A New Orthodoxy in Three Parts

The attached document is in three acts:

- This short introduction;
- A technical whitepaper, "A New Orthodoxy"; and
- A weighty set of endnotes with a glossary of terms, code examples, and general support for "A New Orthodoxy."

Each section is standalone, and given the length of the endnotes, it can be initially set aside.

To provide some context. GeoAutonomy, LLC was created approximately two years ago by a group of professionals who successfully built companies, new technologies, and products that have changed modern society and created $ billion in value for our investors. During the last ten years, we have been involved with autonomy, motion management, and AI by leveraging geometric algebra, an up-and-coming discipline of applied mathematics and physics. Previously, we sold successful products and were granted patents on our earlier geometric algebra work. GeoAutonomy has created new patentable work, a significant new framework and language, and new physical products that address motion management, autonomy, specific AI applications, sensor management, flight, and other technical challenges.

This "New Orthodoxy" points out a new approach that solves persistent problems preventing the full promise of autonomy and intelligent platforms. This title came about because we considered Luther's audacity in nailing his 95 theses to the door in Wittenberg, changing world history, and our need to nail up our points.

This document does not intend to describe GeoAutonomy's technology but instead is intended to stimulate deeper discussions.

A New Orthodoxy

This document intends to highlight the deficiencies in how much of autonomy is being implemented today. Attached are our points of geometric algebra[i] compared to the previous orthodoxy for managing motion, autonomy, and advanced behavior for platforms that fly, drive, dive, or swim.

The Lucretius fallacy [ii] can be summarized as believing that whatever a person or group has experienced is the most extreme or best. In this case, the prevailing views of the framework for managing motion, platform autonomy, and behavior decisions have proceeded as if proven, often ignoring facts inconvenient to the current perception.

Newtonian physics reigned exclusively until challenged by a young patent clerk, who had to partially create a new form of tensor math to describe space-time[iii]. Einstein's determinism reigned supreme until it was replaced at the atomic level by quantum mechanics, beginning with its probabilistic young Turks of Bohr, Heisenberg, and Schrödinger and their uncertainty mapping. Dirac and Feynman used Hamiltonian mechanics to predict and later discover atomic components and interactions for their separate Nobel prize-winning work. The same W. R. Hamilton also promoted the study of quaternions[iv], the forerunner of the larger body of math called Geometric Algebra, sometimes called Clifford Algebra or simply GA (or Conformal Geometric Algebra "CGA" for a specific version of GA that maintains homogeneity for Euclidean space)[v]. For the last 60 years, the high priest of GA has been David Hestenes, who created and defined a unifying version of GA practiced by a small but growing group of scientists and engineers, mostly in robotics, computer graphics, classical and quantum physics, electrodynamics, and relativity. This work has permeated much of our everyday existence with a general lack of appreciation that it lurks "under the hood." Our team's history with GA comes from work associated with JPL and CalTech, drawing directly from Hestenes, which were initially used for navigation in deep space and now navigation/motion management and autonomy on Earth.

GeoAutonomy is focused on solving fundamental problems in motion management, autonomy, and generalized AI. We are focused on the essential issues plaguing society to make the world safer and more efficient, freeing humanity to higher-order tasks. While we are generally considered experts on flight and flying platforms and related autonomy, nothing here is limited to just flight.

In this version of our Geometric Algebra Document, we point out that traditional approaches (the previous orthodoxy) are broken, if not wrong, and that there is a better way to move forward.

Gravitation vector:

For many platforms (UAVs specifically), the gravitational vector ('nadir' — pointing towards the center of mass of the Earth) is lost soon after the "lift-off" or, in the case of water, soon after a fixed location is left. While the internal navigation unit (IMU) contains all the gyroscopes and accelerometers to maintain this vector, it is nonetheless lost for all known UAV applications within seconds of take-off, requiring GPS to correct and sustain this vector based on a satellite fix. No GPS, no stable flight. Indoor flight can maintain the gravitation vector by using external sensors to compute "optical flow," but these techniques require optimal surfaces and lighting. Also, the transition from indoor to outdoor or vice versa is an unstable mess at best.

These limitations are because flight (or swim) control is based on changing positions relative to a reference framework system. With every movement, a new value is obtained from the IMUs and sensor, creating an updated world frame of orientation. These accumulated Euler angle errors regarding gravity become catastrophic in seconds without some extra help.

Using geometric algebra (GA), or more specifically, a well-characterized sub-algebra of generalized GA, there is no need for a common world frame, except for the unique instances that one is required, in the case of the gravitational vector that is rare, if ever, where all sensors (multiple IMUs or optical sensors) are maintained and self-contained in their own GA operators. Even with the cheapest IMUs, the gravitational vectors, sufficient for stable flight, can be maintained for over 20 minutes. For slightly higher-grade IMUs, this time can be significantly extended. The clock starts over every time there is a new landing, a GPS fix, or a new external reference. Since there is a common GA framework for all sensors (more later), all data contributes to stability without accumulating Euler noise.

Besides avoiding a sea of Euler frames and their combined noise, using GA to maintain the gravitational vector gives way to an improved way of handling GPS and sensors. Instead of treating GPS as having sufficient satellites, GA can treat each satellite as a unique input into the GA fabric of computation, treating them as probabilities. If flying in a warehouse with windows, even a brief peak at a few satellites can update the orientation model based on the statistical way the GA space manages orientation in its dense dimensional perspective. More importantly, this means no rough transition with an unstable blink from outdoor to indoor or vice versa.

Immune to GPS and other sensor loss, including jamming:

Geometric algebra offers the advantage of being 'immune to GPS loss,' meaning it can continue functioning even when GPS signals are disrupted or unavailable. For the reasons described above, using GA, GPS is simply a probabilistic input into the motion kinematics, managed in the dense multivectors of the mathematical space. Unlike almost any other approach, GPS is utilized, whether wholly or partially available. If GPS is blocked, then stability and location management by the motion management model are based not on the coordinate grid of the GPS universe but on the density and addition precision of the GA-managed "universe." IMU data is more precise via GA but can be updated by these "blinks" of updates when external sensors can be trusted and managed in the probabilistic dense nature of GA. The same applies to inputs from other sensors, including visual, Lidar, etc.

GPS is increasingly jammed or spoofed in battlefield situations to create false coordinates. A spoofing attempt can be foiled based on this probabilistic approach and the ability to generate assurance around fusing multiple sensors.[vi]

Also detrimental to stable flight is losing a compass heading next to a power line, buildings with metal (inside and out), or a natural or artificial canyon (a downtown city with tall buildings). UAVs often fail when flying close to a tall structure or wall, especially reinforced concrete. This makes getting up close to inspect a bridge problematic (under the bridge is generally a no-go). Using GA as described, these restrictions go away.

Sensor management and coordinate-free integration:

In traditional approaches, every sensor's perspective is managed by forcing it into a common world frame at a relatively high frequency. Using GA this is mainly unnecessary since every sensor and perspective is uniquely preserved and maintained in its own frame without accumulating the noise of conversion. More specifically, a common language based on GA's multivectors is created based on a common geometric underpinning.

Because of the "coordinate-free" nature of GA and its ability to define and maintain all sensor perspectives uniquely and separately while calculating results without requiring a common world frame, adding additional sensors is a straightforward and additive process. This starkly contrasts the sensor and computational tsunami currently at play with autonomous platforms, which require dozens of additional computational elements with every new sensor perspective, resulting in extra cost, complexity, and overhead.

Stability for real-world inspections:

Because of the problems previously described with traditional approaches, there is an inability to hold a platform stable enough to gather micron-level precision, with or without additional sensors. This problem is with motion-flight control systems based on traditional non-GA methods. The further mathematical precision obtained by these GA techniques, as demonstrated by the structured light scanning test published by EPRI (the Energy and Power Research Institute), show a GA-controlled UAV able to effectively gather this level of precision indoors without GPS or optical flow to be practically indistinguishable from a fixed mounted scanner (called a hand scanner in the report). This directly applies to extending the life of power generation plants, inspecting the million-plus underground electrical vaults, or maintaining our transportation infrastructure. Footnote required

Rotational efficiencies:

The design of any system interacting with the real world will require understanding/representing/managing the rotations (or angular position). Even passenger cars will need some angular management, as the car's orientation will depend on its linear translation and the angle at which its wheels are pointing. During a turn, it will travel angularly over an arc. In the case of a platform that flies or swims, these rotations (and translations) can happen in every direction. Further complicating matters is maintaining velocities and accelerations in these rotational domains.

Several mathematical techniques exist for angular/rotational management: Euler angles, rotation matrices, quaternions, etc. Each of these approaches has merits but also limitations. Rotation matrices, for example, specify rotations around each of the principal axes of a coordinate system. This approach has merit because the rotations are defined around the x-y-z axes and can be applied in any chosen order — which is relatively easy to understand/visualize. The limitations are that specific rotations can lead to the expression of gimbal lock and the requirement to maintain these rotation matrices as orthogonal (an additional set of computations).

The use of quaternions (a sub-algebra of GA), which much of the aerospace industry has moved to, eliminates the possibility of gimbal lock. Quaternions have further merit in that a particular axis of rotation (outside of the three principal axes) can be chosen as desired, which removes the need to compose rotations. Avoiding the requirement of orthogonalization (as required by rotation matrix methods), rotations utilizing quaternions require that they are normalized in magnitude to a value of 1 (far less complicated than orthogonalization of a 3x3 matrix).

GA expands on the merits of quaternions by retaining the benefits and allowing further objects to be rotated in a standard mathematical operation. Geometric algebras of appropriately chosen dimensions/signature provide simple representations of geometric objects (spheres, planes, lines, circles, ellipses, etc.) and the ability to rotate these objects around any chosen axis (in any desired plane). More interestingly, conformal geometric algebras allow for translations to be composed with rotations simultaneously and in the same notation as for rotations. GA rotations are simple to describe, compute, and manage, expressed in a single line of GA code (more on this later) versus the pages of equivalent code for other techniques.[vii]

Coding density and hence error reduction:

In maybe the most famous book on extensive system programming, Fred Brooks (program manager of the OS-360 at IBM) in the "Mythical Man Month" warns of drifting away from the central pillars of keeping the development group as small as possible, points out that your key contributors are an order of magnitude more productive than others, suggests standardizing on your essential tools and maintaining them internally, and further points out that there are always proofreader errors so limit your code size (irreducible # of errors).

Attached [viii]as an endnote is an example of a standard 3D rotation, the bread and butter of any motion management, including all UAVs and cars, written in Rust. The first example, #1, is a traditional rotation matrix (SO(3)) implementation, and this code calls upon an external math library of 30,000 plus lines of executable code to support such functions. Example #2 is a rotation in GA. #2 only requires standard library traits found in Rust and no external libraries. Besides the additional precision already discussed, you can quickly see that the executable code is remarkably smaller and more concise (we are not suggesting that all 30,000 lines of the matrix math libraries are used in this example, but even utilizing 1% would add 300 lines of code to that example). Example #3 is the main function for running and printing these two examples, which is provided to show that nothing is hidden.

More importantly, #2 is written in "geometry" for geometric operations, precisely what a rotation is. Angles are preserved, as is noise. Consider how many rotations happen per second when a UAV or car drives. Then, consider all the angles and projections that need to be rotated.

Other examples could be given, including translations (SE(3) — 4D matrix for simultaneous translation/rotation), path planning, or probabilities of collisions.

Screw mechanics:

Screw mechanics of GA enables the combination of rotational and translational equations of motion for a rigid body into a single equation. This reduces the complexity of managing all motion and its associated kinematics. [ix]

Unified math and physics:

Synthetic geometry is fully integrated with computational geometry, allowing rotational and translational operators to be handled evenly with the same algebraic properties.[x] This simplifies operations once GA is used as a unified mathematical framework. Related is that vectors can uniquely represent spheres and hyperplanes: unifying objects, their treatment, and physics. [xi] This makes autonomy based on consistent geometry a reality.

Distributed processing:

Using the standard "language" of GA, we and others have been able to spread the processing load across multiple processors. We have distributed the processes via Movidius vector processors, Nvidia GPU processors, and numerous general-purpose processors. We have split the load between an Nvidia array running on a ground controller and the same flying in the air. A common unified mathematical framework based on a shared library, communicating in a standard "language," all based on GA, gives way to distributed processing and edge computing.

This approach is light and portable and does not require an internet connection to cloud resources.

Recognition benefits using multivectors:

As Gary Marcus comments about the current wave of AI, "Deep Learning is just a fancy way of doing pattern recognition." Using neural nets, we train the system based on massive data sets, leading to impressive results based on leveraging these learned patterns. However, what is learned is often not understood, frequently called "fragility."

One area of advanced recognition and training is Geometric Intelligence, Vector Recognition, and Vector Databases. These approaches would benefit significantly from the additional density of GA's multivectors. Furthermore, these GA-related techniques (Vector Recognition uses a lower-order sub-algebra compared to GA) are far less fragile, given that they expose how they were trained.[xii] This gives way to a more efficient way to train neural net elements.

GA is a mathematical framework for practically everything:

Geometry is at the heart of all autonomous problems and solutions, and using higher math for geometry is common sense. GA provides the basis for a unified mathematical "language" and framework for perceiving the world via sensors, understanding commands, symbolic and semantic reasoning, and processing.

Gimbal Lock:

Gimbal Lock has long plagued human flight, the most famous being on the Apollo 10 mission when a lunar lander was almost lost. While usually considered a mechanically related failure, it is a mathematical flaw of traditional 3-axis matrix control systems.

Quaternions, a subalgebra of GA, alleviate the possibility of gimbal lock being eliminated. Thus, quaternions have found broad applications in aerospace applications. Geometric algebra offers a more robust and generalized solution to that of Euler angles/rotation matrices/quaternions.

Integrating autonomy and behavior information into the fabric of GA:

Using the additional dimensions of GA space/math, information about objects in the space can be embedded and managed. Specifically, information that influences and predicts intentional or unintentional behaviors. This ability to use a unified mathematical GA framework to maintain and make decisions is at the heart of the future of autonomy, including perception and complete or partial behaviors. In this case, partial behaviors mean the thin behaviors for specific applications. A UAV flying in a mine has a vastly different and more limited set of objects, symbolic and associated semantic information, and end behavior it has to manage, compared to a fully autonomous car driving in a city.

GA's framework allows for assigning potentially unlimited tags, characteristics, and properties to objects and environments. These can span a wide range, from threat and physical attributes to potential paths and ethical/moral issues (such as a playground area). GA's strength lies in its ability to embed and maintain all this information in a unified manner, leveraging the computational advantages of GA for immediate access.

Spheres, paths, and probabilities:

Why should you use a higher-dimensional geometric algebra if your problem is from the 3D real world? One reason is that problems can often be formulated more easily and intuitively in more dimensions. One advantage of GA, for instance, is that points, spheres, and planes are easily represented as vector combinations.

Historically, the case of calculating potential collision probability is not a simple action, nor is it easy to describe from the perspective of an avoidance path. GA allows the description of shapes (think paths) based on eclipses, spheres, and other shapes while embedding the physics and probabilities of the various objects and platforms. Intersections, reflections, translations, and projections of these objects are first-order simple calculations in GA. Geometric operations can be expressed easily in Geometric Algebra. Another feature is Duality:

Geometric Algebra allows for division by geometric objects. Switching to the dual can often transform a complex problem into a simpler one. [xiii]

Geometric solutions for a geometric world:

As simple as this statement is, it is the heart of the advantage of the GA approach for motion management and behavior-based autonomy. Commands, behavior, objects, and their properties are expressed in a standard and unified way. A common mathematical framework and desired mission outcomes are the vocabularies of all levels of the system (we call the two levels of our system the Framework for Multi-dimensional Motion "FMM" and Motion Behavioral Language "MBL").

EndNotes

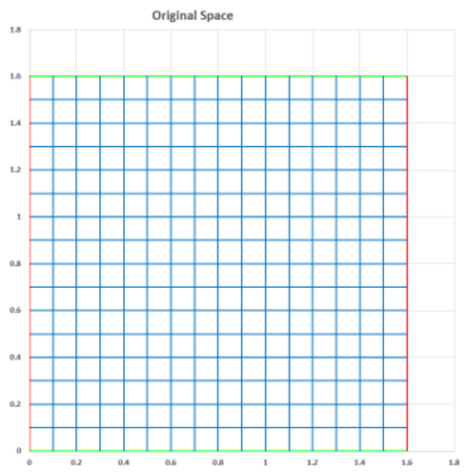
[i] **A Glossary**
Basis
In Euclidean space, a set of mutually orthogonal vectors such that any point in the space is uniquely expressible in terms of those vectors.

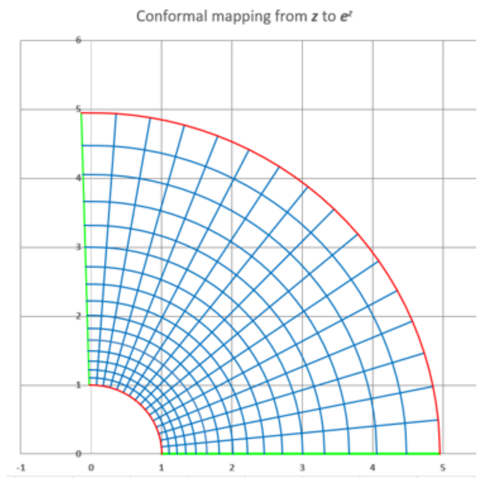
Clifford signature
Denotes a geometric algebra with a pair or triplet of parameters (p, q) or (p, q, r) and the number system used for magnitudes. "p" indicates the number of basis vectors used which square to a value of +1, while "r" gives the number of basis vectors which square to a value of negative 1. If the third term (r) is used, it represents the number of vectors which square to zero (these would be similar to the epsilons used for taking limits in calculus). In this document, the number system used is the real numbers, as opposed to complex or quaternionic numbers.
The closest Clifford algebra to familiar 3-dimensional space is $Cl_{3,0}(\mathbb{R})$, while the most commonly used Conformal geometric algebra (often called simply "4,1" by practitioners) uses a total of five basis vectors, and may be notated using a G or CG instead of C: $G_{4,1}(\mathbb{R})$. This emphasizes the distinction between Clifford and Conformal GAs. Whereas in Clifford Algebra, many algebraic manipulations are possible, many of the expressions which can be generated would be meaningless in CGA. In CGA, one is careful not to attempt constructing entities by combining blades of differing grades. Each entity consists of a single blade or sum of blades within the same grade (see CGA blade chart in document footnotes for an example). Rather than being a constructive process, CGA decomposes a general algebraic expression into terms of matching grades, arriving at the appropriate implied geometric entities, their orientation, and their locations if required.
There are many Clifford and Conformal geometric algebras in use or being researched, and other variations of notations are possible.
In this paper, we do not use the mathematical definitions of Clifford and Conformal geometric algebras (invoking Quadratic Forms, etc.), because that would require a greater degree of technical sophistication than required for our main arguments.

Commutative
The property that an operation has the same result regardless of the order in which the operands are combined. In ordinary multiplication, **a x b = b x a**. This is not true for anticommutative operations, where, generally, **a x b = - b x a**.

Conformal map
A mapping which preserves angles at all scales and also preserves shapes on a small scale. Distortions may emerge in some conformal mappings as one zooms out.
Example: the Mercator map projection. Although the Mercator World map has familiar wild distortions of Greenland and Antarctica's areas, any straight line drawn on this map represents an actual compass bearing. Also, shapes are nearly correct for small areas, and more so for areas near the equator.

In the example created below using the exponential function, notice that a square grid is converted to a circular sector. The overall distortion is considerable, yet all the grid intersections remain as right angles, and on a small scale (within each grid, say) the shape distortion is minimal. The left red line on the original space converts to the inner red arc on the mapped side.





Dirac matrices
A set of matrices, also called the gamma matrices, used in quantum physics, that are connected to the Clifford algebra notated as $\mathbf{Cl_{1,3}(\mathbb{R})}$. They appear in the famous Dirac Equation.

Euler angles
Used to denote the orientation of an object with respect to a fixed coordinate system. In three dimensions, any rotation of an object can be described using three angles each of which are rotations about a particular axis. They can also represent a varying frame of reference, such as that of a moving vehicle. Used in physics, linear algebra, and applied geometry. Attributed to Leonhard Euler.

Gimbal
A rigid frame or ring in which an object is supported by pivots. Two such rings mounted on axes at right angles to each other allow an object such as a ship's compass to remain suspended in a horizontal plane between them regardless of any motion of its support.

Group
In mathematics, a group is a set of elements combined with an operation involving any two elements of the set, usually some form of addition, having an identity element, which corresponds to a "zero", and such that every element has an inverse, meaning that an element combined with its inverse results in the identity.
Example: The position of the hour hand on a clock can represent a group. In this group modular arithmetic would be used. For example, in going past 12, $11 + 5 = 4$, as there is no 16.
Each element has an inverse, e.g. the inverse of 9 is 3: $9 + 3 = 12 = 0$.

Rotations form a group.

Isomorphism
A one-to-one correspondence between the elements of two sets such that the result of an operation on one set corresponds to the result of the analogous result of an operation on their images in the other set.
Example: All (Western) chess sets are isomorphic. The pieces can be matched up one for one between medieval sets, typical Staunton design sets, and geometric-themed Man Ray sets, and, also importantly, they correspondingly move the same way on a chessboard.

Orthogonal
Carries forward the literary meaning of very different or unrelated, so that in statistics it means uncorrelated, and vector math it means lying at or intersecting at right angles (or perpendicular). However, "perpendicular" implies two dimensions or perhaps three, whereas orthogonality is not limited to what we can visualize.
It can also refer to a square matrix with the special property that if it is multiplied by its transpose, the result is the Identity matrix. The notation for this property is: $\mathbf{Q^{-1} = Q^{T}}$.
The set of all n × n orthogonal matrices forms a group called the Orthogonal Group. The set of all n × n orthogonal matrices with determinants that have the value 1 is a subgroup of the orthogonal group called the Special Orthogonal Group. The orthogonal group is denoted O(n) and the special orthogonal group is denoted SO(n). The groups O(n) and SO(n) have several important physical applications, including the description of rotations.

Orthogonalization
The process of finding a set of orthogonal vectors that span a particular space. This is an important issue in computing, as there are several methods, each with its own advantages and drawbacks.

Pauli matrices
The Pauli matrices are a set of simple 2 x 2 matrices containing complex numbers, in which operations between them are equivalent to quaternion operations. They are ubiquitous in quantum mechanics, and are most commonly associated with electron spin and the "Pauli Exclusion Principle", but also play a role in quantum optics and quantum computing.

Projective geometry
The study of how the geometric properties of a figure are altered by projection. There is a one-to-one correspondence between points in a figure and points in its projected image, but often the ratios of lengths will be changed. In central projection for example, a triangle maps into a triangle and a quadrilateral into a quadrilateral, but the sides and angles may change.

Quantum mechanics
The theory that governs the behavior of particles at the atomic and sub-atomic level. It bears the same relationship to Newtonian mechanics as wave optics does to geometrical optics. It is the

foundation of all quantum physics, and includes quantum field theory, quantum chemistry, and quantum information science.
A relatively simple manifestation of quantum mechanics is that electrons release or absorb photons (the smallest particles of light) of only certain distinct wavelengths when they jump from one orbital to another in an atom, and there are no in-between states. This phenomenon is observable as absorption lines in the sun's spectrum, which is not consistent with classical physics. This does not invalidate classical physics, which works quite well at the macroscopic level. One can consider that the summation of super-numerous microscopic quantum processes result in the easily observed macro phenomena of classical physics.

Quaternion
Generalized complex numbers whose discovery (1843) is credited to William Rowan Hamilton, although their existence was first put in print in all but name by the French mathematician and social reformer Olinde Rodrigues in 1840. A quaternion is of the form $\mathbf{a + bi + cj + dk}$, where $\mathbf{i^2 = j^2 = k^2 = -1}$ and $\mathbf{ij = -ji = k}$ and **a, b, c, d** are real numbers. A striking feature of quaternions is that multiplication is not commutative. They have applications in the study of the rotations of rigid bodies in space, and are known to present advantages such as avoidance of gimbal lock.

Radius
The distance from the center of a circle to any point on its circumference or from the center of a sphere to its surface. In polar coordinates, a radius r (distance from a fixed origin) is used with angular position θ to specify the positions of points.

Rotation group
The group consisting of the set of all possible rotations about an axis. This group is a continuous group, i.e. it has an infinite number of members. The rotation group in two dimensions has the property that the concatenation of rotations can be accomplished in any order giving the same final position. However, the rotation group in three dimensions does not. In physical systems the rotation group is closely associated with the angular momentum of the system. There are many applications of the rotation group in the quantum theory of atoms, molecules, and atomic nuclei.

Sandwich operator
A pair of operators which perform an operation on the left and on the right of a mathematical expression. In the case of quaternion rotation, each operator performs a reflection in opposite directions, resulting in a rotation.

SE(3)
Special Euclidean Group in three dimensions. It is used in mathematics, robotics, and computer graphics to describe and manipulate both position and orientation in familiar 3-dimensional space. It employs "homogeneous coordinates" which require a particular matrix representation.

Special orthogonal group
The set of all $n \times n$ orthogonal matrices with determinants that have the value 1 is a subgroup of the orthogonal group called the Special Orthogonal Group.

Tensor
A mathematical entity that is a generalization of a vector. Tensors are used to describe how all the components of a quantity in an n-dimensional system behave under certain transformations, just as a vector can describe a translation from one point to another in a plane or in space.

Translation
The moving of a geometrical figure so that only its position relative to fixed axes is changed, but not its orientation, size, or shape.

Unit vector
A vector with a magnitude of one unit.

Vector
A measure in which direction is important and must usually be specified. For instance, displacement is a vector quantity, whereas distance is a scalar. Weight, velocity, and magnetic field strength are other examples of vectors – they are each quoted as a number with a unit and a direction. Vectors are often denoted by printing the symbol in bold font. Vector algebra treats vectors symbolically in a similar way to the algebra of scalar quantities but with different rules for addition, subtraction, multiplication, etc. Any vector can be represented in terms of component vectors. In particular, a vector in three-dimensional Cartesian coordinates can be represented in terms of three unit vector components i, j, and k directed along the x-, y-, and z-axes respectively. In Geometric Algebra and its sub-algebras, i, j, and k are often reserved for the components of a quaternion, and the x-, y-, z-axis unit vectors are typically labeled $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$.

[ii] In Antifragile, Nassim Taleb writes:

Indeed, our bodies discover probabilities in a very sophisticated manner and assess risks much better than our intellects do. To take one example, risk management professionals look in the past for information on the so-called worst-case scenario and use it to estimate future risks – this method is called "stress testing." They take the worst historical recession, the worst war, the worst historical move in interest rates, or the worst point in unemployment as an exact estimate for the worst future outcome. But they never notice the following inconsistency: this so-called worst-case event, when it happened, exceeded the worst [known] case at the time.

I have called this mental defect the Lucretius problem, after the Latin poetic philosopher who wrote that the fool believes that the tallest mountain in the world will be equal to the tallest one he has observed. We consider the biggest object of any kind that we have seen in our lives or hear about as the largest item that can possibly exist. And we have been doing this for millennia.

Taleb brings up an interesting point, which is that our documented history can blind us. All we know is what we have been able to record. There is an uncertainty that we don't seem to grasp.

We think because we have sophisticated data collecting techniques that we can capture all the data necessary to make decisions. We think we can use our current statistical techniques to draw historical trends using historical data without acknowledging the fact that past data recorders had fewer tools to capture the dark figure of unreported data. We also overestimate the validity of what has been recorded before, and thus the trends we draw might tell a different story if we had the dark figure of unreported data.

Taleb continues:

The same can be seen in the Fukushima nuclear reactor, which experienced a catastrophic failure in 2011 when a tsunami struck. It had been built to withstand the worst past historical earthquake, with the builders not imagining much worse— and not thinking that the worst past event had to be a surprise, as it had no precedent. Likewise, the former chairman of the Federal Reserve, Fragilista Doctor Alan Greenspan, in his apology to Congress offered the classic "It never happened before." Well, nature, unlike Fragilista Greenspan, prepares for what has not happened before, assuming worse harm is possible.

[iii] Albert Einstein was not a mathematician per se, but he is responsible for what is known as the Einstein tensor notation, notable for its brevity, used in his successful theory of gravitation (He and Feynman both grasped that the history of mathematics is largely a series of improvements in notation). Einstein's modification of Newtonian physics turns up in many places as a "relativistic correction factor", but relativity did not alter the idea of a deterministic universe.

[iv] The quaternion is a three-dimensional analog of complex numbers used in trigonometry and calculus. Another British mathematician, William Kingdon Clifford (1845-79) developed an entire branch of modern algebra during his short life, expanding upon the work of Hamilton and the polymath Hermann Grassman. These "Clifford algebras" generalize real numbers, complex numbers, quaternions, and many other hypercomplex number systems, and constitute a considerable area of study.

Clifford algebra is synonymous with Geometric Algebra (or "GA"), with a possible distinction being that GA implies that the "multivectors" involved imply familiar geometric entities. A breakthrough in finding practical applications was made by David Hestenes in recent decades. He found a way to create a space using five basis vectors: three with the familiar x-y-z axes, combined with special "null vectors" to obtain a point at the origin and one at infinity, the latter being analogous to a line or point at infinity in perspective art.

This opened a brave new world of representing geometric operations (intersection, rotation, reflection, dilation) with compact algebraic operations, called "Conformal Geometric Algebra" (CGA).

Since Hestenes' landmark work, additional GAs have been treated in a similar way, each with its characteristic "signature" which denotes the composition of "basis vectors" and thus higher dimensionalities. These higher signature GAs allow for more complex operations and higher order surfaces such as ellipsoids, paraboloids, etc. and new applications are being found. Higher order GAs do require more computing power however.

---

[v] The quaternions exhibit important features of a Geometric Algebra: the geometric product consists of a symmetric and an anti-symmetric part, and isometries (movement of an object or image without changing its shape or size) are "sandwich operators". Consequently, an advantage of Geometric Algebras is that useful subalgebras are embedded within the algebras of higher signature (or dimensionality). This allows for certain operations to be performed where a limited set of multivectors is sufficient, without changing the methods or the symbology.

For instance, in the diagram below, all 32 elements (or "blades") of one instance of Conformable Geometric Algebra are listed with their classification. Highlighted is a subset of 8 which represents "dual quaternions", which could be described as supercharged quaternions geared for performing translation as well as rotation.

List of Blades in Conformal Geometric Algebra
GeoAutonomy©

Grade	Type	possible blades in Grade	Number
0	*scalar*	1	1
1	*vector*	$e_1, e_2, e_3, e_\infty, e_0$	5
2	*bivector*	$e_1 \wedge e_2, e_1 \wedge e_3, e_1 \wedge e_\infty, e_1 \wedge e_0, e_2 \wedge e_3, e_2 \wedge e_\infty, e_2 \wedge e_0, e_3 \wedge e_\infty, e_3 \wedge e_0, e_\infty \wedge e_0$	10
3	*trivector*	$e_1 \wedge e_2 \wedge e_3, e_1 \wedge e_2 \wedge e_\infty, e_1 \wedge e_2 \wedge e_0, e_1 \wedge e_3 \wedge e_\infty, e_1 \wedge e_3 \wedge e_0, e_1 \wedge e_\infty \wedge e_0, e_2 \wedge e_3 \wedge e_\infty, e_2 \wedge e_3 \wedge e_0, e_2 \wedge e_\infty \wedge e_0, e_3 \wedge e_\infty \wedge e_0$	10
4	*quadvector*	$e_1 \wedge e_2 \wedge e_3 \wedge e_\infty, e_1 \wedge e_2 \wedge e_3 \wedge e_0, e_1 \wedge e_2 \wedge e_\infty \wedge e_0, e_1 \wedge e_3 \wedge e_\infty \wedge e_0, e_2 \wedge e_3 \wedge e_\infty \wedge e_0$	5
5	*pseudoscalar*	$e_1 \wedge e_2 \wedge e_3 \wedge e_\infty \wedge e_0$	1

Table of Conformal Geometric Algebra elements, called "blades". Highlighted blades correspond to the elements of the *Dual Quaternion* algebra, a small subset of Conformal Geometric Algebra, although fairly powerful by itself as a technique for computer graphics and small-movement interpolation.

[vi] From Smithsonian Time and Navigation site:
https://timeandnavigation.si.edu/satellite-navigation/gps/risks-to-system

Risks to the System

Threats can imperil satellite navigation systems:

Satellites provide essential navigation services, but threats exist to their operation. Radio interference from both natural and human sources presents serious problems for the system's myriad users. Engineers and scientists continue to develop solutions to ensure the continued operation of global navigation services.

Solar Interference

Solar activity can interfere with satellite signals. Solar storms occasionally interrupt clear reception of signals from space. Those who design satellite systems must plan for these disruptions and be aware of how solar activity varies with the 11-year sunspot cycle.

System Maintenance

The successful operation of a satellite navigation system requires around-the-clock monitoring of the satellites' health and the periodic replacement of older satellites. The process is labor-intensive and expensive and requires multiple backups to ensure continuous operation.

Man-made Radio Interference

GPS and other satellite positioning systems were designed to use quiet parts of the spectrum. However, these channels face the danger of being overwhelmed by communications signals from other nearby frequencies. Engineers must test the possibility of interference from multiple systems.

Intentional Jamming

Although their use is illegal in the United States, portable GPS jammers are traded clandestinely and used by those who wish not to be tracked or otherwise located by GPS. These devices cause nearby navigation systems to malfunction, potentially threatening public safety.

System Under Attack

The increasing reliance on navigation satellites for military and commercial activities makes them a tempting target for an enemy. While it is difficult to disable the satellites themselves, these and other GPS components must be protected from interference or attack.

vii

Approaches to Rotation and Translation

From a working paper by Jacob Davisson

Introduction

In order to properly describe the flight dynamics and physics of a Vehicle, unmanned or otherwise, a number of mathematical techniques must be employed. Some of these methods follow from the nature of problem-solving in physics. Others, may yield under a variety of possible approaches. Handling rigid-body movement is a set of tasks that can be solved using different approaches.

Various methods for representing and applying rotations, and to a lesser extent, translations, will be described in the following pages, with commentary relevant to (1) understanding the approach and (2) how GeoAutonomy has chosen favorable techniques.

Applications

Special Orthogonal Group on $\mathbb{R}^3$

Also known as the *3D Rotation Group*, the Special Orthogonal Group on $\mathbb{R}^3$, ($SO(3)$) is a collection of matrices representing all rotations around the origin in $\mathbb{R}^3$. Matrices in this group are *orthonormal* — meaning that the column and row vectors comprising it are orthogonal and have norm equal to ± 1. In the case of $SO(3)$, the norm is equal to $+1$, and this is why it labeled as *special*. This particular method is called out by name, and utilized in various examples, in the patent filed for by DJI[1].

Rotations in this form are determined by a vector of unit length (around which the rotation is imparted) and the angle which the rigid-body is rotated around that unit vector. Representations of these particular rotations are given below, by the following three 3×3 matrices.

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta_R) & -sin(\theta_R) \\ 0 & sin(\theta_R) & cos(\theta_R) \end{bmatrix} \quad \textbf{Eqn 1}$$

$$R_y = \begin{bmatrix} cos(\theta_P) & 0 & sin(\theta_P) \\ 0 & 1 & 0 \\ -sin(\theta_P) & 0 & cos(\theta_P) \end{bmatrix} \quad \textbf{Eqn 2}$$

$$R_z = \begin{bmatrix} cos(\theta_Y) & -sin(\theta_Y) & 0 \\ sin(\theta_Y) & cos(\theta_Y) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{Eqn 3}$$

Note that in Equations 1, 2, and 3 the represented rotations are around the x-axis, y-axis, and z-axes respectively. For this reason, the entry corresponding to each of these axes is unity along the main diagonal of each matrix. Also, the angular argument for each has been given a subscript that corresponds to the rotational degree of freedom.[2]

As an example of each rotation, we can apply the corresponding matrix to a unit vector along an orthogonal axis, and show how each is rotated. Let X, Y, and Z be unit vectors along the axis that they are named by, let $X^{'}$, $Y^{'}$, and Z' be the transformed (rotated) vectors. Then rotating each one by 90° yields:

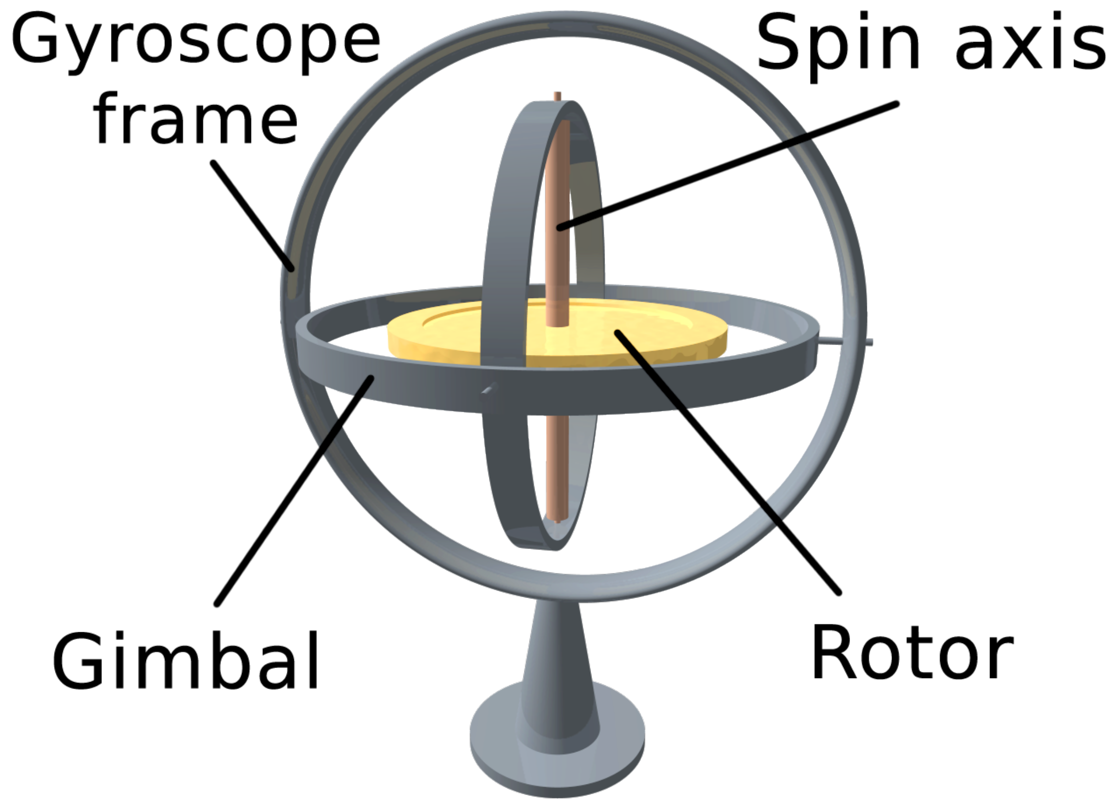
$$Y' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\left(\frac{\pi}{2}\right) & -sin\left(\frac{\pi}{2}\right) \\ 0 & sin\left(\frac{\pi}{2}\right) & cos\left(\frac{\pi}{2}\right) \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = Z \quad \textbf{Eqn 4}$$

$$Z' = \begin{bmatrix} cos\left(\frac{\pi}{2}\right) & 0 & sin\left(\frac{\pi}{2}\right) \\ 0 & 1 & 0 \\ -sin\left(\frac{\pi}{2}\right) & 0 & cos\left(\frac{\pi}{2}\right) \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = X \quad \textbf{Eqn 5}$$

$$X' = \begin{bmatrix} cos\left(\frac{\pi}{2}\right) & -sin\left(\frac{\pi}{2}\right) & 0 \\ sin\left(\frac{\pi}{2}\right) & cos\left(\frac{\pi}{2}\right) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = Y \quad \textbf{Eqn 6}$$

Gimbal Lock

This rotation group, $SO(3)$, appears sufficient for handling rotations of a rigid body in 3-space. As shown in Equations 4, 5, and 6 the position vectors are rotated around the particular axis, at the specified angle. An issue, which is not immediately obvious, can arise when utilizing $SO(3)$ where two or more axes of rotation can become aligned and a *degree of freedom*[3] is lost — this condition is known as *Gimbal Lock*. It can be shown how SO(3) can suffer *gimbal lock* by considering a mechanical example[4], and then applying that knowledge mathematically using $SO(3)$.



A mechanical gyroscope, with all three axes orthogonally oriented.

The figure shows a mechanical gyroscope, with its all three of its axes in an orthogonal configuration — all three degrees of freedom are completely independent. How could we align these axes so that the resulting configuration would exhibit *gimbal lock*? Take any of the three orthogonally-mounted rings, and align it with any of the other two so that they are concentric[5]. This is done by rotating one of these rings 90° in either direction. Once this is done, *gimbal lock* is achieved, and one of the system's degrees of freedom is lost — due to the coupling of two axes.

In order to show how *gimbal lock* is shown in $SO(3)$, the three rotation matrices[6] can be multiplied together[7] and a 90° rotation mathematically imparted by argument substitution.

Eqn 7

$$R = R_x R_y R_z$$
$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta_R) & -sin(\theta_R) \\ 0 & sin(\theta_R) & cos(\theta_R) \end{bmatrix} \begin{bmatrix} cos(\theta_P) & 0 & sin(\theta_P) \\ 0 & 1 & 0 \\ -sin(\theta_P) & 0 & cos(\theta_P) \end{bmatrix} \begin{bmatrix} cos(\theta_Y) & -sin(\theta_Y) & 0 \\ sin(\theta_Y) & cos(\theta_Y) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} cos(\theta_R) & 0 & sin(\theta_R) \\ sin(\theta_R)sin(\theta_P) & cos(\theta_R) & -sin(\theta_R)cos(\theta_P) \\ -cos(\theta_R)sin(\theta_P) & sin(\theta_R) & cos(\theta_R)cos(\theta_P) \end{bmatrix} \begin{bmatrix} cos(\theta_Y) & -sin(\theta_Y) & 0 \\ sin(\theta_Y) & cos(\theta_Y) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} cos(\theta_P)cos(\theta_Y) & -cos(\theta_P)sin(\theta_Y) & -sin(\theta_P) \\ sin(\theta_R)sin(\theta_P)cos(\theta_Y) + cos(\theta_R)sin(\theta_Y) & -sin(\theta_R)sin(\theta_P)sin(\theta_Y) + cos(\theta_R)cos(\theta_Y) & cos(\theta_R)cos(\theta_P) \\ cos(\theta_R)sin(\theta_P)cos(\theta_Y) - sin(\theta_R)sin(\theta_Y) & -cos(\theta_R)sin(\theta_P)sin(\theta_Y) - sin(\theta_R)cos(\theta_Y) & cos(\theta_R)cos(\theta_P) \end{bmatrix}$$

Equation 7 represents the product of all three $SO(3)$ rotations matrices, in the order prescribed on the top line. If a 90° pitch rotation were imparted on the gyroscope, then Equation 7 would take the following values:

$$R\left[\theta_R, \frac{\pi}{2}, \theta_Y\right] = \begin{bmatrix} cos\left(\frac{\pi}{2}\right)cos(\theta_Y) & -cos\left(\frac{\pi}{2}\right)sin(\theta_Y) & -sin\left(\frac{\pi}{2}\right) \\ sin(\theta_R)sin\left(\frac{\pi}{2}\right)cos(\theta_Y) + cos(\theta_R)sin(\theta_Y) & -sin(\theta_R)sin\left(\frac{\pi}{2}\right)sin(\theta_Y) + cos(\theta_R)cos(\theta_Y) & cos(\theta_R)cos\left(\frac{\pi}{2}\right) \\ cos(\theta_R)sin\left(\frac{\pi}{2}\right)cos(\theta_Y) - sin(\theta_R)sin(\theta_Y) & -cos(\theta_R)sin\left(\frac{\pi}{2}\right)sin(\theta_Y) - sin(\theta_R)cos(\theta_Y) & cos(\theta_R)cos\left(\frac{\pi}{2}\right) \end{bmatrix}$$
$$= \begin{bmatrix} 0 & 0 & -1 \\ sin(\theta_R)cos(\theta_Y) + cos(\theta_R)sin(\theta_Y) & -sin(\theta_R)sin(\theta_Y) + cos(\theta_R)cos(\theta_Y) & 0 \\ cos(\theta_R)cos(\theta_Y) - sin(\theta_R)sin(\theta_Y) & -cos(\theta_R)sin(\theta_Y) - sin(\theta_R)cos(\theta_Y) & 0 \end{bmatrix}$$

Eqn 8

Notice that the 90° pitch angle has removed a degree of freedom from Equation 8 — element a_{13} has changed to -1, which is not a modifiable degree of freedom any longer and will not respond to pitch-angle changes.

Translations Described by $SE(3)$

Rotations are not the only required rigid-body motion that must be accounted for in UAV flight systems. Without being able to describe and apply translations inside of the flight control system, the best you could hope for is a drone to rotate in place. In order to describe and apply translations with $SO(3)$, we need to extend the space to a 4-dimensional one, and expand the matrices (as given by Equations 1, 2, and 3) to 4×4 matrices. The group comprising these expanded, 4×4 matrices is known as the *Special Euclidean Group* — denoted $SE(3)$.

A transform matrix in $SE(3)$ is some variation of the following block matrix:

$$\mathbf{T} = \begin{pmatrix} R & V \\ 0_{1\times3} & 1 \end{pmatrix}$$

Eqn 9

Limitations of $SO(3)$

The undesirable condition of G*imbal lock*, as described above, may or may not be possible depending upon the system. Ignorance of it however can be catastrophic in some applications. It is not the only limitation that exists when performing rotations using

$SO(3)$. Another important limitation arises from the properties required of these rotation matrices — orthogonality. Utilization of this technique to impart, or describe, rotations is simple enough on paper where floating point errors are not so much a concern. When applying $SO(3)$, multiple rotations cause the accumulation of many small errors as part of a code base[8] floating point, noise, uncertainty, and rounding errors. Care must be taken to require that these rotation matrices are *re-orthogonalized* when concatenating rotations, which is inevitable.

There are a number of different approaches to *re-orthogonalization* of rotation matrices. Among these are: *Singular-Value Decomposition*, *Gram-Schmidt Orthonormalization*, *QR Decomposition*, etc. An outline of these methods may be given in a future revision of this document, but for now they are out of scope. An abundance of information relative to these methods (and others) is available online, or in linear algebra textbooks.

Since rotations by employing $SO(3)$ rotation matrices are performed using the orthonormal basis vectors of $\mathbb{R}^3$, the *orthonormal* nature of these matrices must be upheld. If it is not, then, as noted, the system that these rotations are applied to will be accumulating errors in any state variable affected by rotation — attitude, orientation, etc.

Quaternions

The group known as the *Quaternions* are often used for handling rotations in aerospace and video game/computer graphics applications. There are a number of benefits in utilizing quaternions instead of the $SO(3)$ rotation matrices described above, and chief among them is the ability to directly perform a rotation around an arbitrary axis — instead of being restricted to rotations around the three basis vectors of the space.

Quaternion algebra is unique, though it does share some properties with ordinary complex algebra. This algebra can certainly be a bit more computationally intensive when done on paper, due to the number of terms found in each product — though, once the general case for each operation is determined by hand, a computer program function can be easily written to handle the computations quickly and effectively.

Properties of the Quaternion Algebra

A quaternion is comprised of a scalar and vector part — with the latter being identified by the **i**, **j**, and **k** unit vectors. The canonical forms of a quaternion follow:

$$q = s + v \qquad \textbf{Eqn 10}$$

$$= \sigma + \alpha\mathbf{i} + \beta\mathbf{j} + \gamma\mathbf{k} \qquad \textbf{Eqn 11}$$

with σ being the scalar part and $\alpha, \beta,$ and γ being coefficients of the unit vector components $\mathbf{i}, \mathbf{j},$ and $\mathbf{k}$. Quaternion algebra has a few unique properties, and some familiar from ordinary complex number algebra. For example, addition of two quaternions is given below.

$$q_1 + q_2 = (s_1 + s_2) + (\boldsymbol{v_1} + \boldsymbol{v_2}) \qquad \textbf{Eqn 12}$$

$$= (\sigma_1 + \sigma_2) + (\alpha_1\mathbf{i} + \beta_1\mathbf{j} + \gamma_1\mathbf{k} + \alpha_2\mathbf{i} + \beta_2\mathbf{j} + \gamma_2\mathbf{k}) \qquad \textbf{Eqn 13}$$

$$= (\sigma_1 + \sigma_2) + (\alpha_1 + \alpha_2)\mathbf{i} + (\beta_1 + \beta_2)\mathbf{j} + (\gamma_1 + \gamma_2)\mathbf{k} \qquad \textbf{Eqn 14}$$

Complex conjugation of quaternions is also a familiar operation, performed in similar manner to doing so in complex numbers $z \in \mathbb{C}$.

$$\begin{aligned} q^* &= (s + \boldsymbol{v})^* \\ &= s - \boldsymbol{v} \\ &= s - (\alpha\mathbf{i} + \beta\mathbf{j} + \gamma\mathbf{k}) \\ &= s - \alpha\mathbf{i} - \beta\mathbf{j} - \gamma\mathbf{k} \qquad \textbf{Eqn 15} \end{aligned}$$

An important operation when working with $z \in \mathbb{C}$, conjugation provides for a simple way to determine magnitude of complex numbers (both $z \in \mathbb{C}$ and quaternions).

$$\|q\| = \sqrt{qq^*} = \sqrt{q^*q} = \sqrt{s^2 + \alpha^2 + \beta^2 + \gamma^2} \qquad \textbf{Eqn 16}$$

Normalizing an arbitrary quaternion is done in the same manner as in other algebras (vector algebra, complex algebra, ...) by division with the norm — yielding a unit-length quaternion.

$$q_{unit} = \frac{q}{\|q\|} = \frac{q}{\sqrt{qq^*}} = \frac{q}{\sqrt{q^*q}} = \frac{q}{\sqrt{s^2 + \alpha^2 + \beta^2 + \gamma^2}} \qquad \textbf{Eqn 17}$$

The application of Equations 16 and 17 allow a multiplicative inverse ("reciprocal") for non-zero quaternions to be defined.

$$q^{-1} = \frac{q^*}{\|q\|^2} \qquad \textbf{Eqn 18}$$

Multiplication of quaternions is a non-commutative operation for the vector components (scalar multiplication is still a commutative operation), and for this reason these products are presented below.

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1 \qquad \textbf{Eqn 19}$$

$$\mathbf{jk} = -\mathbf{kj} = \mathbf{i} \qquad \textbf{Eqn 20}$$

$$\mathbf{ki} = -\mathbf{ik} = \mathbf{j} \qquad \textbf{Eqn 21}$$

$$\mathbf{ij} = \text{-}\mathbf{ji} = \mathbf{k} \qquad \textbf{Eqn 22}$$

The anti-commutative nature of the vector components is clear from Equation 20, 21 and 22.

Quaternion Rotations

Spatial rotations of a rigid-body in $\mathbb{R}^3$ performed using the techniques of $SO(3)$ are about one of the particular *orthogonal* axes. Euler's Rotation Theorem[9] is easily applied using $SO(3)$ when rotations are applied around those *orthogonal* axes, but not so easily applied when a *non-orthogonal* axis is more convenient (or necessary). Quaternion rotations of a rigid-body in $\mathbb{R}^3$ allow for another axis to be easily chosen and yield a mathematical treatment of the rotation that is applied to those around the *orthogonal* axes as easily as it is applied to an arbitrarily chosen axis.

In order to perform a rotation by utilizing quaternions, it is necessary to choose both an axis[10] around which to perform the rigid-body rotation and an angle, θ, to rotate the rigid-body through[11].

Choose an arbitrary axis to rotate about, let $\hat{u} = 0 + u_x i + u_y j + u_z k$ (note that this is a *pure quaternion* — in that it has a zero-valued scalar part). An arbitrary angle, θ is chosen as the angle through which we will rotate about $\hat{u}$. These two quantities allow us to define a ***rotor***, in the following manner.

$$\boldsymbol{r} \quad = e^{\left[\left(\frac{\theta}{2}\right)\hat{u}\right]} \qquad \boldsymbol{Eqn\ 23}$$

$$\boldsymbol{r} \quad = e^{\left[\left(\frac{\theta}{2}\right)(0+u_x i+u_y j+u_z k)\right]} \qquad \boldsymbol{Eqn\ 24}$$

$$\boldsymbol{r} \quad = e^{\left[\left(\frac{\theta}{2}\right)(u_x i+u_y j+u_z k)\right]} \qquad \boldsymbol{Eqn\ 25}$$

$$\boldsymbol{r} \quad = cos\left(\frac{\theta}{2}\right) + sin\left(\frac{\theta}{2}\right)\left(u_x i + u_y j + u_z k\right) \qquad \boldsymbol{Eqn\ 26}$$

Note the application of Euler's Formula[12] from Equation 23 to 26.

In order to perform a rotation, the generation of a *rotor* is necessary. Its conjugate is also required, which we can write by the ordinary manner.

$$r \quad = e^{-\left(\frac{\theta}{2}\right)\hat{u}} = cos\left(\frac{\theta}{2}\right) - sin\left(\frac{\theta}{2}\right)\left(u_x i + u_y j + u_z k\right) \qquad \boldsymbol{Eqn\ 27}$$

$$r^{*} \quad = e^{+\left(\frac{\theta}{2}\right)\hat{u}} = cos\left(\frac{\theta}{2}\right) + sin\left(\frac{\theta}{2}\right)\left(u_x i + u_y j + u_z k\right) \qquad \boldsymbol{Eqn\ 28}$$

With Equations 27 and 28 we can write the rotation of an arbitrary vector, $v = a_x i + a_y j + a_z k$ into rotated vector v', in the following way:

$$\boldsymbol{v}' \quad = r\boldsymbol{v}r^{*}$$
$$= \left(cos\left(\frac{\theta}{2}\right) + sin\left(\frac{\theta}{2}\right)\left(u_x i + u_y j + u_z k\right)\right)\boldsymbol{v}\left(cos\left(\frac{\theta}{2}\right) - sin\left(\frac{\theta}{2}\right)\left(u_x i + u_y j + u_z k\right)\right) \boldsymbol{Eqn\ 29}$$

- and from here to be expanded in another revision.

Quaternion Limitations

As with $SO(3)$, the use of quaternions is not without limitations — though they are fewer. By use of $SO(3)$ to describe and perform rotations about an arbitrary axis it is necessary to write these as the composition of rotations around each of the three orthonormal axes of the space. It is also necessary that in applications these matrices be orthogonalized often, to ensure that they retain all of the necessary properties of the group — rotating around an axis that is *supposed* to be orthogonal, but isn't, would result in rotating portions of the vector that are not to be rotated.

Quaternions remove the restrictions of $SO(3)$, by not requiring that rotations occur around one of the three orthogonal axes of the space. The requirement of orthogonalizing, as in the use of $SO(3)$, is also gone — and any axis rotated around with quaternions is expressed as a unit vector.

One similarity that quaternions share with $SO(3)$, is that applications of rotations are restricted to vectors. This works well for rigid-bodies, as a well-chosen position vector representing a point on a rigid-body will rotate the entire rigid-body as desired. It still may mean that there is somewhat of a mixed-representation of objects[13].

Conformal Geometric Algebra

The approaches previously discussed, have their own benefits and limitations. As a specific example, the rotation matrix approach of $SO(3)$ allows for a fairly simple way to rotate vectors around either of the three orthogonal axes — but not around other non-orthogonal axes. The quaternion approach changes this, and allows for rotation of vectors around arbitrary axes — but not more complicated geometric objects. By a suitable application of *Geometric Algebra*[14], any geometric object that can be represented in the space can be rotated around any other object in the space. Previous examples of limitations from other approaches are effectively absent, and the full power of geometry and geometric representations is available to the user.

No discussion, be it in white paper, academic research paper, or other, is considered sufficient regarding *Geometric Algebra* without an introduction to the basic properties of the algebra. This particular section will include a reasonably thorough discussion of the salient properties of the algebra, in order for a more fulfilling discussion of the technique to take place. The use of Geometric Algebra is the approach that GeoAutonomy has chosen for its past, present, and future products. Properties of Geometric Algebra described in the following section will be covering both the ordinary and conformal geometric algebras — as the properties of both share many similarities, with the conformal mapping and certain properties being unique to the conformal algebra.

Algebraic Properties of the (Conformal) Geometric Algebra

A *Geometric Algebra* is an algebra written on a field[15] and may, or may not, extend the dimensionality of said field. It is just as possible to write a geometric algebra on $\mathbb{R}^3$ as $\mathbb{G}^3(\mathbb{R}^3)$ as it is to write $\mathbb{G}^5(\mathbb{R}^3)$. The number of basis vectors corresponds to the dimensionality of the *Geometric Algebra*, and these basis vectors are most commonly denominated $e_1, e_2, \cdots, e_i$, where $i \in \mathbb{N}$. This nomenclature allows for a simpler enumeration of basis vectors when the dimensionality of the *Geometric Algebra* gets sufficiently high.

Some operations from ordinary vector algebra are maintained in this approach, such as the *Inner Product*. In example, let $V_1 = a_1e_1 + b_1e_2 + c_1e_3$ and $V_2 = a_2e_1 + b_2e_2 + c_2e_3$. The inner product of these two vectors, V_1 and V_2 is given by:

$$
\begin{aligned}
V_1 \cdot V_2 &= (a_1e_1 + b_1e_2 + c_1e_3) \cdot (a_2e_1 + b_2e_2 + c_2e_3) && \textbf{\textit{Eqn}}\ \mathbf{30} \\
&= a_1a_2 + b_1b_2 + c_1c_2 && \textbf{\textit{Eqn}}\ \mathbf{31}
\end{aligned}
$$

Equation 30 shows the inner product in geometric algebra as familiar from vector algebra — which works across spaces of any dimension.

An important note to make here, is that the basis vectors do not commute under the geometric product — this is due to the fact that they do not commute under the exterior product — the order of the product of these elements is quite important to note and keep track of. This matter does not arise from the inner product, since they are orthogonal basis vectors the inner product of two dissimilar basis vectors is necessarily zero. Instead, this matter arises from another operation — the other half of the *Geometric Product* — which is colloquially known as the *Wedge Product*, and more specifically known as the *Exterior Product*. Whereas the *inner product* yields the projection of one vector onto another (which shows how similar the orientation is between the two vectors in the product — the parallel component of their orientation), the *exterior product* yields the rejection of the two vectors (how dissimilar the orientation is between the two — the orthogonal component of their orientation). Properties of the *exterior product* of basis vectors can be shown in a relatively short list.

$$
\begin{aligned}
e_i \wedge e_i &= 0 && \textbf{\textit{Eqn}}\ \mathbf{32} \\
e_i \wedge e_j &= -e_j \wedge e_i && \textbf{\textit{Eqn}}\ \mathbf{33} \\
(e_i \wedge e_j)\,(e_i \wedge e_j) &= -(e_i \wedge e_j)(e_j \wedge e_i) = -(e_ie_i) = -1 && \textbf{\textit{Eqn}}\ \mathbf{34} \\
a_1e_i \wedge b_1e_j &= (a_1b_1)e_ie_j && \textbf{\textit{Eqn}}\ \mathbf{35} \\
\alpha(e_i \wedge e_j) &= (\alpha e_i) \wedge e_j && \textbf{\textit{Eqn}}\ \mathbf{36}
\end{aligned}
$$

Equation 34 shows the geometric product of two bivectors. If this were an exterior product applied to the same bivector, the result would necessarily be zero, due to the result in Equation 32. Note that the *grade* of the algebraic object in Equation 35 has increased from *grade-1* to *grade-2*. This occurs with orthogonal vectors, and other objects of grade lower than that of the dimension of the space. Due to Equation 32, a barrier is erected to the possibility of higher-grade geometric objects in the space that have more than one of the same basis vector (e_i) present — there is no possibility of having a higher-grade object in the space having the form: $e_ie_je_k\ldots e_i$.

Geometric algebra adds an additional product, that yields the sum of the inner and exterior products — and it is known as the *Geometric Product*. Given two arbitrary objects in the geometric algebraic space[16], the geometric product can be written as such.

$$
M_1M_2 = M_1 \cdot M_2 + M_1 \wedge M_2 \quad \textbf{Eqn 37}
$$

There are other ways to express the geometric product, by application of the *grade-selection operator* (also known as the: *grade-projection operator*), which may be outlined later in this document. Properties of the geometric product are also worth mentioning, and are given below. Equation 38 through 40 are the laws of associativity and distributivity.

$$(AB)C = A(BC) \qquad \textbf{\textit{Eqn}}\ \mathbf{38}$$
$$A(B+C) = AB + AC \qquad \textbf{\textit{Eqn}}\ \mathbf{39}$$
$$(B+C)A = BA + CA \qquad \textbf{\textit{Eqn}}\ \mathbf{40}$$

Multiplication of two arbitrary vectors, $A_1 = \sigma_1 + a_1e_1 + b_1e_2 + c_1e_3$ and $A_2 = \sigma_2 + a_2e_1 + b_2e_2 + c_2e_3$, and applying the rules of the algebra would yield the following:

$$A_1A_2 = (\sigma_1 + a_1e_1 + b_1e_2 + c_1e_3)(\sigma_2 + a_2e_1 + b_2e_2 + c_2e_3) \qquad \textbf{\textit{Eqn}}\ \mathbf{41}$$

$$= \sigma_1\sigma_2 + \sigma_1a_2e_1 + \sigma_1b_2e_2 + \sigma_1c_2e_3 + a_1e_1\sigma_2 + a_1e_1a_2e_1 + a_1e_1b_2e_2 + a_1e_1c_2e_3$$
$$+ b_1e_2\sigma_2 + b_1e_2a_2e_1 + b_1e_2\,b_2e_2 + b_1e_2c_2e_3 + c_1e_3\sigma_2 + c_1e_3a_2e_1 + c_1e_3b_2e_2 + c_1e_3c_2e_3$$
$$= \sigma_1\sigma_2 + \sigma_1a_2e_1 + \sigma_1b_2e_2 + \sigma_1c_2e_3 + \sigma_2a_1e_1 + a_1a_2e_1e_1 + a_1b_2e_1e_2 + a_1c_2e_1e_3$$
$$+ \sigma_2b_1e_2 + a_2b_1e_2e_1 + b_1b_2e_2e_2 + b_1c_2e_2e_3 + \sigma_2c_1e_3 + a_2c_1e_3e_1 + b_2c_1e_3e_2 + c_1c_2e_3e_3$$
$$= \sigma_1\sigma_2 + \sigma_1a_2e_1 + \sigma_1b_2e_2 + \sigma_1c_2e_3 + \sigma_2a_1e_1 + a_1a_2 + a_1b_2e_1e_2 + a_1c_2e_1e_3$$
$$+ \sigma_2b_1e_2 - a_2b_1e_1e_2 + b_1b_2 + b_1c_2e_2e_3 + \sigma_2c_1e_3 - a_2c_1e_1e_3 - b_2c_1e_2e_3 + c_1c_2$$

So, after reorganizing: $A_1A_2 = (\sigma_1\sigma_2 + a_1a_2 + b_1b_2 + c_1c_2)$ ***Eqn 42***

$$+ (\sigma_1a_2 + \sigma_2a_1)\,e_1 + (\sigma_1b_2 + \sigma_2b_1)\,e_2 + (\sigma_1c_2 + \sigma_2c_1)\,e_3$$
$$+ (a_1b_2 - a_2b_1)\,e_1e_2 + (a_1c_2 - a_2c_1)\,e_1e_3 + (b_1c_2 - b_2c_1)\,e_2e_3.$$

The three lines of Eqn 42 show the terms of the product organized by grades. The upper line consists of scalars, the middle line of the three single vectors multiplied by scalar terms, and the lower line consists of the three possible bivectors multiplied by scalar terms. Note the symmetries. Although all algebraic steps are shown so that you may follow it, a seasoned practitioner might write out Eqn 42 directly, skipping the intermediates. There is also a matrix version.

In order to continue the discussion of *Conformal Geometric Algebra*, while minimizing the myriad derivations, it is necessary to determine the mapping of vectors from $\mathbb{R}^3$ into $\mathbb{G}^{4,1,0}$. Define a mapping, $M(v)\colon \mathbb{R}^3 \to \mathbb{G}^{4,1,0}$ to be

$$M(v): \mathbb{R}^n \to \mathbb{G}^{4,1,0} \quad = v + \frac{v^2}{2} e_\infty + e_0 \qquad \textbf{Eqn 43}$$

This mapping is required due to the additional dimensions of the space. In Equation 43 the vector v is the ordinary three-component vector as known to students of vector algebra and calculus. An example of how to map a vector,

$v = 1e_1 + 0e_2 + 0e_3$, into this *conformal geometric algebra* space follows:

$$\begin{aligned} M(1e_1 + 0e_2 + 0e_3) &= (1e_1 + 0e_2 + 0e_3) + \frac{(1e_1 + 0e_2 + 0e_3)^2}{2} e_\infty + e_0 \\ M(1e_1) &= (1e_1) + \frac{(1e_1)^2}{2} e_\infty + e_0 \\ M(1e_1) &= 1e_1 + \frac{(1)}{2} e_\infty + e_0 \qquad \boldsymbol{Eqn\ 44} \end{aligned}$$

Equation 44 is the result of mapping the unit vector along the x-axis into $\mathbb{G}^{4,1,0}$ — notice the additional basis vector components, e_0 and e_∞, included in this mapped vector. These new basis vectors are generated by particular summations of e_+ and e_-, described[17] as follows.

$$\begin{aligned} e_0 &= \frac{1}{2}(e_- - e_+) \qquad \boldsymbol{Eqn\ 45} \\ e_\infty &= e_- + e_+ \qquad \boldsymbol{Eqn\ 46} \end{aligned}$$

Subscripts for Equations 45 and 46 signify that e_0 points at the origin, and e_∞ points at infinity. This can be shown by applying a few limits:

$$\begin{aligned} \lim_{|v| \to \infty} \left(\frac{2}{v^2}\right) M(v) &= \lim_{|v| \to \infty} \left(\frac{2}{v^2}\right) v + e_\infty + \left(\frac{2}{v^2}\right) e_0 = e_\infty \qquad \boldsymbol{Eqn\ 47} \\ \lim_{v \to 0} M(v) &= e_0 \qquad \boldsymbol{Eqn\ 48} \end{aligned}$$

Equations 47 and 48 show that as the mapped vector goes to ∞ or 0, respectively, the resultant mapping points at either e_∞ or e_0.

Rotations in Geometric Algebra

Rotations in *Geometric Algebra* are unique when contrasted with those done by applying $SO(3)$ or those done by quaternions. In order to perform a rotation using $SO(3)$ rotation matrices, it is necessary to determine which composition of rotations around the three orthogonal basis vectors corresponds to a rotation about whichever axis is desired (assuming it is not one of the three orthonormal basis vectors). The *quaternions* allow for a substantial improvement over the use of $SO(3)$, by allowing the user to choose an arbitrary vector to rotate around — but the use of *quaternions* is still limited by rotating an arbitrary vector around another.

Geometric Algebra, in contrast to the limitations of the other two methods outlined herein, allows a user to rotate any arbitrary geometric object in the space around an angle inside an arbitrary plane (the same plane orthogonal to the vector that would be chosen if applying a *quaternion* rotation). In order to perform a rotation in *geometric algebra*, it is first necessary to construct a *rotor*. A *Conformal geometric algebra* further extends the usefulness of the *rotor* by allowing translations to be described by this same manner — which can then be composed alongside rotations by the same application of these translational rotors.

A *rotor* is a mathematical device that, when appropriately applied to an object in a *geometric algebra* space, rotates the object it was applied to. In *geometric algebra*, rotations are applied around a bivector – the *grade-2* elements of the space. Bivectors are most simply generated[18] by application of the exterior product to two vectors.

The following discussion should help clarify the creation and application of a rotor to an object in a geometric algebra space. Some of this is necessary, and helps relate the use of a rotor to knowledge that any graduate of a mathematics, physics, or engineering program should have readily available.

According to the reference text, Geometric Algebra for Computer Science, there are a number of ways in which a rotor can be defined in a geometric algebra. Some definitions strictly make use of the resultant bivector left after the geometric product of two unit vectors, and the products resulting by multiplication with other multivectors in the space. By expanding this geometric product, as follows, the exponential form of the rotor can be determined. Let a_1 and b_1 be unit vectors with $a_1, b_1 \in \mathbb{G}^3$

$$a_1 b_1 \quad = a_1 \cdot b_1 + a_1 \wedge b_1 = cos\left(\frac{\theta}{2}\right) - Isin\left(\frac{\theta}{2}\right) \quad \textbf{Eqn 49}$$

Note that, in Equation 49, since unit vectors were used in this geometric product, the coefficient for each trigonometric term is $(+1)$. In the same equation, I represents the unit 2-blade for the $a_1 \wedge b_1$-plane. Careful observers will recognize this equation as *Euler's Identity*, when considering that $I = a_1 \wedge b_1$ and that $I^2 = (a_1 \wedge b_1)(a_1 \wedge b_1) = -(a_1 \wedge b_1)(b_1 \wedge a_1) = -1$

Perhaps the most useful definition of a rotor[19], is that describing the rotor as an exponential of bivectors. In this description, the rotation is described using a bivector to encode the plane in which the rotation takes place as well as the angle through which the object is rotated. In order to construct this rotor, it is necessary to know the angle, θ, through which the rotation is to take place and the bivector, B, inside which that angle is to be swept out. It is important to note here, similarly to rotors constructed from quaternions, that the angle of rotation will be twice the angle desired — since there is a factor of $\frac{1}{2}$ in the rotor.

In geometric algebra, the constructed rotor can be written as:

$$\boldsymbol{R} \quad = e^{\left(-B\frac{\theta}{2}\right)} \quad \boldsymbol{Eqn\ 50}$$

The application of this rotor, to an arbitrary multivector in the geometric algebra space, is done in the same manner as with quaternions — conjugation[20]. As in the application of a quaternion-derived rotor, the rotation is made in the following manner — where the conjugate[21] of the rotor is necessary.

$$\boldsymbol{v}_1^{rot} \quad = \boldsymbol{RvR}^* \quad \boldsymbol{Eqn\ 51}$$

Rotations in Conformal Geometric Algebra

The *Conformal Geometric Algebra* space that is employed by GeoAutonomy at present[22] is $\mathbb{G}^{4,1,0}$ — this algebra has four basis vectors that square to $+1$, one basis vector that squares to -1, and zero basis vectors that square to 0. This is evident by inspection of the provided signature: $4,1,0$. The canonical set of basis vectors from $\mathbb{R}^3$ is extended in $\mathbb{G}^{4,1,0}$ to the following:

$$e_1, e_2, e_3, e_+, e_-$$

The additional basis vectors in $\mathbb{G}^{4,1,0}$ square to, respectively, $+1$ and -1. This addition to the set of basis vectors for the space provides the signature boost that makes $\mathbb{G}^{4,1,0}$ a 5-dimensional space. Equation 47 shows that e_∞ points to infinity in the conformal geometric algebra space. By having this *one-point compactification* with ∞, another unique property of rotors is exhibited.

A translation can be achieved, in a conformal geometric algebra, by *rotating around the "point at infinity"*. By doing so, the object being moved is so far away from the center of rotation, that it becomes in effect, a straight line movement (translation). This is one of the distinct advantages of utilizing a conformal geometric algebra space over an ordinary one. Further descriptions will be provided if required.

Rotation and Translation Footnotes

[1] Aircraft Attitude Control Methods — WIPO Publication Number: WO 2015/180171 A1

[2] e.g. A rotation around the x-axis, when aligned with the front of an object (front of a UAV), corresponds to roll.

[3] A degree of freedom, in this case, refers to the number of independent variables (or parameters) that define the state of a rigid body. As an example, a train has one degree of freedom in its position due to being constrained by the track it travels on. An aircraft (as well as a sailboat), on the other hand, has six degrees of freedom — three describing directions it can translate and three corresponding to attitude.

[4] A mechanical gyroscope — image in the figure found at: https://unmannedengineeriablog.wordpress.com/2016/01/22/ difference- between- gyroscope- and- gymbal/.

[5] Sharing the same center.

[6] Given by Equations 1, 2, and 3.

[7] SO(3) is a nonabelian group, meaning that the order in which rotations are applied makes a difference.

[8] For any code base, be it vehicular control, UAV control system, cell phone screen rotation, etc.

[9] This theorem states that in R^3 any rigid-body displacement in which one point on said rigid-body remains fixed is equivalent to a single rotation through some angle about the corresponding axis running through that point.

[10] Any arbitrary axis is a fine choice.

[11] This representation of the quantities required to perform a rotation is known as the axis-angle representation.

[12] Euler's Formula, as taught in undergraduate math, physics, and engineering courses, is: $e^{i\theta} = \cos(\theta) + i \sin(\theta)$

[13] Both the vectors being rotated and the rotors themselves.

[14] Here, specifically Conformal Geometric Algebra.

[15] Typically, for physical purposes, this is R^3.

[16] The Geometric Product is an operation that can be performed on any object in the space — not just vectors, but any objects that can be part of the canonical multivector of the space.

[17] Definitions for these two vectors vary depending on the author or source. It is not uncommon to see these multiplied by 2, to remove the fractional coefficient, or other seemingly minor changes.

[18] There are other products that generate bivectors — the dual of a vector in $G R^3$

[19] Certainly for our purposes.

[20] Otherwise, and colloquially known as: the sandwich product.

[21] More properly, the reversion.

[22] It is more than likely that future implementations, codebases, and products will employ higher-order algebras as well as the ones discussed here.

References

1. [1] Leo Dorst, Daniel Fontijne, and Stephen Mann. Geometric Algebra for Computer Science. June 2021. isbn: 9780123749420.
2. [2] Elias Riedel Gårding. "Geometric algebra, conformal geometry and the common curves problem". PhD thesis. 2017.
3. [3] Wikipedia. Euler's rotation theorem — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.
   php?title=Euler's%20rotation%20theorem&oldid=1014607746. [Online; accessed 27-May-2021]. 2021.
4. [4] Wikipedia. Quaternion — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=
   Quaternion&oldid=1025276734. [Online; accessed 27-May-2021]. 2021.

viii

Rust Algorithm 1 – density/matrix/src/lib.rs

```
//!
//! File: density/matrix/src/lib.rs
//! Author: Jacob Davisson
//! Last Modified: 13 May 2024
//! Purpose: Provide a library with definitions, functions
//!          trait implementations, etc. for handling
//!          matrix and vector operations.
//! Depends: ndarray::*;
//!          std::fmt::Display;
//!          std::fmt::Write;
//!
#![allow(non_snake_case)]
#![allow(non_camel_case_types)]
#![allow(unused)]
#![allow(dead_code)]
// #![recursion_limit = "65536"]
use ndarray::*;
use std::fmt::Display;
use std::fmt::Write;
use std::marker::Copy;
pub enum AxisSelection {
    I, //Identity
    X, //R_{X}
    Y, //R_{y}
    Z, //R_{z}
```

```
}
impl AxisSelection {
    pub fn new() -> Self {
        AxisSelection::I
    }
    pub fn X() -> Self {
        AxisSelection::X
    }
    pub fn Y() -> Self {
        AxisSelection::Y
    }
    pub fn Z() -> Self {
        AxisSelection::Z
    }

}
#[derive(Clone, Debug)]
pub struct Vector {
    vec: Array1<f32>,
}
impl Vector {
    pub fn new() -> Self {

Vector {

            vec: arr1(&[0.0, 0.0, 0.0]),
        }

}

    pub fn set(&mut self, x: f32, y: f32, z: f32) -> Self {
        Vector {
            vec: arr1(&[x, y, z]),
        }

}

    pub fn fullRotation(&mut self, x: f32, y: f32, z: f32) -> Self {
        Self::new()
    }

}
impl Display for Vector {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self.vec)
    }

}
pub struct RotationMatrix {
    axis: AxisSelection,
    mat: Array2<f32>,
}
impl RotationMatrix {
    pub fn new() -> Self {
        RotationMatrix {
            axis: AxisSelection::I,
            mat: arr2(&[[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]]),
        }
}
    pub fn newRot(&mut self, a: AxisSelection, ang: f32) -> Self {
        match a {
            AxisSelection::I => RotationMatrix::new(),
            AxisSelection::X => RotationMatrix::Rx(ang),
            AxisSelection::Y => RotationMatrix::Ry(ang),
            AxisSelection::Z => RotationMatrix::Rz(ang),
} }
    pub fn Rx(theta: f32) -> Self {
        let t = (theta * std::f32::consts::PI) / 180.0;
        RotationMatrix {
            axis: AxisSelection::X,
            mat: arr2(&[
                [1.0, 0.0, 0.0],
                [0.0, (t).cos(), -(t).sin()],
                [0.0, (t).sin(), (t).cos()],

            ]),
} }
    pub fn Ry(theta: f32) -> Self {
        let t = (theta * std::f32::consts::PI) / 180.0;
        RotationMatrix {
            axis: AxisSelection::Y,
            mat: arr2(&[
                [(t).cos(), 0.0, (t).sin()],
                [0.0, 1.0, 0.0],
                [-(t).sin(), 0.0, (t).cos()],

            ]),
} }
    pub fn Rz(theta: f32) -> Self {
        let t = (theta * std::f32::consts::PI) / 180.0;
        RotationMatrix {
            axis: AxisSelection::Z,
            mat: arr2(&[
                [(t).cos(), -(t).sin(), 0.0],
                [(t).sin(), (t).cos(), 0.0],
                [0.0, 0.0, 1.0],

            ]),

        }

    }
} // END impl RotationMatrix
impl Display for RotationMatrix {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self.mat)
    }
}
impl std::ops::Mul for RotationMatrix {
    type Output = Array<f32, Ix2>;
    fn mul(self, rhs: Self) -> Self::Output {
        self.mat * rhs.mat
    }
}
impl std::ops::Mul<Vector> for RotationMatrix {
    type Output = Vector;
    fn mul(self, rhs: Vector) -> Self::Output {
        let mut v = Vector::new();
        v.vec = self.mat.dot(&rhs.vec);
        v
} }
```

Rust Algorithm 2 – density/ga/src/lib.rs

```
//!
//! File: density/ga/src/lib.rs
//! Author: Jacob Davisson
//! Last Modified: 13 May 2024
//! Purpose: Provide a library with definitions, functions
//!          trait implementations, etc. for handling
//!          Geometric Algebra operations.
//! Depends: ndarray::*;
//!          std::fmt::Display;
//!          std::fmt::Write;
//!
#![allow(non_snake_case)]
#![allow(non_camel_case_types)]
#![allow(unused)]
#![allow(dead_code)]
use std::fmt::{Debug, Display};
use std::ops::Mul;
// No associated VectorPart enumeration,
// as it should be plain to see that
// positionally we have e1, e2, e3...
#[derive(Copy, Clone, Debug)]
pub struct Vector(f32, f32, f32);
impl Vector {
    pub fn new() -> Vector {
        Vector(0.0, 0.0, 0.0)
    }
    pub fn set(&mut self, a: f32, b: f32, c: f32) -> () {
        self.0 = a;
self.1 = b;
self.2 = c; }
    pub fn setnew(a: f32, b: f32, c: f32) -> Vector {
        let mut v: Vector = Vector::new();
        v.set(a, b, c);
        v
}
    pub fn rotate(&self, theta: f32, b: Bivector) -> Vector {
        let t = (theta * std::f32::consts::PI) / 180.0;
        let a: (f32, f32) = (t.cos(), t.sin());
        let b: (f32, f32) = (t.cos(), -t.sin());
        let vv: (f32, f32, f32) = (self.0, self.1, self.2);
        let mut v: Vector = Vector::new();
        match b.1 {
            BivectorParts::e12 => v.set(
                self.0 * a.0.powi(2) + self.0 * a.1.powi(2),
                self.1 * a.0.powi(2) + self.1 * a.1.powi(2),
                self.2 * a.0.powi(2) + self.2 * a.1.powi(2),
            ), // v(cos^2(theta) + sin^2(theta))
        }
v }
}
impl Display for Vector {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "<{}e1, {}e2, {}e3>", self.0, self.1, self.2)
    }
}
#[derive(Copy, Clone, Debug)]
pub enum BivectorParts {
    e12,
e23,
e31, }
#[derive(Copy, Clone, Debug)]
pub struct Bivector(f32, BivectorParts);
impl Bivector {
    pub fn new() -> Bivector {
        Bivector(0.0, BivectorParts::e12)
    }
    pub fn set(&mut self, s: f32, p: BivectorParts) -> () {
        self.0 = s;
self.1 = p; }
    pub fn setnew(s: f32, p: BivectorParts) -> Bivector {
        let mut b: Bivector = Bivector::new();
        b.set(s, p);
        b
} }
impl Display for Bivector {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match self.1 {
            BivectorParts::e12 => write!(f, "{}e12", self.0),
            BivectorParts::e23 => write!(f, "{}e23", self.0),
            BivectorParts::e31 => write!(f, "{}e31", self.0),
} } }
```

Rust Algorithm 3 – density/bin/src/main.rs

```
//!
//! File: density/bin/src/main.rs
//! Author: Jacob Davisson
//! Last Modified: 13 May 2024
//! Purpose: Compiles to binary for executing the code
//!          found in other libraries in this workspace.
//! Depends: ndarray::*;
//!          matrix::*;
//!          std::f32::*;
//!
#![allow(non_snake_case)]
#![allow(non_camel_case_types)]
#![allow(unused)]
#![allow(dead_code)]
use matrix::*;
use ndarray::*;
use std::f32::*;
fn main() {
    let a: RotationMatrix = RotationMatrix::new();
    let Rx: RotationMatrix = RotationMatrix::Rx(45.0);
    let Ry: RotationMatrix = RotationMatrix::Ry(45.0);
    let Rz: RotationMatrix = RotationMatrix::Rz(45.0);
    let mut v: Vector = Vector::new().set(1.0, 0.0, 0.0);
    println!("{}", a);
    println!("Vector v ::\n{}", v);
    println!("Ry ::\n {}", &Ry);
    let newvec = Rx * v;
    println!("Rx * v = {}", newvec);
    println!("Rotate v by Ry(45)::\n{}", Ry * newvec);

}
```

[ix]Screw Mechanics from Hestenes' paper "Old Wine in new Bottles."

Screw theory with geometric algebra enables us to combine the rotational and translational equations of motion for a rigid body into a single equation. The kinematics of a body point

$x = Dx_0D^{-1}$ (64) is completely characterized by the displacement spinor $D = D(t)$, which obeys

imply the invariants:

the kinematical equation

with

$\dot{D} = \frac{1}{2}VD$ (65) $V = -i\omega + ve$, (66)

11

where ω is the angular velocity of the body and we can take v to be its center- of-mass velocity. It follows that

$\dot{x} = V \cdot x$ and $\dot{x} = \omega \times x + v$.

A comomentum P is defined for the body by
$P = MV = iI\omega + mve_* = il + pe_*$. (67)

This defines a generalized "mass tensor" M in terms of the inertia tensor I and the body mass m. According to the transformation equations below, the comomentum is a coscrew.

The coforce or wrench W acting on a rigid body is defined in terms of the torque Γ and net force f by

$W = i\Gamma + fe_*$. (68) The dynamical equation for combined rotational and translational motion then

conservation law for kinetic energy

$K = \frac{1}{2} V \cdot P = \frac{1}{2}(\omega \cdot l + v \cdot p)$. (71) A change of reference frame, including a shift of base point, is expressed by

$x \longrightarrow x' = Ux = UxU^{-1}$. (72) We consider here only the case when the spinor U is constant. Then (72) induces the transformations

$V' = UV$, (73)

$P = UP'$. (74) Thus, the transformation of V is Covariant, while the transformation of P is Contravariant. Their scalar product is the Invariant

$P' \cdot V' = P \cdot V$. (75)

There is much more about all this in [8], [9] and [10], especially applications. For more screw theory, see [11] and [12].

---

x

"Understanding Geometric Algebra" by Kenichi Kanatani, CRC Press 2015. From Section 8.6, Pg 132

Conformal geometry studies *conformal transformations,* for which two definitions exist. In a broad sense, they are transformations that preserve angles made by tangents to curves and surfaces at their intersections; in a narrow sense, they are transformations defined throughout the space, including infinity, that map spheres to spheres. To specifically mean the latter, they are referred to as spherical *conformal transformations* or *Mobius transformations.* Here, we consider conformal transformations in the latter sense that map spheres to spheres and preserve the angles made by their tangent planes at their intersections; a plane is regarded as a sphere of infinite radius. Since the intersection of two spheres is a circle, circles are mapped to circles, and the angles made by their tangent lines at the intersections are preserved; a line is regarded as a circle of infinite radius.

… conformal transformations are defined by versors (products of vectors that have a certain grade) . …..
The set of all Conformal transformations constitutes a group of transformations, includes the following familiar subgroups:
Similarities
Rigid motions
Rotations (note this is a compound reflection)
Reflections
Dilations (and contractions)
Translations
The identity

… compositions of rigid motions and reflections are called *isometries* or *Euclid transformations*. They themselves constitute a closed subgroup of conformal transformations that preserve length; they include translations, rotations and the identity as its subgroups.

From Section 8.7, p141-142

Conformal mappings are mappings that preserve angles between tangents. In 2D they are given by an analytical (or regular or holomorphic) function over a domain of the complex plane, familiar to anyone taking a course in Complex Variables.

xi

Old Wine in New Bottles: A new algebraic framework for computational geometry David Hestenes

Introduction

My purpose in this chapter is to introduce you to a powerful new algebraic model for Euclidean space with all sorts of applications to computer-aided geometry, robotics, computer vision and the like. A detailed description and analysis of the model is soon to be published elsewhere [1], so I can concentrate on highlights here, although with a slightly different formulation that I find more convenient for applications. Also, I can assume that this audience is familiar with Geometric Algebra, so we can proceed rapidly without belaboring the basics.

https://www.researchgate.net/publication/270492105_Old_wine_in_new_bottles

[xii] **Relational is not enough**

https://frankzliu.com/blog/a-gentle-introduction-to-vector-databases

Data is everywhere. In the early days of the internet, data was mostly structured, and could easily be stored and managed in relational databases. Take, for example, a book database:

ISBN	Year	Name	**Author**
0767908171	2003	A Short History of Nearly Everything	Bill Bryson
039516611X	1962	Silent Spring	Rachel Carson
0374332657	1998	Holes	Louis Sachar

...

Storing and searching across table-based data such as the one shown above is exactly what relational databases were designed to do. In the example above, each row within the database represents a particular book, while the columns correspond to a particular category of information. When a user looks up book(s) through an online service, they can do so through any of the column names present within the database. For example, querying over all results where the author name is Bill Bryson returns all of Bryson's books.

As the internet grew and evolved, unstructured data (magazine articles, shared photos, short videos, etc.) became increasingly common. Unlike structured data, there is no easy way to store the contents of unstructured data within a relational database. Imagine, for example, trying to search for similar shoes given a collection of shoe pictures from various angles; this would be impossible in a relational database since understanding shoe style, size, color, etc… purely from the image's raw pixel values is impossible.

This brings us to vector databases. The increasing ubiquity of unstructured data has led to a steady rise in the use of machine learning models trained to understand such data. `word2vec`, a natural language processing (NLP) algorithm which uses a neural network to learn word associations, is a well-known early example of this. The `word2vec` model is capable of turning single words (in a variety of languages, not just English) into a list of floating point values, or vectors. Due to the way models is trained, vectors which are close to each other represent words which are similar to each other, hence the term *embedding vectors*. We'll get into a bit more detail (with code!) in the next section.

Armed with this knowledge, it's now clear what vector databases are used for: searching across images, video, text, audio, and other forms of unstructured data via their *content* rather than keywords or tags (which are often input manually by users or curators). When combined with powerful machine learning models, vector databases have the capability of revolutionizing semantic search and recommendation systems.

Data UID[1]	**Vector representation**
00000000	[-0.31, 0.53, -0.18, …, -0.16, -0.38]
00000001	[0.58, 0.25, 0.61, …, -0.03, -0.31]
00000002	[-0.07, -0.53, -0.02, …, -0.61, 0.59]
…	

In the upcoming sections, I'll share some information about why embedding vectors can be used to represent unstructured data, go over algorithms for indexing and searching across vector spaces, and present some key features a modern vector database must implement.

`x2vec`: A new way to understand data

The idea of turning a piece of unstructured data into a list of numerical values is nothing new[2]. As deep learning gained steam in both academic and industry circles, new ways to represent text, audio, and images came to be. A common component of all these representations is their use of embedding vectors generated by trained deep neural networks. Going back to the example of `word2vec`, we can see that the generated embeddings contain significant semantic information.

[xiii] From Hildenbrand:
Why should you use a 5-dimensional geometric algebra if your problem is from the 3D real world?

One reason is that problems can often be formulated more easily and intuitively in a higher number of dimensions. One advantage of CGA, for instance, is that points, spheres and planes are easily represented as vectors (linear combination of blades of grade 1).

Geometric Operations can be expressed easily in Geometric Algebra.

Duality: Geometric Algebra allows for division by geometric objects. This can sometimes transform a difficult problem into a simpler one.