yields

$$\boldsymbol{e}_i \boldsymbol{e}_j = \begin{cases} 1 \text{ for } i = j, \\ \boldsymbol{e}_i \wedge \boldsymbol{e}_j \text{ for } i \neq j. \end{cases}$$

The set of basis elements $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n\}$ can be expanded to a basis of $\mathcal{G}_n$ that contains:

scalar:	1
vectors:	$\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n$
bivectors:	$\{\boldsymbol{e}_i \wedge \boldsymbol{e}_j\}, \quad \forall i, j,\ i \neq j$
trivectors:	$\{\boldsymbol{e}_i \wedge \boldsymbol{e}_j \wedge \boldsymbol{e}_k\}, \quad \forall i, j, k,\ i \neq j \neq k$
$\vdots$	
n-blade:	$\boldsymbol{e}_1 \wedge \cdots \wedge \boldsymbol{e}_n$ (pseudoscalar I_n)

For $\mathbb{R}^3$, if we use the notation $\boldsymbol{e}_{ij} = \boldsymbol{e}_i \wedge \boldsymbol{e}_j$, then the geometric algebra $\mathcal{G}_3$ has the following basis: $\{1, \boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3, \boldsymbol{e}_{12}, \boldsymbol{e}_{13}, \boldsymbol{e}_{23}, I_3\}$, where the bivectors play an important role as they can be used to describe spatial rotations (called motors or rotors).

Indeed, we can see this by using as multivector a linear combination of a scalar and a bivector component $\boldsymbol{r} = \exp(\frac{\theta}{2}\boldsymbol{e}_{21}) = \cos(\frac{\theta}{2}) - \sin(\frac{\theta}{2})\,\boldsymbol{e}_{12}$, together with the corresponding complement $\tilde{\boldsymbol{r}} = \exp(\frac{\theta}{2}\boldsymbol{e}_{12}) = \cos(\frac{\theta}{2}) + \sin(\frac{\theta}{2})\,\boldsymbol{e}_{12}$. With this pair of multivectors, the operation $\boldsymbol{v}' = \boldsymbol{r}\boldsymbol{v}\tilde{\boldsymbol{r}}$ will rotate the vector $\boldsymbol{v} = v_1\boldsymbol{e}_1 + v_2\boldsymbol{e}_2$ by an angle θ in the plane described by $\boldsymbol{e}_{12}$, which can be calculated as

$$\begin{aligned} \boldsymbol{r}\boldsymbol{v}\tilde{\boldsymbol{r}} &= \big(\cos(\tfrac{\theta}{2}) - \sin(\tfrac{\theta}{2})\,\boldsymbol{e}_{12}\big)\,\big(v_1\boldsymbol{e}_1 + v_2\boldsymbol{e}_2\big)\,\big(\cos(\tfrac{\theta}{2}) + \sin(\tfrac{\theta}{2})\,\boldsymbol{e}_{12}\big) \\ &= \big(v_1\cos(\theta) - v_2\sin(\theta)\big)\,\boldsymbol{e}_1 + \big(v_1\sin(\theta) + v_2\cos(\theta)\big)\,\boldsymbol{e}_2 \end{aligned}$$

whose second row was worked out by observing that $\boldsymbol{e}_{12}\boldsymbol{e}_1 = \boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_1 = -\boldsymbol{e}_2\boldsymbol{e}_1\boldsymbol{e}_1 = -\boldsymbol{e}_2$, that $\boldsymbol{e}_{12}\boldsymbol{e}_2 = \boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_2 = \boldsymbol{e}_1$, and by using the trigonometry relations $\sin(\theta) = 2\sin(\frac{\theta}{2})\cos(\frac{\theta}{2})$ and $\cos(\theta) = \cos(\frac{\theta}{2})^2 - \sin(\frac{\theta}{2})^2$.

For $\theta = \frac{\pi}{2}$, the above result, for example, gives the resulting vector $\boldsymbol{v}' = -v_2\boldsymbol{e}_1 + v_1\boldsymbol{e}_2 = \left[\begin{smallmatrix} -v_2 \\ v_1 \end{smallmatrix}\right]$. By replacing $\boldsymbol{e}_{12}$ with any other plane defined as a bivector, we can apply any other desired rotation in space. Interestingly, other geometric objects (not only vectors) can also be rotated with this operation. In the above equation, one might also recognize that the *sandwich* product $\boldsymbol{r}\boldsymbol{v}\tilde{\boldsymbol{r}}$ that we used has a similar corresponding operation in quaternion algebra.

4.2.2. Conformal geometric algebra (CGA). Translations can also be described by rotors in the conformal geometric algebra (CGA), a five-dimensional representation of the three-dimensional Euclidean space. To do so, the conformal model extends the real vector space $\mathbb{R}^3$ by adding two extra basis vectors, e and $\bar{e}$, with the properties $e^2 = 1$ and $\bar{e}^2 = -1$. This enlarged vector space is usually denoted by $\mathbb{R}^{4,1}$.

In addition, these two extra vectors allow us to define two null vectors $n_\infty = \bar{e} + e$ and $n_0 = \bar{e} - e$, where n_∞ is associated with the point at infinity and n_0 with the origin.

The conformal geometric algebra of $\mathbb{R}^3$ is denoted by $\mathcal{G}_{4,1}$ and can be seen as a geometric algebra of higher dimension (specifically, the geometric algebra of $\mathbb{R}^{4,1}$). In addition to the translation operations, other general geometric entities (points, lines, planes, circles, etc.) and the relations between them can also easily be described in CGA. The construction of these objects has an intuitive interpretation: They are built from a set of control points by simply using the outer product. For example, a circle is constructed with three points passing through this circle. If one moves one of these three points to infinity, the radius of this circle becomes infinite and the two other points effectively define a line. Similarly, a sphere is constructed with four points on the surface of this sphere, and we can move one of these points to infinity to define a plane (namely, a sphere of infinite radius). Operations such as intersections or interpolations using these objects hold a similar intuitive geometric reasoning.

CGA can also be extended to geometric algebra of higher dimensions. This could, for example, be used in robotics to model quadric surfaces such as cylinders or ellipsoids (58).

4.3. Practical usage and applications

GA has recently attracted major attention in the computer graphics community, and important efforts have been made to democratize and promote the field, such as the `https://bivector.net` website. This renewal of interest has been more timid in robotics, but a similar trend as in CG can easily be anticipated, since the benefit of GA in robotics builds on the same core foundations as in computer graphics. Indeed, geometric algebra can be used by a wide range of practitioners.

At a high-level geometric level, GA allows users to easily and intuitively define varied geometric shapes by defining a set of points in space, where four point describe the surface of a sphere, three points define a sphere, two points define a line, and so on. As illustrated in **Figure 4-*left***, functions can then be called or created, with the advantage that they will be valid for this whole set of objects. For example, writing a code to compute intersections will be the same if we want to compute the intersection between two spheres (giving a circle), the intersection between two planes (giving a line), or the intersection of a sphere

and a line (giving two points). Interestingly, the computation of intersections also handles the case in which the two objects do not intersect, with a continuous representation that can fluently switch between closest distances and intersections.

At an intermediate level, GA can be used to describe the position and orientation of a robot gripper in a homogeneous manner, as well as the evolution of the whole kinematic chain of a robot. The notion of forward and inverse kinematics can then be generalized to a set of objects (rather than just points), see **Figure 4-*right***.

At a more advanced level, GA can be extended to dynamics. The framework additionally offers many possible extensions for more expert users, including other variants of the GA model family, such as *projective geometric algebra* (PGA), as well as higher-dimensional versions of CGA to represent quadric surfaces and ellipses, or for use in multiarm scenarios (i.e., beyond bimanual robots).

As a starting point for testing and developing robotics applications based on CGA, an open source software library called *GAFRO*, which specifically targets robotics, is available at `https://gitlab.com/gafro`. It includes codes in C++ and Python, integrated into the ROS framework, together with an accompanying tutorial article (53).

The applications of GA in robotics are diverse, ranging from robot vision (59) to medical robotics (60), multicopters (61), and parallel robotics (62), see also (51) for a collection of applications.

For robot manipulators as serial kinematic chains, GA brings a geometric perspective to inverse kinematics (63). Moreover, it brings new perspectives to the modeling of cost functions in optimal control problems, allowing control and planning problems to be formulated uniformly across different geometric primitives, leading to a low symbolic complexity of the resulting expressions and a geometric intuitiveness (52). It can also be employed for dual arm manipulation to model various relative and absolute geometric constraints required for handling tasks involving two robot end effectors (46) (see **Figure 4b**).

GA also provides a robust way to measure distances to singularities (64). Indeed, the homogeneous treatment of position and orientation of a rigid body in space, provided by motors/rotors in GA, sheds new light on singularity identification in arbitrary serial robots (both redundant and non-redundant). This is achieved by modeling the twists defined by the joint axes of the robot, effectively avoiding the computation of Jacobian determinants, and providing distance measures to avoid singular configurations.

Dynamic extensions are also possible by extending the kinematic definition to measures of momenta and inertia, where the equations of motion for a rigid body can be derived by differentiating the momentum of the body (65). CGA can also be applied to the recursive forward dynamics computation of serial kinematic chains (66), providing a coordinate-free view of the algorithm and a geometrically meaningful interpretation of the involved quantities. This is achieved by extending the Lagrangian dynamics of a serial manipulator to include an inertia tensor, which can subsequently be used in inverse dynamics control schemes.

GA can also be used for force–motion control in contact with curved surface (24), allowing multiple task prioritization with no need to account for coordinate frames, which would be the case with traditional methods. Here, geometric algebra provides an intuitive way to represent a line corresponding to the surface orientation at a given position. The robot then tracks this 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 having conflicting

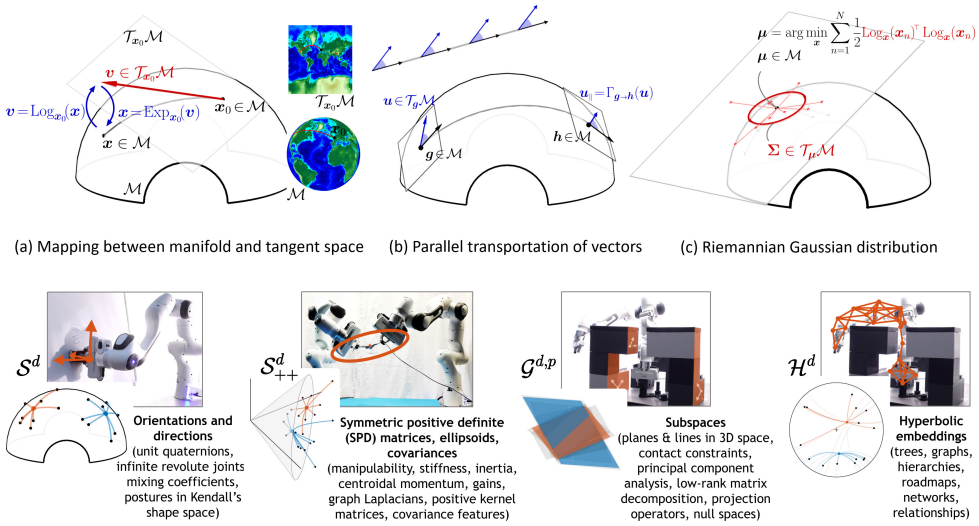


Figure 5: *Top:* Principles of Riemannian geometry relevant for applications in robotics, depicted here on a $\mathcal{S}^2$ manifold embedded in $\mathbb{R}^3$. *(a)* Bidirectional mappings between tangent space and manifold. *(b)* Parallel transport of a vector along a geodesic. The gray rectangles show the coordinate systems of several tangent spaces along the geodesic. The black vectors represent the directions of the geodesic in the tangent spaces. The blue vectors are transported from $\boldsymbol{g}$ to $\boldsymbol{h}$. *(c)* Computing the center and spread of data points on a manifold computed as a Riemannian Gaussian distribution. *Bottom:* Examples of Riemannian manifolds relevant for robotics, see (67) for details. The data points (black dots/planes) are segmented into two classes, each represented by a center (red and blue dots/planes). The geodesics depicted show the specificities of each manifold.

objectives to be handled.

5. RIEMANNIAN MANIFOLDS

Riemannian geometry is another relevant framework for robotics. It provides a principled and simple way to extend algorithms initially developed for Euclidean data to other curved spaces (manifolds), by exploiting their local resemblance to Euclidean spaces and by efficiently taking into account prior geometric knowledge about these manifolds (68).

As a first illustration of Riemannian manifolds, we can observe that the shortest trajectory of a flight connecting two cities in the world will be a curved path on a sphere. Because of the geometry of the sphere, when the two cities are distant, this curved geodesic path will not have the same length as the Euclidean distance joining the two cities with a straight line (i.e., with a path inside the earth). For shorter distances (e.g., distances between two houses in the same city), the Euclidean distance remains a convenient approximation. More rigorously, this approximation can be constructed in Riemannian geometry by defining a tangent space at one of the points, which is tangent to the manifold, and mapping the other point of the manifold in this tangent space, so that the distance between the two points (as a straight line) is the same as the length of the curved geodesic path (see **Figure 5a**).

This tangent space is Euclidean, which allows standard vector and distance operations to be considered. Similarly, the shortest route between two points on a mountain needs to take into account the curved landscape, resulting in possibly elaborate curved paths. If we

assume that the change of terrain is smooth enough, Riemannian geometry can also be used to formalize this route-finding problem as a geodesic path computation. In this case, the use of tangent spaces is even more localized since the curvature can change quickly, requiring a wise use of tangent spaces computed at different points on the manifold. In these two simple examples, the tangent space takes the form of a plane that touches the manifold at a given point, but tangent spaces can also be other geometric spaces when considering other types of manifolds.

The approach of locally representing the manifold as a tangent space can at first sight resemble the other common operation in optimization and optimal control, which consists of approximating a function with a Taylor series. Indeed, both methods establish representations that are simpler than the original function/manifold by approximating the shape of the original function/manifold at specific points (in a linear form for Riemannian manifolds, and as power series for Taylor expansions). They also both typically come with associated algorithms (e.g. cost minimization) that most often need the process to re-approximate the function at consecutively computed points. Despite this analogy, the key difference is that the tangent spaces in Riemannian geometry locally mimic the manifold's structure, by continuously connecting the different tangent spaces on the manifold. In other words, it "remembers" the manifold's local linearity. The geometric version thus comes with additional perspectives and geometric objects compared with the Taylor expansion that algebraically approximates the function using its derivatives.

There are also tight connections between Riemannian manifolds and Lie groups (69). While Riemannian manifolds target a wider diversity of geometries, Lie groups are inherently more structured because they combine both a manifold structure (smoothness) and a group structure (providing multiplication and inverse operations), which is typically exploited in robotics to handle 3D orientation and 6D full pose data encoding both position and orientation.

The most common use of Riemannian manifolds in robotics is for orientation tracking, which requires computing the distance between two orientations and executing the corresponding actions to reduce this distance. Encoding the current orientation and target orientation as unit quaternions allows the distance and action to be computed using a *logarithmic map* function that takes into account the shape of the manifold on which unit quaternions are located (70).

More generally, *Riemannian manifolds* can be connected to a wide range of problems in robotics, including learning and optimization problems that rely on distances and Gaussian distributions, which thus also requires uncertainty and statistical models to be extended to non-Euclidean data. This is relevant since data in robotics consist of values that are often constrained in a way that adheres to a particular geometry. Riemannian manifolds therefore allow data of various forms to be treated in a unified manner, which has the advantage that existing models and algorithms initially developed for Euclidean data can be extended to a wider range of data structures, by efficiently taking into account prior geometric knowledge about these manifolds. They can, for example, be used to revisit constrained optimization problems formulated in Euclidean space, treating such problems as unconstrained, while inherently taking into account the geometry of the data.

5.1. Mathematical overview

A d-dimensional Riemannian manifold $\mathcal{M}$ is a mathematical space, a smooth manifold, equipped with a Riemannian metric, which is used to measure distances, angles, and other geometric properties on the manifold. This manifold locally behaves like the Euclidean space $\mathbb{R}^d$, meaning that if we zoom in close enough to any point on this manifold, it would look like a flat standard Euclidean space. If we are close enough, we cannot distinguish between two manifolds, but if we zoom out, these manifolds can look very dissimilar and have very different properties.

For example, if we are close enough, a point on a curve or circle will look like a line, and the surface of an object or a sphere will look like a plane. In this latter example, the object is in the 3D space that we live in, and the plane when we zoom in can be described with 2D coordinates. Riemannian geometry can similarly consider objects that do not fit in our 3D physical space and are thus harder to imagine or visualize, but the computation steps are exactly the same. When developing algorithms using Riemannian manifolds, it can thus be useful in practice to first test the algorithms with data that in a lower-dimension space that can be easily visualized and analyzed within our 3D world, before extending the approach to objects of higher dimensions (for examples of low-dimensional representations of various manifolds, see **Figure 5-*bottom***).

For each point $\boldsymbol{p} \in \mathcal{M}$, there exists a tangent space $\mathcal{T}_{\boldsymbol{p}}\mathcal{M}$ that locally linearizes the manifold. On a Riemannian manifold, the metric tensor induces a positive definite inner product on each tangent space $\mathcal{T}_{\boldsymbol{p}}\mathcal{M}$, which allows vector lengths and angles between vectors to be measured. The affine connection, computed from the metric, is a differential operator that provides, among other functionalities, a way to compute geodesics and to transport vectors on tangent spaces along any smooth curves on the manifold. It also fully characterizes the intrinsic curvature and torsion of the manifold. The Cartesian product of two Riemannian manifolds is also a Riemannian manifold (often called a manifold bundle or manifold composite), which allows joint distributions to be constructed on any combination of Riemannian manifolds (e.g., to combine position and orientation data).

For applications in robotics, at least three principles of Riemannian geometry need to be considered, which are described next.

5.1.1. Geodesics. A geodesic is a curve that locally minimizes the distance between two points on a manifold. In a standard Euclidean space, one way to describe a straight line connecting two points (i.e., the shortest path between two points) is to think of it as a path whose first derivative is constant and second derivative is zero. Similarly, on a Riemannian manifold, the use of a geodesic equation to minimize the length or energy of a curve leads to the result that the second derivative of the curve is minimized along the geodesic. There is an intuitive analogy with the cartography problem of representing our spherical earth as a planar map, which induces spatial distortions whose strengths will depend on the choice of the central reference point (historically, Europe). On this planar map, the geodesic path of a flight route between two points will typically look curved (see **Figure 5a**).

5.1.2. Mapping functions between manifold and tangent spaces. The exponential map $\text{Exp}_{\boldsymbol{x}_0} : \mathcal{T}_{\boldsymbol{x}_0}\mathcal{M} \rightarrow \mathcal{M}$ maps a point $\boldsymbol{u}$ in the tangent space of $\boldsymbol{x}_0$ to a point $\boldsymbol{x}$ on the manifold, so that $\boldsymbol{x}$ lies on the geodesic starting at $\boldsymbol{x}_0$ in the direction $\boldsymbol{u}$. The norm of $\boldsymbol{u}$ is equal to the geodesic distance between $\boldsymbol{x}_0$ and $\boldsymbol{x}$. The inverse map is called the logarithmic map $\text{Log}_{\boldsymbol{x}_0} : \mathcal{M} \rightarrow \mathcal{T}_{\boldsymbol{x}_0}\mathcal{M}$. **Figure 5a** depicts these mapping functions.

5.1.3. Parallel transport. Parallel transport $\Gamma_{\boldsymbol{g}\to\boldsymbol{h}} : \mathcal{T}_{\boldsymbol{g}}\mathcal{M} \to \mathcal{T}_{\boldsymbol{h}}\mathcal{M}$ moves vectors between tangent spaces such that the inner product between two vectors in a tangent space is conserved. It employs the notion of connection, defining how to associate vectors between infinitesimally close tangent spaces. This connection allows the smooth transport of a vector from one tangent space to another by sliding it (with infinitesimal moves) along a curve. **Figure 5b** illustrates this operation. Parallel transport allows a vector $\boldsymbol{u}$ in the tangent space of $\boldsymbol{g}$ to be transported to the tangent space of $\boldsymbol{h}$, by ensuring that the angle (i.e., inner product) between $\boldsymbol{u}$ and the direction of the geodesic (represented as black vectors) are conserved. At point $\boldsymbol{g}$, this direction is expressed as $\mathrm{Log}_{\boldsymbol{g}}(\boldsymbol{h})$. This operation is crucial to combine information available at $\boldsymbol{g}$ with information available at $\boldsymbol{h}$, by taking into account the rotation of the coordinate systems along the geodesic (see rectangles in gray). In Euclidean space (top-left inset), the parallel transport is simply the identity operator (a vector operation can be applied to any point without additional transformation). By extension, a covariance matrix $\boldsymbol{\Sigma}$ can be transported with $\boldsymbol{\Sigma}_{\parallel} = \sum_{i=1}^{d} \Gamma_{\boldsymbol{g}\to\boldsymbol{h}}(\boldsymbol{v}_i)\, \Gamma_{\boldsymbol{g}\to\boldsymbol{h}}^{\top}(\boldsymbol{v}_i)$, using the eigendecomposition $\boldsymbol{\Sigma} = \sum_{i=1}^{d} \boldsymbol{v}_i \boldsymbol{v}_i^{\top}$. For several manifolds in robotics, this transport operation can equivalently be expressed as a linear mapping $\boldsymbol{\Sigma}_{\parallel} = \boldsymbol{A}_{\boldsymbol{g}\to\boldsymbol{h}} \boldsymbol{\Sigma}\, \boldsymbol{A}_{\boldsymbol{g}\to\boldsymbol{h}}^{\top}$.

5.2. Homogeneous manifolds

The presence of structured geometric data in robotics has motivated researchers to explore which aspects of robotics may be represented by Riemannian manifolds. The most common manifolds in robotics are homogeneous, where the geometry is uniform throughout the space, with the same curvature at any point on the manifold (i.e., looking locally the same at any point on the manifold), which provides simple analytic expressions for exponential/logarithmic mapping and parallel transport (for a review, see (67)).

Figure 5-*bottom* shows four examples of homogeneous Riemannian manifolds that can be useful in robotics, represented here as $\mathcal{S}^2$, $\mathcal{S}^2_{++}$, $\mathcal{G}^{3,2}$ and $\mathcal{H}^2$ manifolds. The sphere manifold $\mathcal{S}^d$ is characterized by constant positive curvature. The elements of $\mathcal{S}^d_{++}$ can be represented as the interior of a convex cone embedded in its tangent space Sym^d. Here, the three axes shown in the figure correspond to A_{11}, A_{12} and A_{22} in the symmetric positive definite (SPD) matrix $\left(\begin{smallmatrix} A_{11} & A_{12} \\ A_{12} & A_{22} \end{smallmatrix}\right)$, where the points inside the cone are on the manifold, with tangent spaces covering the entire space (inside and outside the code). $\mathcal{G}^{d,p}$ is the Grassmann manifold of all p-dimensional subspaces of $\mathbb{R}^d$. The hyperbolic manifold $\mathcal{H}^d$ is characterized by constant negative curvature. Several representations exist: $\mathcal{H}^2$ can, for example, be represented as the interior of a unit disk in Euclidean space, with the boundary of the disk representing infinitely remote point (a Poincaré disk model, as depicted here). In this model, geodesic paths are arcs of circles intersecting the boundary perpendicularly.

5.2.1. Sphere manifolds $\mathcal{S}^d$. The sphere manifold $\mathcal{S}^d$ is the most widely used non-Euclidean manifold in robotics, as it can encode direction and orientation information (71). Unit quaternions $\mathcal{S}^3$ can be used to represent the orientations of end effectors (tool center points). $\mathcal{S}^2$ can be used to represent a unit directional vector perpendicular to surfaces (e.g., for contact planning). Infinite revolute joints can be represented on the torus $\mathcal{S}^1 \times \mathcal{S}^1 \times \ldots \times \mathcal{S}^1$.

The exponential and logarithmic maps corresponding to the distance

$$d(\boldsymbol{x}, \boldsymbol{y}) = \arccos(\boldsymbol{x}^{\top}\boldsymbol{y}),$$

with $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{S}^d$ are computed as

$$\boldsymbol{y} = \mathrm{Exp}_{\boldsymbol{x}}(\boldsymbol{u}) = \boldsymbol{x}\cos(\|\boldsymbol{u}\|) + \frac{\boldsymbol{u}}{\|\boldsymbol{u}\|}\sin(\|\boldsymbol{u}\|),$$
$$\boldsymbol{u} = \mathrm{Log}_{\boldsymbol{x}}(\boldsymbol{y}) = d(\boldsymbol{x},\boldsymbol{y})\,\frac{\boldsymbol{y} - \boldsymbol{x}^\top\boldsymbol{y}\,\boldsymbol{x}}{\|\boldsymbol{y} - \boldsymbol{x}^\top\boldsymbol{y}\,\boldsymbol{x}\|}.$$

The parallel transport of $\boldsymbol{v} \in \mathcal{T}_{\boldsymbol{x}}\mathcal{S}^d$ to $\mathcal{T}_{\boldsymbol{y}}\mathcal{S}^d$ is given by

$$\Gamma_{\boldsymbol{x}\to\boldsymbol{y}}(\boldsymbol{v}) = \boldsymbol{v} - \frac{\mathrm{Log}_{\boldsymbol{x}}(\boldsymbol{y})^\top\boldsymbol{v}}{d(\boldsymbol{x},\boldsymbol{y})^2}\Big(\mathrm{Log}_{\boldsymbol{x}}(\boldsymbol{y}) + \mathrm{Log}_{\boldsymbol{y}}(\boldsymbol{x})\Big).$$

In some applications, it can be convenient to define the parallel transport with the alternative equivalent form

$$\Gamma_{\boldsymbol{x}\to\boldsymbol{y}}(\boldsymbol{v}) = \boldsymbol{A}_{\boldsymbol{x}\to\boldsymbol{y}}\,\boldsymbol{v}, \quad \text{with}$$
$$\boldsymbol{A}_{\boldsymbol{x}\to\boldsymbol{y}} = -\boldsymbol{x}\sin(\|\boldsymbol{u}\|)\overline{\boldsymbol{u}}^\top + \overline{\boldsymbol{u}}\cos(\|\boldsymbol{u}\|)\overline{\boldsymbol{u}}^\top + (\boldsymbol{I} - \overline{\boldsymbol{u}}\,\overline{\boldsymbol{u}}^\top), \quad \boldsymbol{u} = \mathrm{Log}_{\boldsymbol{x}}(\boldsymbol{y}), \quad \text{and} \quad \overline{\boldsymbol{u}} = \frac{\boldsymbol{u}}{\|\boldsymbol{u}\|},$$

highlighting the linear structure of the operation.

Note that in the above representation, $\boldsymbol{u}$ and $\boldsymbol{v}$ are described as vectors with $d+1$ elements contained in $\mathcal{T}_{\boldsymbol{x}}\mathcal{S}^d$. An alternative representation consists of expressing $\boldsymbol{u}$ and $\boldsymbol{v}$ as vectors of d elements in the coordinate system attached to $\mathcal{T}_{\boldsymbol{x}}\mathcal{S}^d$.

5.2.2. Ellipsoids and symmetric positive definite matrices as $\mathcal{S}^d_{++}$ manifolds. Symmetric positive definite (SPD) matrices are also widely used in robotics. They can represent stiffness, manipulability, inertia ellipsoids, or weighting matrices. They can also be used to process sensory data in the form of spatial covariances. For example, the robot's manipulability capability in translation and rotation can be encoded as a manipulability ellipsoid $\mathcal{S}^6_{++}$ (72). Manipulability ellipsoids can be exploited as descriptors to transfer movement and manipulation skills between humans and robots, as well as between robot platforms characterized by different kinematic chains (73).

For an affine-invariant distance between $\boldsymbol{X}, \boldsymbol{Y} \in \mathcal{S}^d_{++}$

$$d(\boldsymbol{X},\boldsymbol{Y}) = \big\|\log(\boldsymbol{X}^{-\frac{1}{2}}\boldsymbol{Y}\boldsymbol{X}^{-\frac{1}{2}})\big\|_{\mathrm{F}},$$

the exponential and logarithmic maps on the SPD manifold can be computed as

$$\boldsymbol{Y} = \mathrm{Exp}_{\boldsymbol{X}}(\boldsymbol{U}) = \boldsymbol{X}^{\frac{1}{2}}\exp\big(\boldsymbol{X}^{-\frac{1}{2}}\boldsymbol{U}\boldsymbol{X}^{-\frac{1}{2}}\big)\boldsymbol{X}^{\frac{1}{2}},$$
$$\boldsymbol{U} = \mathrm{Log}_{\boldsymbol{X}}(\boldsymbol{Y}) = \boldsymbol{X}^{\frac{1}{2}}\log\big(\boldsymbol{X}^{-\frac{1}{2}}\boldsymbol{Y}\boldsymbol{X}^{-\frac{1}{2}}\big)\boldsymbol{X}^{\frac{1}{2}}.$$

The parallel transport of $\boldsymbol{V} \in \mathcal{T}_{\boldsymbol{X}}\mathcal{S}^d_{++}$ to $\mathcal{T}_{\boldsymbol{Y}}\mathcal{S}^d_{++}$ is given by

$$\Gamma_{\boldsymbol{X}\to\boldsymbol{Y}}(\boldsymbol{V}) = \boldsymbol{A}_{\boldsymbol{X}\to\boldsymbol{Y}}\,\boldsymbol{V}\,\boldsymbol{A}_{\boldsymbol{X}\to\boldsymbol{Y}}^\top, \quad \text{with} \quad \boldsymbol{A}_{\boldsymbol{X}\to\boldsymbol{Y}} = \big(\boldsymbol{Y}\boldsymbol{X}^{-1}\big)^{\frac{1}{2}}.$$

5.2.3. Riemannian Gaussian distribution. Several approaches can be used to extend Gaussian distributions to Riemannian manifolds. The simplest consists of estimating the mean of the Gaussian as a centroid on the manifold (also called the Karcher/Fréchet mean), and representing the dispersion of the data as a covariance matrix evaluated in the tangent space of the mean (67, 74) (see **Figure 5c**). This is an approximation, because distortions arise

when points are too far apart from the center. However, this distortion is negligible in most robotics applications. In particular, this effect is strongly attenuated when the Gaussian covers a small part of the manifold.

The Riemannian Gaussian representation on a manifold $\mathcal{M}$ is defined as

$$\mathcal{N}_{\mathcal{M}}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \left((2\pi)^d|\boldsymbol{\Sigma}|\right)^{-\frac{1}{2}} e^{-\frac{1}{2}\operatorname{Log}_{\boldsymbol{\mu}}(\boldsymbol{x})^\top \boldsymbol{\Sigma}^{-1} \operatorname{Log}_{\boldsymbol{\mu}}(\boldsymbol{x})},$$

where $\boldsymbol{x} \in \mathcal{M}$ is a point of the manifold, $\boldsymbol{\mu} \in \mathcal{M}$ is the mean of the distribution (origin of the tangent space), and $\boldsymbol{\Sigma} \in \mathcal{T}_{\boldsymbol{\mu}}\mathcal{M}$ is the covariance defined in this tangent space.

For a set of N data points, this geometric mean corresponds to the minimization

$$\min_{\boldsymbol{\mu}} \sum_{n=1}^{N} \operatorname{Log}_{\boldsymbol{\mu}}(\boldsymbol{x}_n)^\top \operatorname{Log}_{\boldsymbol{\mu}}(\boldsymbol{x}_n),$$

which can be solved by a simple and fast Gauss–Newton iterative algorithm. The algorithm starts from an initial estimate on the manifold and an associated tangent space. The data points $\{\boldsymbol{x}_n\}_{n=1}^N$ are projected in this tangent space to compute a direction vector, which provides an updated estimate of the mean. This process is repeated by iterating

$$\boldsymbol{u} = \frac{1}{N}\sum_{n=1}^{N} \operatorname{Log}_{\boldsymbol{\mu}}(\boldsymbol{x}_n), \qquad \boldsymbol{\mu} \leftarrow \operatorname{Exp}_{\boldsymbol{\mu}}(\boldsymbol{u}),$$

until convergence. In practice, for homogeneous manifolds with constant curvatures, this iterative algorithm will converge very quickly, with only a couple of iterations.

After convergence, a covariance matrix is computed in the tangent space as $\boldsymbol{\Sigma} = \frac{1}{N-1}\sum_{n=1}^{N} \operatorname{Log}_{\boldsymbol{\mu}}(\boldsymbol{x}_n) \operatorname{Log}_{\boldsymbol{\mu}}(\boldsymbol{x}_n)^\top$, see **Figure 5c**.

This distribution can, for example, be used in a control problem to represent a reference to track with an associated required precision learned from a set of demonstrations. Importantly, this geometric mean can be directly extended to weighted distances, which is useful for mixture modeling, fusion (product of Gaussians), regression (Gaussian conditioning) and planning problems (67, 74).

Gaussians on Riemannian manifolds can also be exploited for automatic selection of one or more manifolds by analyzing the variations in a list of candidate manifolds, which can be used in robotics to extract which manifold(s) can best describe the skill or motion based on invariant characteristics (75, 76). More generally, the combination of statistics and Riemannian geometry in learning, control, and optimization applications offers many possible research extensions. A first observation if that in many robotics challenges, a common factor is the need to handle (co)variations, statistics, and/or propagation of uncertainty.

In practice, algorithms that were initially developed for standard Euclidean data can be easily transformed to extend their application to other Riemannian manifolds. Moreover, it is straightforward to consider multiple manifolds simultaneously, with one part of the data on one manifold and the other part on another manifold. This is interesting for robotics because the approach can model heterogeneous data in a unified manner, while jointly considering co-variations and uncertainty through the use of concatenated tangent spaces. This capability has several practical research applications, as it is common in robotics to juggle several types of sensors and actuators, and Riemannian geometry offers a simple and formal way to homogeneously handle heterogeneous sources of data.

5.2.4. Other homogeneous manifolds. In addition, several types of homogeneous Riemannian manifolds remain largely underexploited in robotics, even though most of them are mathematically well-understood and characterized by simple analytic expressions. For example, Grassmann and Stiefel manifolds seem particularly promising to handle problems in robotics where projections and subspaces are required in the computation, which can, for example, be employed to keep the most essential characteristics of datasets (low rank factorization).

In this context, one option is to treat the data as points on a Grassmann manifold and to compute weighted averages on this manifold, effectively providing a dictionary of basis functions (77). This approach consists of computing the average subspace spanned by the data, that is, the first moment on the Grassmann manifold. While standard principal component analysis (PCA) is concerned with the second moment of the data (covariance), the geometric averaging approach simplifies subspace estimation. As demonstrated by (77), this formulation of PCA simplifies the development of robust extensions. Indeed, the concept of studying the subspaces spanned by the data (rather than the data itself) opens several new directions that can be explored in robotics. As an extension to the computation of weighted averages, one example concerns the clustering of points on manifolds, as proposed in (78) in the context of muscle synergies.

Grassmann and Stiefel manifolds are also promising for problems that require considering tasks prioritization (e.g., inverse kinematics with kinematically redundant robots), because the manifold can provide a geometric interpretation and treatment of nullspace structures.

Other Riemannian manifolds, such as hyperbolic manifolds, also seem well-suited to bring a probabilistic treatment to tree-based structures, graphs, and Toeplitz/Hankel matrices (79). Hyperbolic manifolds can also be used for data structures forming hierarchies or taxonomies (80). They form an extension of tree-like structures to continuous domains, depicted in **Figure 5** (bottom-right) as points on a disc (2D embedding), where points near the center correspond to nodes near the trunk of the tree and points approaching the border of the disc correspond to the leaf nodes of the tree.

5.3. Inhomogeneous manifolds

More general forms of manifolds with nonconstant curvature can also be employed. In particular, this includes spaces endowed with a metric, characterized by a weighting matrix used to compute distances. Many problems in robotics can be formulated with such a smoothly varying matrix $\boldsymbol{G}(\boldsymbol{x})$ that can, for example, be used to evaluate the weighted norm of a command $\boldsymbol{u}$ applied at a location $\boldsymbol{x}$ as a quadratic error term $c(\boldsymbol{u}) = \boldsymbol{u}^\top \boldsymbol{G}(\boldsymbol{x})\boldsymbol{u}$, forming a Riemannian metric that describes the underlying manifold (with inhomogeneous curvature).

A typical example of a smoothly varying metric in robotics is when the weight matrix is a mass inertia matrix so that inertia is taken into account in order to generate a movement from a starting joint configuration to a target joint configuration while minimizing kinetic energy (81, 82, 83, 84, 85, 86) (see **Figure 6-*left***). This technique can similarly be used to model uncertainty and variations in the data (87), and for optimization problems such as control policies accounting for a Riemannian metric (88, 89). More generally, the consideration of a metric with a smoothly varying weighting matrix also provides a modular framework to bridge multiple scientific disciplines with robotics, including motor control

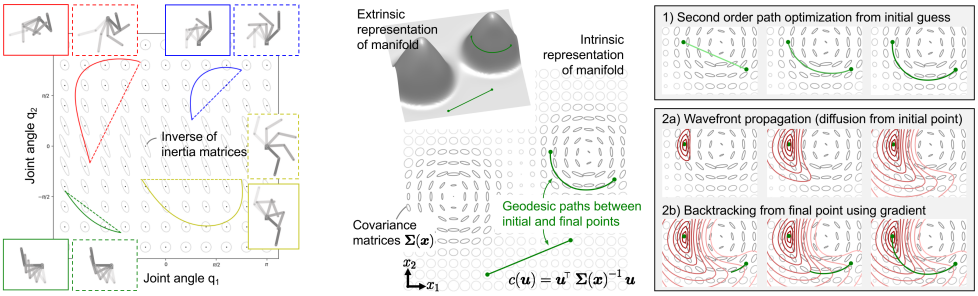


Figure 6: *Left:* Computing geodesic paths in the joint angle space of a two-link planar robot to account for inertia in the movement. Inverse weight matrices are depicted as ellipsoids to locally visualize the equidistant points (with respect to the metric) that the robot can reach. The geodesics are shown as solid lines, and the baseline movements that would be produced by ignoring inertia are shown as dashed lines, corresponding to linear interpolation between the two joint angle poses. *Center:* Example with a user-defined metric in task space based on a covariance matrix to generate paths that naturally curve around obstacles when the obstacles are close and are otherwise straight (see the two point-to-point trajectories in green, either far from or close to the obstacle). *Right:* Computation of geodesics.

and human motion science (90).

Figure 6-*center* shows two representations of the same geometric object, where the description of a manifold through its metric corresponds to an intrinsic representation. Indeed, differential geometry can either be intrinsic, meaning that the spaces it considers are smooth manifolds whose geometric structure is governed by a Riemannian metric that determines how distances are measured near each point, or extrinsic, where the studied manifold is part of some ambient flat Euclidean space. The inset graph shows an example of an extrinsic representation corresponding to the metric depicted as an intrinsic representation. This example was crafted so that the corresponding extrinsic representation can be represented in a 3D space, but this is not always possible; in other cases, it can require more dimensions, and the extrinsic representations can be difficult to estimate. Fortunately, Riemannian geometry does not require an explicit representation—only the intrinsic representation is required in the computation.

Since the use of a metric is tightly connected to control, planning, and optimization problems, we can also refer to the use of the inhomogeneous manifolds described above as *Riemannian optimization*, which brings a geometric interpretation to cost minimization problems. In this context, it is not the data points that are on the manifold; instead, the metric forms the manifold. Often, we need to consider Riemannian manifolds for both the data and the metric (e.g., in optimization problems with orientation data and weight matrices).

5.3.1. Computation of geodesics for inhomogeneous manifolds. In contrast to homogeneous manifolds, for which geodesics can be computed in a very fast and straightforward manner, computation can be harder for manifolds with nonconstant curvature. **Figure 6-*right*** shows two examples of methods to compute these geodesics. The first is an example of iterative methods to solve the underlying differential equation problem, by starting from an initial guess and updating the guess until convergence (second-order optimization formulated here as a Gauss–Newton path optimization problem). One disadvantage of this

approach is that a good initial guess is required.

The second example first builds a distance field from one of the points (2a), and then use the gradient of the constructed distance field to backtrack a path from the second point (2b). The first step of this process can, for example, be implemented with a Riemannian fast marching algorithm (91). One disadvantage of this approach is that the process becomes very expensive for higher dimensions.

To address this inconvenience, another line of research starts from the observation that distance fields have an eikonal equation property, as seen in Section 3 in the context of distance fields. In the case of distances governed by a Riemannian metric defined by $\boldsymbol{G}$, a similar weighted variant of the eikonal equation $\|\boldsymbol{\nabla} f\|_{\boldsymbol{G}} = 1$ can be used. Similarly to the approach used with distance fields, this property can be used to estimate the distance function with a function approximator, which is the core principle of physics-informed neural networks (92, 93). The same process can be applied in robotics applications to compute geodesics in a very fast and modular manner (86).

6. CONCLUSION

This article has presented an overview of geometric approaches and frameworks to facilitate the acquisition and transfer of skills in robotics, together with promising extensions. Adopting geometric representations can simplify learning and transfer problems, while providing a theoretically sound treatment of such structures in robotics. In essence, the targeted goal is to leverage prior knowledge on the geometric structures of the problem to design learning, planning and control algorithms that can be simpler, more generic and/or more robust, without having to change the algorithm for each new geometric problem.

ACKNOWLEDGMENTS

This work was supported by the Swiss National Science Foundation (SNSF) through the HORACE project (`https://horace-robotics.github.io`), and by the Swiss State Secretariat for Education, Research and Innovation (SERI) 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).

LITERATURE CITED

1. TRI LBM Team, Barreiros J, Beaulieu A, Bhat A, Cory R, et al. 2025. A careful examination of large behavior models for multitask dexterous manipulation. *arXiv:2507.05331*
2. Khazatsky A, Pertsch K, Nair S, Balakrishna A, Dasari S, et al. 2024. DROID: A large-scale in-the-wild robot manipulation dataset. *arXiv:2403.12945*
3. Zitkovich B, Yu T, Xu S, Xu P, Xiao T, et al. 2023. *RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control.* In *Proc. Conf. on Robot Learning*, pp. 2165–2183
4. Chatzilygeroudis K, Vassiliades V, Stulp F, Calinon S, Mouret JB. 2020. A survey on policy search algorithms for learning robot controllers in a handful of trials. *IEEE Trans. on Robotics* 32(2):328–347
5. Hornung A, Wurm KM, Bennewitz M, Stachniss C, Burgard W. 2013. OctoMap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots* 34:189–206

6. Büttner S, Marton ZC, Hertkorn K. 2015. Automatic scene parsing for generic object descriptions using shape primitives. *Robotics and Autonomous Systems* 76:93–112
7. Rosenfeld A, Pfaltz JL. 1968. Distance functions on digital pictures. *Pattern Recognition* 1(1):33–61
8. Hart JC. 1996. Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer* 12(10):527–545
9. Conte G, Longhi S, Zulli R. 1995. Non-holonomic motion planning using distance field. In *Intelligent Autonomous Vehicles 1995*, ed. A Halme, K Koskinen, pp. 91–96. Oxford: Pergamon
10. Dyck M, Sachtler A, Klodmann J, Albu-Schäffer A. 2022. Impedance control on arbitrary surfaces for ultrasound scanning using discrete differential geometry. *IEEE Robotics and Automation Letters* 7(3):7738–7746
11. Ratliff N, Zucker M, Bagnell JA, Srinivasa S. 2009. *CHOMP: Gradient optimization techniques for efficient motion planning*. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pp. 489–494
12. Oleynikova H, Taylor Z, Fehr M, Siegwart R, Nieto J. 2017. *Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning*. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pp. 1366–1373
13. Koptev M, Figueroa N, Billard A. 2022. Neural joint space implicit signed distance functions for reactive robot manipulator control. *IEEE Robotics and Automation Letters (RA-L)* 8(2):480–487
14. Liu P, Zhang K, Tateo D, Jauhri S, Peters J, Chalvatzaki G. 2022. *Regularized deep signed distance fields for reactive motion generation*. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pp. 6673–6680
15. Driess D, Ha JS, Toussaint M, Tedrake R. 2022. *Learning models as functionals of signed-distance fields for manipulation planning*. In *Proc. Conference on Robot Learning (CoRL)*, pp. 245–255
16. Weng T, Held D, Meier F, Mukadam M. 2023. *Neural Grasp Distance Fields for Robot Manipulation*. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pp. 1814–1821
17. Wu L, Lee KMB, Le Gentil C, Vidal-Calleja T. 2023. Log-GPIS-MOP: A unified representation for mapping, odometry, and planning. *IEEE Transactions on Robotics* 39(5):4078–4094
18. Vasu S, Talabot N, Lukoianov A, Baqué P, Donier J, Fua P. 2022. *HybridSDF: Combining Deep Implicit Shapes and Geometric Primitives for 3D Shape Representation and Manipulation*. In *Intl Conf. on 3D Vision*
19. Gropp A, Yariv L, Haim N, Atzmon M, Lipman Y. 2020. *Implicit Geometric Regularization for Learning Shapes*. In *Proc. Intl Conf. on Machine Learning (ICML)*, pp. 3789–3799
20. Marić A, Li Y, Calinon S. 2024. Online learning of continuous signed distance fields using piecewise polynomials. *IEEE Robotics and Automation Letters (RA-L)* 9(6):6020–6026
21. Li Y, Zhang Y, Razmjoo A, Calinon S. 2024. *Representing Robot Geometry as Distance Fields: Applications to Whole-body Manipulation*. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pp. 15351–15357
22. Crane K, Weischedel C, Wardetzky M. 2017. The heat method for distance computation. *Commun. ACM* 60(11):90–99
23. Bilaloglu C, Löw T, Calinon S. 2023. Whole-body ergodic exploration with a manipulator using diffusion. *IEEE Robotics and Automation Letters (RA-L)* 8(12):8581–8587
24. Bilaloglu C, Löw T, Calinon S. 2025. Tactile ergodic coverage on curved surfaces. *IEEE Transactions on Robotics (T-RO)* 41:1421–1435
25. Nießner M, Zollhöfer M, Izadi S, Stamminger M. 2013. Real-time 3D reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (ToG)* 32(6):1–11
26. Brudermüller L, Lembono TS, Shetty S, Calinon S. 2021. *Trajectory Prediction with Compressed 3D Environment Representation using Tensor Train Decomposition*. In *Proc. IEEE Intl Conf. on Advanced Robotics (ICAR)*, pp. 633–639

27. Pujol E, Chica A. 2023. Adaptive approximation of signed distance fields through piecewise continuous interpolation. *Computers & Graphics* 114:337–346
28. Williams O, Fitzgibbon A. 2007. *Gaussian Process Implicit Surfaces*. In *Gaussian Processes in Practice*
29. Mahler J, Patil S, Kehoe B, van den Berg J, Ciocarlie M, et al. 2015. *GP-GPIS-OPT: Grasp planning with shape uncertainty using Gaussian process implicit surfaces and Sequential Convex Programming*. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pp. 4919–4926
30. Caccamo S, Bekiroglu Y, Ek CH, Kragic D. 2016. *Active exploration using Gaussian Random Fields and Gaussian Process Implicit Surfaces*. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pp. 582–589
31. Dragiev S, Toussaint M, Gienger M. 2011. *Gaussian process implicit surfaces for shape estimation and grasping*. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pp. 2845–2850
32. Li M, Hang K, Kragic D, Billard A. 2016. Dexterous grasping under shape uncertainty. *Robotics and Autonomous Systems* 75:352–364
33. Martens W, Poffet Y, Soria PR, Fitch R, Sukkarieh S. 2017. Geometric priors for Gaussian process implicit surfaces. *IEEE Robotics and Automation Letters* 2(2):373–380
34. Amanhoud W, Khoramshahi M, Billard A. 2019. *A dynamical system approach to motion and force generation in contact tasks*. In *Proc. Robotics: Science and Systems (RSS)*
35. Joukov V, Kulić D. 2022. Fast approximate multi-output gaussian processes. *IEEE Intelligent Systems* 37(4):56–69
36. Park JJ, Florence P, Straub J, Newcombe R, Lovegrove S. 2019. *DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation*. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 165–174
37. Li Y, Chi X, Razmjoo A, Calinon S. 2024. *Configuration Space Distance Fields for Manipulation Planning*. In *Proc. Robotics: Science and Systems (RSS)*
38. Bilaloglu C, Löw T, Calinon S. 2025. Diffusion-based virtual fixtures. *arXiv:2411.02169*
39. Guillard B, Stella F, Fua P. 2022. *MeshUDF: Fast and Differentiable Meshing of Unsigned Distance Field Networks*. In *Proc. European Conference on Computer Vision (ECCV)*, pp. 576–592
40. Li Y, Calinon S. 2025. From movement primitives to distance fields to dynamical systems. *IEEE Robotics and Automation Letters (RA-L)* 10(9):9550–9556
41. Yang Y, Ivan V, Vijayakumar S. 2015. *Real-time motion adaptation using relative distance space representation*. In *Proc. Intl Conf. on Advanced Robotics (ICAR)*, pp. 21–27
42. Marić F, Giamou M, Hall AW, Khoubyarian S, Petrović I, Kelly J. 2022. Riemannian Optimization for Distance-Geometric Inverse Kinematics. *IEEE Trans. on Robotics* 38(3):1703–1722
43. Mildenhall B, Srinivasan PP, Tancik M, Barron JT, Ramamoorthi R, Ng R. 2021. NeRF: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM* 65(1):99–106
44. Kerbl B, Kopanas G, Leimkuehler T, Drettakis G. 2023. 3D Gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.* 42(4)
45. Li Y, Li S, Sitzmann V, Agrawal P, Torralba A. 2022. *3D neural scene representations for visuomotor control*. In *Conference on Robot Learning*, pp. 112–123
46. Löw T, Calinon S. 2024. *Extending the Cooperative Dual-Task Space in Conformal Geometric Algebra*. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pp. 14882–14887
47. Lasenby J, Lasenby AN, Doran CJL. 2000. A unified mathematical language for physics and engineering in the 21st century. *Philos. Trans. R. Soc. A* 358(1765):35821–39
48. Hestenes D. 1966. *Space-Time Algebra*. New York: Gordon and Breach
49. Hestenes D, Sobczyk G, Marsh JS. 1985. Clifford algebra to geometric calculus: A unified language for mathematics and physics. *American Journal of Physics* 53(5):510–511

50. Hitzer E, Lavor C, Hildenbrand D. 2022. Current survey of Clifford geometric algebra applications. *Mathematical Methods in the Applied Sciences* 47(3):1331–1361
51. Bayro-Corrochano E. 2020. *Geometric Algebra Applications Vol. II: Robot Modelling and Control*. Springer International Publishing
52. Löw T, Calinon S. 2023. Geometric algebra for optimal control with applications in manipulation tasks. *IEEE Transactions on Robotics (T-RO)* 39(5):3586–3600
53. Löw T, Abbet P, Calinon S. 2025. GAFRO: Geometric algebra for robotics. *IEEE Robotics and Automation Magazine* 32(3):184–194
54. Marinho MM, Figueredo LFC, Adorno BV. 2015. *A Dual Quaternion Linear-Quadratic Optimal Controller for Trajectory Tracking*. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pp. 4047–4052
55. Fonseca MdPA, Adorno BV, Fraisse P. 2020. Coupled task-space admittance controller using dual quaternion logarithmic mapping. *IEEE Robotics and Automation Letters* 5(4):6057–6064
56. Adorno BV, Marinho MM. 2021. DQ Robotics: A library for robot modeling and control. *IEEE Robotics Automation Magazine* 28(3):102–116
57. Parameswaran R, Panakkal SM, Vedan MJ. 2024. Fluid Maxwell's equations in the language of geometric algebra. *Pramana - Journal of Physics* 98(62)
58. Breuils S, Nozick V, Sugimoto A, Hitzer E. 2018. Quadric conformal geometric algebra of $\mathbb{R}^{9,6}$. *Advances in Applied Clifford Algebras* 28(35)
59. Sommer G. 2005. *Applications of Geometric Algebra in Robot Vision*. In *Computer Algebra and Geometric Algebra with Applications*, pp. 258–277
60. Bayro-Corrochano E, Garza-Burgos AM, Del-Valle-Padilla JL. 2020. Geometric intuitive techniques for human machine interaction in medical robotics. *Intl Journal of Social Robotics* 12:91–112
61. Arellano-Muro CA, Osuna-González G, Castillo-Toledo B, Bayro-Corrochano E. 2020. Newton–Euler modeling and control of a multi-copter using motor algebra $\mathbf{G}^{+}_{3,0,1}$. *Advances in Applied Clifford Algebras* 30(19)
62. Hadfield H, Wei L, Lasenby J. 2020. *The Forward and Inverse Kinematics of a Delta Robot*. In *Advances in Computer Graphics*, pp. 447–458
63. Zaplana I, Hadfield H, Lasenby J. 2022. Closed-form solutions for the inverse kinematics of serial robots using conformal geometric algebra. *Mechanism and Machine Theory* 173:104835
64. Zaplana I, Hadfield H, Lasenby J. 2022. Singularities of serial robots: Identification and distance computation using geometric algebra. *Mathematics* 10(12)
65. Selig JM, Bayro-Corrochano E. 2010. Rigid body dynamics using Clifford algebra. *Advances in Applied Clifford Algebras* 20:141–154
66. Löw T, Calinon S. 2024. *Recursive Forward Dynamics for Serial Kinematic Chains using Conformal Geometric Algebra*. In *Proc. Intl Workshop on the Algorithmic Foundations of Robotics (WAFR)*
67. Calinon S. 2020. Gaussians on Riemannian manifolds: Applications for robot learning and adaptive control. *IEEE Robotics and Automation Magazine (RAM)* 27(2):33–45
68. Lee JM. 2018. *Introduction to Riemannian Manifolds*. Springer
69. Solà J, Deray J, Atchuthan D. 2021. A micro Lie theory for state estimation in robotics. *arXiv:1812.01537v9*
70. Grassia FS. 1998. Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools* 3(3):29–48
71. Zeestraten MJA, Havoutis I, Silvério J, Calinon S, Caldwell DG. 2017. An approach for imitation learning on Riemannian manifolds. *IEEE Robotics and Automation Letters (RA-L)* 2(3):1240–1247
72. Jaquier N, Rozo L, Caldwell DG, Calinon S. 2021. Geometry-aware manipulability learning, tracking and transfer. *International Journal of Robotics Research (IJRR)* 40(2–3):624–650
73. Jaquier N, Rozo L, Calinon S. 2020. *Analysis and Transfer of Human Movement Manipulability*

in Industry-like Activities. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pp. 11131–11138
74. Saveriano M, Abu-Dakka FJ, Kyrki V. 2023. Learning stable robotic skills on riemannian manifolds. *Robotics and Autonomous Systems* 169:104510
75. Ti B, Razmjoo A, Gao Y, Zhao J, Calinon S. 2023. A geometric optimal control approach for imitation and generalization of manipulation skills. *Robotics and Autonomous Systems* 164:104413
76. Mühlbauer M, Stulp F, Calinon S, Albu-Schäffer A, Silvério J. 2025. A unified framework for probabilistic dynamic-, trajectory- and vision-based virtual fixtures. *arXiv:2506.10239*
77. Hauberg S, Feragen A, Enficiaud R, Black MJ. 2016. Scalable robust principal component analysis using Grassmann averages. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 38(11):2298–2311
78. Li X, Zeng H, Li Y, Song A. 2024. Quantification of upper-limb motor function for stroke rehabilitation through manifold similarity of muscle synergy. *IEEE Robotics and Automation Letters* 9(2):1859–1866
79. Chevallier E, Barbaresco F, Angulo J. 2015. *Probability Density Estimation on the Hyperbolic Space Applied to Radar Processing*. In *Geometric Science of Information*, pp. 753–761. Springer Intl Publishing
80. Jaquier N, Rozo L, González-Duque M, Borovitskiy V, Asfour T. 2024. *Bringing motion taxonomies to continuous domains via GPLVM on hyperbolic manifolds*. In *Proc. Intl Conf. on Machine Learning (ICML)*, pp. 21373–21409
81. Zefran M, Kumar V. 1996. *Planning of smooth motions on SE(3)*. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pp. 121–126
82. Arimoto S, Yoshida M, Sekimoto M, Tahara K. 2009. A Riemannian-geometry approach for modeling and control of dynamics of object manipulation under constraints. *Journal of Robotics* 2009:1–16
83. Biess A, Flash T, Liebermann DG. 2011. Riemannian geometric approach to human arm dynamics, movement optimization, and invariance. *Phys. Rev. E* 83(3):031927
84. Albu-Schäffer A, Sachtler A. 2023. *What can algebraic topology and differential geometry teach us about intrinsic dynamics and global behavior of robots?* In *Robotics Research*, ed. A Billard, T Asfour, O Khatib, pp. 468–484, pp. 468–484. Cham: Springer Nature Switzerland
85. Jaquier N, Asfour T. 2023. *Riemannian Geometry as a Unifying Theory for Robot Motion Learning and Control*. In *Robotics Research*, ed. A Billard, T Asfour, O Khatib, pp. 395–403, pp. 395–403. Cham: Springer Nature Switzerland
86. Li Y, Qiu J, Calinon S. 2025. A Riemannian take on distance fields and geodesic flows in robotics. *arXiv:2412.05197*
87. Hauberg S, Freifeld O, Black MJ. 2012. *A Geometric take on Metric Learning*. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2024–2032
88. Mainprice J, Ratliff N, Schaal S. 2016. *Warping the workspace geometry with electric potentials for motion optimization of manipulation tasks*. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pp. 3156–3163
89. Cheng CA, Mukadam M, Issac J, Birchfield S, Fox D, et al. 2018. RMPflow: A computational graph for automatic motion policy generation. *Proc. Intl Workshop on the Algorithmic Foundations of Robotics (WAFR)* :1–45
90. Bennequin D, Fuchs R, Berthoz A, Flash T. 2009. Movement timing and invariance arise from several geometries. *PLoS Comput. Biol.* 5(7):1–27
91. Mirebeau JM. 2019. Riemannian fast-marching on Cartesian grids, using Voronoi's first reduction of quadratic forms. *SIAM Journal on Numerical Analysis* 57(6):2608–2655
92. Raissi M, Perdikaris P, Karniadakis GE. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics* 378:686–707
93. Kelshaw D, Magri L. 2025. *Computing distances and means on manifolds with a metric-*

constrained eikonal approach. In *Proceedings A*, vol. 481, pp. 20240270. The Royal Society