May the Forque Be with You

Dynamics in PGA

Leo Dorst & Steven De Keninck
University of Amsterdam
Amsterdam, The Netherlands

Version 2.1 – February 17, 2022

Contents

Physics in the Plane-Based Geometric Algebra PGA

In our introductory guide [13], PGA was treated as a branch of geometry in the usual mathematical sense: geometric planes, lines or points in space without necessary physical reality. In this new guide, we will use those algebraic elements to model physical objects and their motions under forces.

This extension of PGA to physics is rather specific in its techniques, and new intermediate concepts become the basic algebraic blocks for modelling. There will be no new algebra and we refer for the basics to [13], but we will make that known algebra correspond with real-world physics. We will find that we can improve the classical way of computing in physics considerably, in three ways:

- PGA integrates geometric primitives and the motion operators acting on them;
- PGA requires only half the number of physical variables;
- PGA produces algorithms that are agnostic to the dimension of physical space.

The combination of these properties is especially handy, and leads to compact and robust code, notably not requiring a distinction between 2D and 3D in its primitives or functions. While this powerful integration is new, the compact PGA description of motion operators is very reminiscent of other frameworks such as *Screw Theory* and *Spatial Vector Algebra* (both with 19th century provenance). Somehow, those missed a natural embedding in the fully unified framework of PGA; we will see how close they came.

So, this guide helps you make a fresh start in physical modelling. We show how to model velocity, momentum and forces in an integrated manner that no longer artificially separates the translational from the rotational (and requires no prior choice of space dimensions). The strange and perhaps cringeworthy '*forque*' in the title is an embodiment of that: we will define a dual bivector that encapsulates all of *for*ce and tor*que* combined, and we show the advantages of that in description and computation.

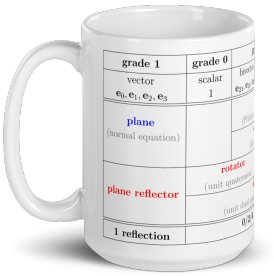
In two additionally included Appendices A and B (which may be consulted from the main text by linked pointers), we pull the integrated PGA elements apart into their classical constituents, just to show that PGA really provides the same description of Nature as classical physics. At the same time, doing so exposes explicitly the implicit non-computational (because non-algebraic) rules that one needed to learn to apply the classical formalism.

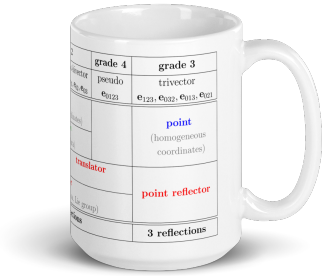
To show that the PGA remodelling is indeed effective, we provide programs using it, for you to play with. These will have the amazing property that they work in any dimension; that also means that they can be specified in the lowest dimension in which the problem makes sense. For instance, we show how the full 3D motion of a massive cube on a spring in gravity is computed by a program that literally encodes the whole situation in 1D. When this 1D program is given 3D initial conditions, it employs them to fill the available 3D space with the appropriate Newtonian motion.

Our focus audience for this text is workers in the computer sciences (notably computer graphics and robotics), who will enjoy those explicit computational aspects; but in Appendix B we also develop some pen-and-paper techniques to solve the equations exactly (in the simpler cases) to show the convenience of PGA to classical physics and to verify the precision of the numerical methods.

If this guide makes you want to know more, you can always find additional tutorial material, lectures, software and even PGA merch at our home site `https://bivector.net`.

Leo Dorst & Steven De Keninck
Amsterdam & Bornem, 2022





Chapter 1

Elements of Physical Geometry

When we use PGA in physics, our elements may have acquire mass, vectors may represent velocities or momenta, bivectors may be angular momenta or forces. Processing them requires some care in their units and dimensional properties (here we use 'dimension' in the sense of 'physical measurement dimension'). In this section we treat some of those preliminaries on dimensionality and representation of the basic geometrical elements, before we tackle the physics.

1.1 Planes are Vectors

When one has chosen as spatial reference an (arbitrary) origin at a point O, the simplest representable plane is a plane p through that origin. It is represented in PGA as a purely Euclidean vector $\mathbf{n}$ (refer to [13] if this is new to you). In more classical applied linear algebra, we would call $\mathbf{n}$ the *normal vector* of the plane p; but in PGA, $\mathbf{n}$ simply *is* the plane p through the origin.

$$p = \mathbf{n}.$$

This change of perspective is important: a normal vector exists only at the origin, a plane may be anywhere; we should not let our representation get in the way of the reality we want to describe, so we will move the plane very soon!

We commonly normalize a plane p to have $p^2 = 1$, so for this origin plane $\mathbf{n}^2 = 1$. Such a normalized plane is a dimensionless mathematical element. We would denote a weighted plane at the origin as $\alpha\mathbf{n}$. In 3D, this plane is the dual (not the reciprocal!) of a Euclidean area 2-blade, and so α should have the dimension measure of [area]. A density, with dimension measure of 1/[area], can then be represented by the reciprocal vector p^{-1}. (In d dimensions, the hyperplane $\alpha\mathbf{n}$ would give α the dimension of [length]^d.)

Planes moved outside the origin require additional specification of their distance δ to the (arbitrary) origin O. In homogeneous coordinates, this would be done by extending the parameters to $[\mathbf{n}, -\delta]$. In PGA, we explicitly denote the basis vector $\boldsymbol{\epsilon}$ of the extra dimension, and write $p = \mathbf{n} - \delta\boldsymbol{\epsilon}$. This $\boldsymbol{\epsilon}$ anti-commutes with the Euclidean basis vectors, and satisfies $\boldsymbol{\epsilon}^2 = 0$, see [13].[1]

[1] We have changed the extra homogeneous basis vector relative to [13] (versions less than 2.0); our current $\boldsymbol{\epsilon}$ is $-e$ relative to the e used in that source (which would denote the plane as $p = \mathbf{n} + \delta\, e$). The new

This representation $p = \mathbf{n} - \delta\boldsymbol{\epsilon}$ is consistent with translating a normalized plane away from the origin to pass through a point at location $\mathbf{q}$. Indeed, applying the translation versor $T_\mathbf{t} \equiv \exp(-\boldsymbol{\epsilon}\mathbf{t}/2) = 1 - \boldsymbol{\epsilon}\mathbf{t}/2$ to $\mathbf{n}$ gives:

$$p = (1 - \boldsymbol{\epsilon}\mathbf{q}/2)\,\mathbf{n}\,(1 + \boldsymbol{\epsilon}\mathbf{q}/2) = \mathbf{n} - (\mathbf{q}\cdot\mathbf{n})\,\boldsymbol{\epsilon} = \mathbf{n} - \delta\,\boldsymbol{\epsilon},$$

where $\delta = \mathbf{q}\cdot\mathbf{n}$. Since the normal vector $\mathbf{n}$ has been normalized, the first term is a dimensionless quantity; but then so should the second term be. Since δ has the dimension measure of [length], we find that the extra representational vector $\boldsymbol{\epsilon}$ should be considered to have the dimension [1/length]. Geometrically, $\boldsymbol{\epsilon}$ is the representation of the 'plane at infinity' (we called it the 'vanishing plane' in [13]); it apparently contains a length scale for the space. By contrast, the Euclidean unit vectors $\mathbf{e}_i$, which are a basis to specify any $\mathbf{n}$, represent coordinate planes through the origin and are dimensionless mathematical quantities.

We see the dimensional nature of $\boldsymbol{\epsilon}$ as [1/length] also in the motors (aka versors or rotors or spinors) which act as motion operators. The translation motor $\exp(-\boldsymbol{\epsilon}\mathbf{q}/2)$ should have a dimensionless argument in its exponent; since the translation parameter vector $\mathbf{q}$ has the dimension measure of [length], this also confirms the [1/length] dimensionial property of $\boldsymbol{\epsilon}$. When we introduce a *velocity* $\mathbf{v}$ of translation, with dimension measure of [length/time], we can rewrite the exponent as:

$$\exp(-\boldsymbol{\epsilon}\mathbf{v}\,t/2),$$

with t the amount of [time] that was used by the motor to obtain the required translation.

1.2 Points are Trivectors

A point in 3D PGA is represented as a trivector, since it can be formed as the intersection of three planes (see [13], in d-dimensional PGA, it is a d-vector) by means of the meet operation, represented algebraically as the wedge product.

1.2.1 Representing a Finite Point

The normalized point at the (arbitrary) origin O in 3D PGA may be formed by the intersection of the three coordinate planes $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$, and is therefore represented by the purely Euclidean trivector:

$$O = \mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \mathbf{e}_3 = \mathbf{e}_1\,\mathbf{e}_2\,\mathbf{e}_3 \equiv \mathbf{I}_3,$$

with $\mathbf{I}_3$ denoting the normalized Euclidean pseudoscalar in a coordinate-free manner. We prefer to use the notation O since we aim at coordinate-free dimension-agnostic geometry, but may sometimes use $\mathbf{I}_3$ when we do a coordinate calculation in 3D. The normalized geometrical point at the origin is thus a dimensionless quantity; and since all points can be made from it through translation, so are all points.

practice corresponds much more nicely with programs written in homogeneous coordinates and prevents many headaches in sign. When used in programs, it is customary to have $\boldsymbol{\epsilon}$ as the zeroth basis vector $\boldsymbol{\epsilon} = \mathbf{e}_0$, allowing index notations like $\mathbf{e}_{01}$ for $\boldsymbol{\epsilon}\,\mathbf{e}_1$.

When we move the point to a location $\mathbf{q}$, the trivector Q representing it is found by applying a translation motor to O. This gives

$$Q = (1 - \boldsymbol{\epsilon}\,\mathbf{q}/2)\,O\,(1 + \boldsymbol{\epsilon}\,\mathbf{q}/2) = (1 - \boldsymbol{\epsilon}\,\mathbf{q})\,O = O + \mathbf{q}\,\mathcal{I}, \tag{1.1}$$

where in the final rewriting we employ the *pseudoscalar* of d-dimensional PGA, defined as

$$\text{PGA pseudoscalar:} \quad \mathcal{I}_d \equiv \boldsymbol{\epsilon}\,O = \boldsymbol{\epsilon}\,\mathbf{I}_d,$$

and is a multivector of grade $(d+1)$ (we indexed it with the space it represents rather than its grade). We will often write just $\mathcal{I}$ for $\mathcal{I}_d$ if the dimension d is clear from the context and/or does not feature in the computations. In algorithms, we will denote the basis vector $\boldsymbol{\epsilon}$ by index 0, as $\mathbf{e}_0$, to allow shorthand like $\mathcal{I}_3 = \mathbf{e}_{0123}$. Note that the PGA pseudoscalar $\mathcal{I}$ contains $\boldsymbol{\epsilon}$, and therefore has the measure of [1/length]; in eq.(1.1), the combination $\mathbf{q}\mathcal{I}$ is dimensionless, so the choice of pseudoscalar effectively fixes the units for the length of $\mathbf{q}$, viewed as a displacement.

Alternatively, you can construct the point at location $\mathbf{q} = q_1\mathbf{e}_1 + q_2\mathbf{e}_2 + q_3\mathbf{e}_3$ as the intersection of the planes $\mathbf{e}_1 - q_1\boldsymbol{\epsilon}$, $\mathbf{e}_2 - q_2\boldsymbol{\epsilon}$ and $\mathbf{e}_3 - q_3\boldsymbol{\epsilon}$, which gives the same result:

$$\begin{aligned}
&(\mathbf{e}_1 - q_1\boldsymbol{\epsilon}) \wedge (\mathbf{e}_2 - q_2\boldsymbol{\epsilon}) \wedge (\mathbf{e}_3 - q_3\boldsymbol{\epsilon}) \\
&\quad = \mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \mathbf{e}_3 - q_1\,\boldsymbol{\epsilon} \wedge \mathbf{e}_2 \wedge \mathbf{e}_3 - q_2\,\boldsymbol{\epsilon} \wedge \mathbf{e}_3 \wedge \mathbf{e}_1 - q_3\,\boldsymbol{\epsilon} \wedge \mathbf{e}_1 \wedge \mathbf{e}_2) \\
&\quad = O - \boldsymbol{\epsilon}\,(q_1\,\mathbf{e}_2 \wedge \mathbf{e}_3 + q_2\,\mathbf{e}_3 \wedge \mathbf{e}_1 + q_3\,\mathbf{e}_1 \wedge \mathbf{e}_2) \\
&\quad = O - \boldsymbol{\epsilon}\,\mathbf{q}\,\mathbf{I}_3 \\
&\quad = O + \mathbf{q}\,\mathcal{I}_3.
\end{aligned}$$

The additive representation $O + \mathbf{q}\,\mathcal{I}$ shows that the point could also be constructed by adding a properly scaled 'vector direction' $\mathbf{q}\mathcal{I}$ (see below) to the point at the origin (but that method generalizes less directly to non-point elements).

These expressions represent the point Q relative to the origin O, which was arbitrary. Yet, perhaps surprisingly, the actual value of the trivector Q is independent of the choice of origin. For when we introduce as our preferred origin an alternative location C with location vector $\mathbf{c}$, so that $C = O + \mathbf{c}\,\mathcal{I}$, we can rewrite Q as:

$$\begin{aligned}
Q &= O + \mathbf{q}\,\mathcal{I} \\
&= O + (\mathbf{c} + (\mathbf{q} - \mathbf{c}))\,\mathcal{I} \\
&= C + (\mathbf{q} - \mathbf{c})\,\mathcal{I} \\
&\equiv C + \mathbf{r}\,\mathcal{I},
\end{aligned}$$

We emphasize that the algebraic value of the point trivector Q is the same, but it is now parametrized from C by the relative position vector $\mathbf{r} \equiv \mathbf{q} - \mathbf{c}$, as it should. This is how the unchanged geometrical point Q looks if you were placing yourself at location C: it is at relative location $\mathbf{r}$. It has the same split form of 'point plus relative location times pseudoscalar' as looking at it from the origin, or from any other point.

We will call this rewriting of the algebraic element Q into terms involving a point C and 'something more' a *point split*. This 'reparametrization without changing the element' is

a useful feature of PGA that the classical vector representation of points does not provide. The commonly used 'homogeneous coordinates' for a point share the same origin-independent property, but it is good to realize this explicitly, since their emphasis on coordinate representations tends to hide it. We also observe that the $(d+1)$-dimensional pseudoscalar $\mathcal{I}_d$ satisfies

$$\mathcal{I}_d \equiv \boldsymbol{\epsilon}\,\mathbf{I}_d = \boldsymbol{\epsilon}\,O = \boldsymbol{\epsilon}\,Q = \boldsymbol{\epsilon}\,C,$$

so it can be written in terms of any normalized point. It is thus independent of the choice of origin, as it should be, since it is an algebraic property of the whole space.

All finite points in the above representation are normalized: they satisfy $O^2 = Q^2 = C^2 = \mathbf{I}_d^2 = (-1)^{d(d-1)/2}$. Hence for 2D and 3D we get $Q^2 = -1$ as the square of a normalized point.

1.2.2 Representing a Vanishing Point

A vanishing point $V_{\mathbf{u}}$ in direction $\mathbf{u}$ is represented as[2]

$$V_{\mathbf{u}} = \mathbf{u}\,\mathcal{I}_d.$$

Comparing to eq.(1.1), a direction is thus 'a point without an O part'; you can read the $\mathcal{I}_d$ as 'ideal', if you wish, since in projective geometry such points would be called *ideal points*. The multiplicative occurrence of the pseudoscalar shows that a vanishing point is 'purely ideal': it has no finite positional aspects.

1.2.3 Massive Points

We repeat that in PGA, a geometrical point Q at location $\mathbf{q}$ is represented by the trivector

$$Q = O + \mathbf{q}\,\mathcal{I}_d = \mathbf{e}_1\,\mathbf{e}_2\,\mathbf{e}_3 - q_1\,\boldsymbol{\epsilon}\,\mathbf{e}_2\,\mathbf{e}_3 - q_2\,\boldsymbol{\epsilon}\,\mathbf{e}_3\,\mathbf{e}_1 - q_3\,\boldsymbol{\epsilon}\,\mathbf{e}_1\,\mathbf{e}_2.$$

When written out in 3D PGA, the point thus has coordinates $(1, q_1, q_2, q_3)$ on the basis $\big(\mathbf{I}_3,\ \mathbf{e}_1\mathcal{I}_3,\ \mathbf{e}_2\mathcal{I}_3,\ \mathbf{e}_3\mathcal{I}_3\big) = \big(\mathbf{e}_{123},\ \mathbf{e}_{032},\ \mathbf{e}_{013},\ \mathbf{e}_{021}\big)$ (note the swapping of some of the Euclidean indices). So if that basis is implicitly understood, this coordinate representation of a point is in practice just like homogeneous coordinates. The same clearly holds in d-dimensional PGA relating to d-dimensional homogeneous coordinates. Note that the coefficients q_i with dimension measure of [length] are used on basis 3-blades with $\boldsymbol{\epsilon}$ of dimension measure [1/length], and the dimensionless $\mathbf{e}_i$ factors. So indeed, a geometrical point is a dimensionless quantity.

A physical point has a mass, which we will incorporate as the algebraic weight factor of the d-vector:

$$\text{Point } Q \text{ with mass } m\text{:} \quad mQ = m\,(O + \mathbf{q}\,\mathcal{I}).$$

We can now consider the dimension measure of the physical point trivector to be [mass]. The Euclidean unit pseudoscalar O is still dimensionless, and so is the geometric point Q. At the usual academic risk of sounding pedantic, we should perhaps get into the habit of denoting a mass of 1 unit at Q as $1\,Q$ (or $1.00\,Q$), rather than as Q, to denote that this element no longer represents merely the purely algebraic element Q. Even better, write it like Q kg (or your own preferred non-international mass unit, such as 'lb').

[2]This may appear to differ in a sign from early versions of [13], but that text uses another pseudoscalar based on $e \equiv -\boldsymbol{\epsilon}$.

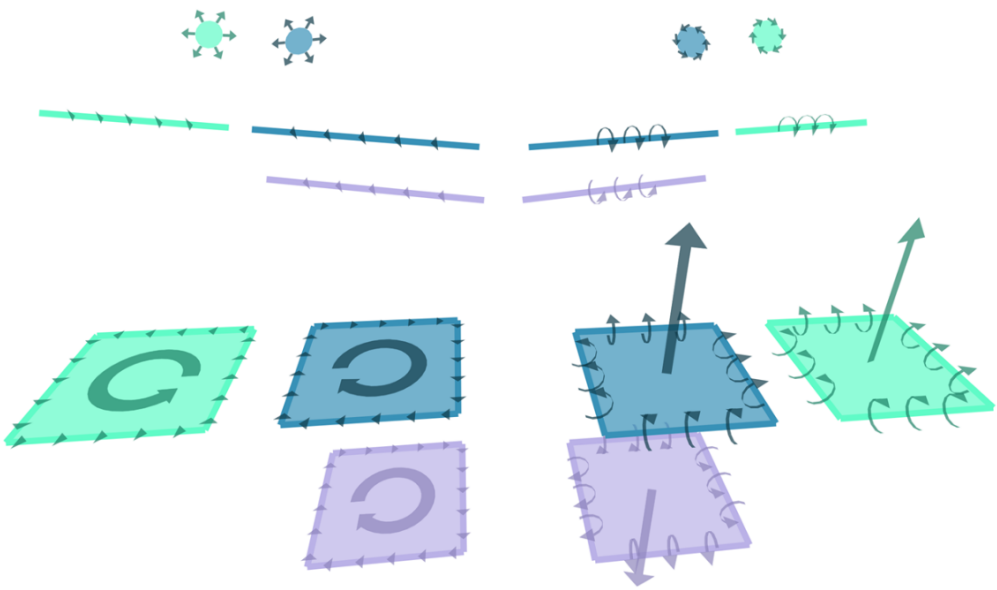


Figure 1.1: *Objects of 3D PGA and their two possible orientation assignments: internal and external. Orientation type shows in the effect of reflections. Blue: oriented element; green: orthogonal reflection; purple: parallel reflection (orthogonal to exactly one factor). Left half: join-based, right half: meet-based.*

1.3 Two Types of 3D Lines

All geometric elements of PGA come in two types of orientations: internal and external, see Figure 1.1. For a plane, the *normal vector* is an *external* way of denoting its orientation, useful for discriminating its sides. By contrast, a *handedness symbol within* the plane is an *internal* orientation. The two types of planes reflect differently; an externally oriented mirror reflected in itself changes sign, an internally oriented mirror does not.

Similarly, there are two types of lines: the ones you move along (internally oriented), and the ones you move around (external orientation). In 3D, there are thus two ways to construct lines: as the intersection of two planes, and as the connection of two points. They both lead to a bivector representation, one dual to the other. In other dimensions, the situation is less confusing (since the grades of the types then differ), and therefore a slight generalization, even if just to 2D, helps inform the 3D subtleties. So let us look beyond 3D.

1.3.1 Hyperlines as Meet of Planes: 2-vectors

The outer product of two planes $\mathbf{n}_1$ and $\mathbf{n}_2$ at the origin in 3D produces by intersection the element

$$L = \mathbf{n}_1 \wedge \mathbf{n}_2. \tag{1.2}$$

Geometrically in 3D, this is a line, which we might call the *meet line* of the planes. But in 2D the intersection of two 2D hyperplanes (there better known as 'lines') produces a point. When

talking across dimensions, we prefer to use the term *hyperline* for the geometric semantics of a 2-blade.

In 3D, such a 2-blade meet line can be used in an exponential to generate the motion that keeps L invariant: a rotation if both planes are non-ideal, a translation if either is ideal or the two planes are parallel (see [13]). In 2D, the exponentiation of the meet hyperline (which there is better known as a 'point') similarly generates a rotation (when the point is finite) or a translation (when the point is ideal). We could borrow the commonly used term 'axis' to denote this usage of the hyperlines – if you are willing to see a planar point as a 2D rotation axis. However, in dynamics, there are multiple axes (for the body shape, the rotational motion, or the angular momentum), so syntax and semantics become confused. For this tutorial text, the term 'hyperline' is used as the factual term for the geometric element; that also serves as a useful reminder that PGA is essentially dimension-agnostic.

1.3.2 Lines as the Join of Points: $(d-1)$-vectors

Points are the intersection of d hyperplanes, and are therefore d-vectors in the $(d+1)$-dimensional PGA of our d-dimensional space. The line connecting two points is represented by their *join*, which is a $(d-1)$-vector (see [13]). Therefore we should use $(d-1)$-vectors to represent lines when those are viewed as being swept out by points. In 3D PGA, these are of course bivectors (just like the meet lines were bivectors), and they are sometimes called 'dual lines' if the meet lines of type eq.(1.2) are seen as 'direct lines'. But others use the terms conversely. In this text we will use the term '*(join) line*' for the line representation as a $(d-1)$-vector, producible by joining two points in d-dimensional space.

A natural form of the join line L_o through the origin O, in a direction denoted by the Euclidean vector $\mathbf{u}$ is then:

$$\text{join line:}\quad L_o = O \vee (O + \mathbf{u}\mathcal{I}) = O \vee (\mathbf{u}\mathcal{I}) = \cdots = \mathbf{u} \cdot O, \tag{1.3}$$

with $O = \mathbf{I}_d$ the Euclidean pseudoscalar, which represents the point at the origin. The details of the computation may be found in eq.(C.7), using techniques from [13].[3] Such a join line is normalized when its direction vector $\mathbf{u}$ is a unit vector; then $L\widetilde{L} = \mathbf{u}^2 = 1$.

Physically moving the origin join line $L_o = \mathbf{u} \cdot O$, to pass instead through a point Q at location $\mathbf{q}$ gives the join line as a $(d-1)$-blade of the form

$$\begin{aligned} L &= (1 - \boldsymbol{\epsilon}\mathbf{q}/2)\,(\mathbf{u} \cdot O)\,(1 + \boldsymbol{\epsilon}\mathbf{q}/2) \\ &= \big(\mathbf{u} - \boldsymbol{\epsilon}\mathbf{q}\mathbf{u}O + \mathbf{u}O\boldsymbol{\epsilon}\mathbf{q}\big) \\ &= \big(\mathbf{u} + (\mathbf{q} \wedge \mathbf{u})\boldsymbol{\epsilon}\big)\,O \\ &= \mathbf{u} \cdot O + (\mathbf{u} \wedge \mathbf{q})\,\mathcal{I} \\ &= \mathbf{u} \cdot (O + \mathbf{q}\,\mathcal{I}) \qquad (1.4) \\ &= \mathbf{u} \cdot Q. \qquad (1.5) \end{aligned}$$

[3] For readers of [13] lower than version 2.0, we have now decided to redefine the join with a swapped order relative to that source, namely as $A \vee B = (B\mathcal{I}^r \wedge A\mathcal{I}^r)\,\mathcal{I} = \star^{-1}(\star A \wedge \star B)$, to correspond better to existing homogeneous coordinate software. This also returns the Common Factor Axiom to the order it has in [3]: $(A \wedge B) \vee (B \wedge C) = (A \wedge B \wedge C) \vee B$.

The various alternative ways of writing, as geometric product, sum of terms, or dot product are each handy in different contexts. The expression $\mathbf{u} \cdot Q$ is an especially direct and memorable way to represent this (join) line through Q in direction $\mathbf{u}$. Note that while for the point at the origin $\mathbf{u} \cdot O = \mathbf{u}\, O$ (so that the dot product can be replaced by a geometric product), the same does not hold for other points like Q. Therefore it is a good habit to always write a (join) line in the dot product form, even at O.

1.3.3 Hyperline Units

It is natural to interpret the weight of the intersection L of two hyperplanes mathematically as an 'intersection strength', ranging from 1 for orthogonal intersection of two normalized hyperplanes $\mathbf{n}_1$ and $\mathbf{n}_2$, to 0 for parallel hyperplanes, to -1 for orthogonal intersection 'in the opposite order'. In 3D, meet lines of the form eq.(1.2) thus derive a measure of the numerical stability relative to noise in the planes: a meet line with weight close to 0 is ill determined in its parameters of location and direction. If the planes were weighted, the resulting line weight is proportional to the product of the plane weights. In 2D, the meet of two lines (2D hyperplanes) produces a 'weighted point'. Here too the weight is not physical mass, but rather a numerical measure of the signed orthogonality of intersection (it is the sine of the angle between normalized lines).

But if we view the 3D meet line as denoting a motion bivector (rotational or translational), we may want to interpret its weight as a measure of the velocity (angular or linear). For a normalized line with $L\widetilde{L} = 1$, we can write the corresponding rotation motor around that line in the form

$$\exp(-L\omega t/2),$$

and then ωt is the amount of rotation in [radians], and ω is the angular velocity in [radians]/[time]. So the angular velocity line would be naturally represented as the weighted line $L\omega$, with units [radians]/[time].

It thus appears we should separate geometry and kinematics: even if we have two 3D planes that determine a rotation line, we should intersect the planes, normalize the meet line to make it dimensionless, and then determine a rotation rate for it before we construct the motor. (And similarly in 2D, where two lines intersect in a weighted point, the center of rotation.)[4]

1.3.4 (Join) Line Units

Lines (as opposed to hyperlines) are made from the connection of points, and their canonical form $\mathbf{u} \cdot Q$ indicates clearly that they have as their unit the measure of separation given by $\mathbf{u}$. If $\mathbf{u}$ is merely a distance, the join line has the measure of [length]; if $\mathbf{u}$ is a velocity, then the join line has the measure [length/time]. If the point Q is a physical mass point, the unit would be that of momentum: [mass $\times$ length/time] – and we will see that indeed, join lines model momentum.

[4]Interestingly, sometimes we can encode the geometry to provide the kinematics quite naturally, as in the tricycle example of [13]. There the intersection strength parameterized the speed of the motion in a geometrically sensible manner: it naturally trades off translational and rotational aspects, and a velocity rate results that generates a cardioid as fixed-time endpoints, see Figure 1.2.

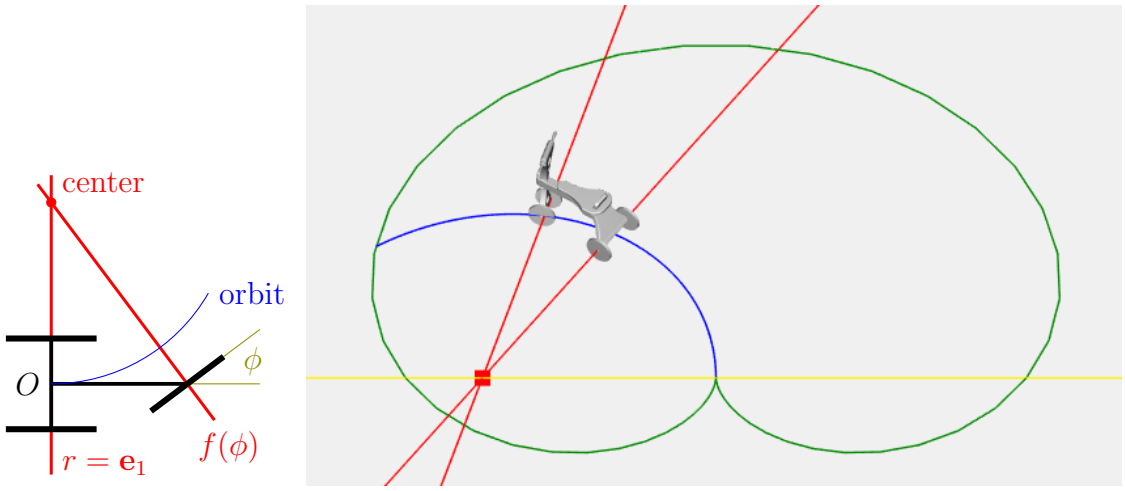


Figure 1.2: *From [13]: Geometry and orbits of a tricycle. When moved one time-unit under the motor* $M_\phi = \exp(r \wedge f(\phi)/2)$*, the endpoints* $M_\phi O \widetilde{M}_\phi$*, for various steering angles* ϕ*, form a cardioid.*

In PGA, (join) lines behave somewhat like tangent vectors, they denote the classical concept of 'a velocity at a point'. However, at any point Q on the join line $\mathbf{u} \cdot Q$, this quantity has the same value; so there is a translational invariance to join lines which we usually do not assign to tangent vectors.

1.3.5 Adding and Splitting Lines or Hyperlines

The element $\mathbf{u} \cdot Q$, split as $\mathbf{u} \cdot Q = \mathbf{u} \cdot (O + \mathbf{q}\mathcal{I})$ represents a join line in terms of its classical geometric parameters of direction $\mathbf{u}$ and relative location $\mathbf{q}$ to the origin (or any other point). A join line is of grade $(d-1)$, and in any dimension dual to a *2-blade* (i.e., the outer product of two vectors).

But we will also need the sums of join lines. In general, that is not the dual of a 2-blade, but the dual of a *bivector* (i.e., an element of grade 2 that cannot necessarily be written as the outer product of two vectors). That distinction between 2-blade and 2-vector, and hence between a $(d-1)$-blade and a $(d-1)$-vector, is relevant in a representational space of more than 3 dimensions; so for 3D PGA and beyond.

Similarly, sums of meet lines occur naturally when we attempt to write a product of motors as the exponential of a single bivector (see [13]), so in the characterization of general Euclidean motions. They are usually 2-vectors, not simply 2-blades.

Let us therefore consider the structure of bivectors and their duals a bit more closely. There are three handy ways of partitioning a PGA bivector B or dual bivector B_* in calculations:

$$\text{Euclidean split bivector:} \quad B = \mathbf{B} + \boldsymbol{\epsilon}\mathbf{v} \tag{1.6}$$

$$\text{Screw split bivector:} \quad B = \omega\, L + \nu\, L\mathcal{I} \tag{1.7}$$

$$\text{Point split at } Q \text{ of dual bivector:} \quad B_* = \mathbf{p} \cdot Q + \mathbf{R}\mathcal{I}, \tag{1.8}$$

where Q is a point d-vector (of the form $Q = O + \mathbf{q}\mathcal{I}$), $\mathbf{B}$ and $\mathbf{R}$ are Euclidean bivectors, L is a 2-blade (in 3D representing a meet line), while $\omega \geq 0$, and ν can have any sign. Let us discuss their uses.

- *Euclidean split* $B = \mathbf{B} + \boldsymbol{\epsilon}\mathbf{v}$.
  Eq.(1.6) distinguishes the Euclidean 2-blade part $\mathbf{B}$ and the 'ideal' part $\boldsymbol{\epsilon}\mathbf{v}$ involving a Euclidean vector $\mathbf{v}$. Since $\boldsymbol{\epsilon}^2 = 0$, the split eq.(1.6) is useful in calculations where one needs to eliminate terms containing $\boldsymbol{\epsilon}$, often leading to rapid cancellation.

  However, beware: this split is *not equivariant* - the Euclidean part of a moved element is not the moved Euclidean part. Therefore, this is not a truly geometrical split. We will employ it mostly to show the correspondence of PGA expressions for dynamics with their classical counterparts (which are surprisingly often origin-dependent), in Appendices A and B. It is a practice to be avoided from now on...

- *Screw split* $B = \omega\, L + \nu\, L\mathcal{I}$.
  The screw split is basically the PGA version of Chasles' Theorem in 3D, and could be called a *Chasles split* (see [13] where it is called the *bivector split*). In this form it is valid only for 3D PGA, in general dimensions there are not just these two terms. This split is useful when the bivector B is viewed as a screw generator, and then splits it as a weighted sum of a normalized meet line 2-blade L (so that $L^2 = -1$) and a directional part $L\mathcal{I}$ (with $(L\mathcal{I})^2 = 0$), generating a translation along the line L. In contrast to the Euclidean split, the constituents L and $L\mathcal{I}$ commute.

  The screw split is equivariant ([22] calls it the 'invariant split'), and therefore has objective geometrical meaning. It is convenient when the bivector B is used in an exponential, since it permits a factorization of the resulting motor into $V = \exp(B) = \exp(\omega\, L)\, \exp(\nu\, L\mathcal{I})$. This commuting factorization is used to compute the logarithm of a motor in 2D or 3D [11], we'll briefly treat it in Section 1.5. If B happens to be a 2-blade, either ω or ν is zero. Appendix C.2 shows the detailed computation, more background may be found in [13],[17],[22].

- *Point split of dual bivector* $B_* = \mathbf{p} \cdot Q + \mathbf{R}\,\mathcal{I}$.
  Join lines represented by dual bivectors represent the (momentary) path of a motion (or its local momentum, or a force, as we will soon see). Often we have a special point Q involved in the motion, e.g., the current moving point. It is then natural to consider the decomposition of B_* into a Euclidean join line $\mathbf{p}\cdot Q$ through that point, and an ideal remainder $\mathbf{R}\mathcal{I}$. An example is the description of the momentum of an object relative to its center of mass.

  This point split is geometrical, its transform by a motor M is a similar split relative to the transformed point $Q' \equiv MQ\widetilde{M}$, for $B'_* = MB_*\widetilde{M} = (M\mathbf{p}\widetilde{M}) \cdot (MQ\widetilde{M}) + (M\mathbf{R}\widetilde{M})\mathcal{I} = \mathbf{p}' \cdot Q' + \mathbf{R}'\mathcal{I}$.

In this text, we will use whichever split is most convenient for the local purpose at hand. But we reiterate that in PGA, one should be able to work without splitting at all.

<table>
<tr><th>grade 1</th><th>grade 0</th><th colspan="2">grade 2</th><th>grade 4</th><th>grade 3</th></tr>
<tr><td>vector
$\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$</td><td>scalar
1</td><td>E-bivector
$\mathbf{e}_{23}, \mathbf{e}_{31}, \mathbf{e}_{12}$</td><td>$\boldsymbol{\epsilon}$-bivector
$\mathbf{e}_{01}, \mathbf{e}_{02}, \mathbf{e}_{03}$</td><td>pseudo
$\mathbf{e}_{0123}$</td><td>trivector
$\mathbf{e}_{123}, \mathbf{e}_{032}, \mathbf{e}_{013}, \mathbf{e}_{021}$</td></tr>
<tr><td rowspan="2">plane
(normal equation)</td><td rowspan="2"></td><td colspan="2">line
(Plücker coordinates)</td><td rowspan="2"></td><td rowspan="2">point
(homogeneous coordinates)</td></tr>
<tr><td colspan="2">screw
(Lie algebra)</td></tr>
<tr><td rowspan="2">plane reflector</td><td colspan="2">rotator
(unit quaternion)</td><td colspan="2">translator</td><td rowspan="2">point reflector</td></tr>
<tr><td colspan="4">motor
(unit dual quaternion, Lie group)</td></tr>
<tr><td>1 reflection</td><td colspan="4">0/2/4 reflections</td><td>3 reflections</td></tr>
</table>

Table 1.1: *The components of the elements of 3D PGA can be stored as different kinds of blades, often in quadruples. If done in the smart SDK arrangement, this can use SIMD/GPU hardware for graphics efficiently. E denotes Euclidean elements; $\boldsymbol{\epsilon}$ denotes elements on the vanishing plane. (Actually, the translator is placed here for symmetry, its scalar is always 1 so does not need to be stored, and we set its pseudoscalar to zero; this artificial symmetry improves efficiency.) In gray, we have indicated classical concepts now structurally embedded in PGA. Minor caveats: not all linear combinations of the bivectors are lines, only those that are factorziable by the outer product. Similarly, the point reflector is indeed 3 planar reflections; but in general, 3 planar reflections generate a more general transformation. From [13].*

1.4 The Elements of 3D PGA

Clearly, 3D PGA interests us particularly, due to the universe we found ourselves in at birth. Table 1.1, adapted from [13], gives a full overview of the elements of 3D PGA.[5] It arranges them in a convenient manner, highly suitable for implementation on SIMD/GPU hardware. Quadruples represent data items that are naturally grouped in geometric computations, with a basis chosen to have the sign swap behavior within each group be universal (see the detailed description in [13]). The indicated arrangement minimizes data fetches.

In 3D PGA, both meet lines and join lines are represented on a bivector basis. In the table, they are denoted simply as 'lines'; similarly, the term 'screws' is used to stand for both screws and co-screws (for those who know Screw Theory, see Section 2.8).

When we start computing kinematics and dynamics, some routine computations recur. We have collected some of those nuggets and their derivations in Appendix C.1, so that we can refer to them and focus more fully on the physics than on the algebra in the main text. Skip them at first reading, and use them as needed; or view them as exercises to brush up

[5] If you like to have this picture around, we have its mug shot as one of our Geometric Products at `bivector.net/merch.html`.

```
// 14 muls, 5 additions, 1 divide, 1 acos, 1 sqrt
function log(r) {
  if (r[0]==1) return bivector(r[1],r[2],r[3],0,0,0);
  var a = 1/(1 - r[0]*r[0]),                // inv squared length.
      b = Math.acos(r[0])*Math.sqrt(a),     // rotation scale
      c = a*r[7]*(1 - r[0]*b);              // translation scale
  return bivector( c*r[6] + b*r[1], c*r[5] + b*r[2], c*r[4] + b*r[3], b*r[4], b*r[5], b*r[6] );
}
```

```
// 17 mul, 8 add, 2 div, 1 sincos, 1 sqrt
function exp(B) {
  var l = (B[3]*B[3] + B[4]*B[4] + B[5]*B[5]);
  if (l==0) return rotor(1, B[0], B[1], B[2], 0, 0, 0, 0);
  var m = (B[0]*B[5] + B[1]*B[4] + B[2]*B[3]), a = sqrt(l), c = cos(a), s = sin(a)/a, t = m/l*(c-s);
  return rotor(c, s*B[0] + t*B[5], s*B[1] + t*B[4], s*B[2] + t*B[3], s*B[3], s*B[4], s*B[5], m*s);
}
```

Figure 1.3: *Efficient algorithms for* log *and* exp *in 3D PGA, by Steven De Keninck from [6]. Verifying their correctness is a useful exercise in PGA techniques. The bivector basis used is* $(\mathbf{e}_{01}, \mathbf{e}_{02}, \mathbf{e}_{03}, \mathbf{e}_{23}, \mathbf{e}_{31}, \mathbf{e}_{12})$.

on your GA skills.

1.5 The Motor Logarithm

While not strictly necessary for this text, it is probably good to know some more details about the relationship between motors and bivectors, especially in 3D PGA. This will aid you in interpolating motions.

A bivector B is a natural parametrization of a motor $M = \exp(B)$ (note that we omit the usual factor of $-\frac{1}{2}$, to keep the correspondence a bit more straightforward in this section). Conversely, given a motor, its bivector can be found by means of the GA logarithm:

$$M = e^B \quad \Longleftarrow \quad B = \log(M).$$

There is the usual ambiguity of 2π in bivector angle for logarithms of rotational motors (which is why we avoid the double implication, though in practice it holds well enough).

Multiplication of motors does *not* lead to the addition of their bivectors, and so the logarithm of a product is not simply the sum of the logarithms of the factors. This only holds if the factors were commuting; there is some extra work required if they were not (see [22] for the details in n-dimensional GA, of arbitrary signature).

In 3D PGA there are efficient algorithms for computing exp and log, indicated in Figure 1.3. The log results in a bivector in the form of its Chasles split (see Section 1.3.5).

Chapter 2

Geometrical Physics

2.1 The Kinematics of Euclidean Motion

Motions are represented by motors, i.e., exponentials of bivectors (see [13]). In the usual convention, a bivector Bt would lead to the motor $\exp(-Bt/2)$ (the inclusion of the factor $-\frac{1}{2}$ guarantees that orientation angles and translation measures retain their usual classical values and orientations). When we study kinematics and dynamics, we are interested in changes of motion. There are some subtleties, which we make explicit in this Section.

2.1.1 Resolving a Differentiation Paradox

We start with a paradox:

> When differentiating a motor $M(t) = \exp(-Bt/2)$ (with a constant bivector B), we obtain $\frac{d}{dt}M \equiv -\frac{1}{2}B\ \exp(-Bt/2) = -\frac{1}{2}B\,M$, so that $B = -2\dot{M}\widetilde{M}$. On the other hand, since B and $\exp(-Bt/2) = M$ commute, we could also write $\dot{M} = -\frac{1}{2}M\,B$, and find $B = -2\widetilde{M}\dot{M}$. The two answers are in general not the same, so which is the true rate of change bivector B of the motor $M(t)$?

Actually, we should have phrased the question more subtly. When we have a time-varying motor, it should be considered in its context: it was a motor applied to a certain motion state M_0. That motor M_0 brought an element X, originally in a standard specification state X_0, to its current position and attitude, resulting in the initial element $M_0X_0\widetilde{M_0}$ at time 0. From that state, it is moved by $\exp(-Bt/2)$. So we should have stated that the actual time-dependency of the motor M is $M(t) = \exp(-Bt/2)M_0$. Differentiating we now find (not assuming anything about commutation relations of M_0 and B):

$$\dot{M} = -\tfrac{1}{2}B\,M, \quad \text{so} \quad B = -2\,\dot{M}\,\widetilde{M}. \tag{2.1}$$

This is the bivector rate of the motor in the *world frame*: it describes the time derivative for elements already moved to an initial state by the 'offset' motor M_0.

We can also consider the derivative for a time-dependent motor $\exp(-Bt/2)$ which was applied first, moving the object in the body frame, after which a steady motor M_0' was applied. At $t = 0$, this should be the same state as above, so the two M_0's coincide, i.e.,

$M_0' = M_0$, but the B's are different. Then adding primes to distinguish this alternative situation, we have $M(t) = M_0 \exp(-B't/2)$, and we find

$$\dot{M} = -\tfrac{1}{2} M\, B', \quad \text{so} \quad B' = -2\, \widetilde{M}\, \dot{M}. \tag{2.2}$$

This is the bivector rate of the motor in the *body frame*.

The paradox is now resolved: we would characterize the circumstance in which body and world frame initially coincide at $t = 0$ by setting $M_0 = 1$; and precisely then and only then are eq.(2.1) and eq.(2.2) interchangeable.

In a typical situation we could choose to describe a motion in either way, so that the time dependent motor $M(t)$ can be expressed as $M(t) = \exp(-Bt/2)M_0 = M_0 \exp(-B't/2)$. It follows that the bivector rates of the two representations are closely related: world frame B is the moved version of body frame B':

$$B = M\, B'\, \widetilde{M}.$$

Both B' and B have their uses to describe the rate of change of the motor M at time t: though the world frame description may appear more natural, actually the motion relative to the body frame will be found in Section 2.2.9 to simplify setting up the differential equations of the motion, and computing their solution.

2.1.2 The Bivector Velocity (or Rate) B

Let us generalize slightly, for in the above $M = \exp(-Bt/2)$ had a constant rotational velocity B. In general this is not the case; yet the conclusions of eq.(2.1) and eq.(2.2) are still essentially valid; and in that constant-B form rather easy to remember or rederive (which is why we showed them first).

But more generally, take a world frame motor $M(t) = \exp(-B(t)/2)M_0$ with $B(t)$ a time-dependent bivector. Then the time derivative will involve $\frac{d}{dt}B(t)$, and it is customary to denote this 'bivector velocity' by B. We can now show that

$$B = -2\dot{M}\, \widetilde{M}, \tag{2.3}$$

so in form equivalent to the case of constant B above. Note that the measure of B is 1/[time], since motors are dimensionless. Here is how we derive eq.(2.3):

1. Since M is a motor, it satisfies $M\widetilde{M} = 1$. Differentiating, we find $\dot{M}\widetilde{M} + M\dot{\widetilde{M}} = 0$, so

$$\dot{M}\widetilde{M} = -M\dot{\widetilde{M}}, \quad \text{so} \quad \dot{\widetilde{M}} = -\widetilde{M}\dot{M}\widetilde{M}. \tag{2.4}$$

2. With this result, we can write

$$\begin{aligned}
\dot{X}(t) &\equiv \tfrac{d}{dt}X(t) \\
&= \tfrac{d}{dt}(MX_0\widetilde{M}) \\
&= (\tfrac{d}{dt}M)X_0\widetilde{M} + MX_0(\tfrac{d}{dt}\widetilde{M}) \\
&= \dot{M}X_0\widetilde{M} - MX_0(\widetilde{M}\dot{M}\widetilde{M}) \\
&= (\dot{M}\widetilde{M})\,(MX_0\widetilde{M}) - (MX_0\widetilde{M})\,(\dot{M}\widetilde{M}) \\
&= \big(-\tfrac{1}{2}B\big)\,X(t) - X(t)\,\big(-\tfrac{1}{2}B\big) \\
&= X(t) \times B,
\end{aligned}$$

using eq.(2.3). This employs the geometric algebra *commutator product* $\times$ defined by

$$P \times Q \equiv \tfrac{1}{2}(P\,Q - Q\,P). \tag{2.5}$$

3. The derivative of X must have the same grade as X. Only for bivectors is the grade preserved in a commutator product (see e.g. [10]), therefore B must be a bivector.

   In a 4-dimensional geometric algebra (like 3D PGA is), we can show this grade preservation directly, by a memorable argument. Rewrite $B = -2\dot{M}\widetilde{M} = 2M\dot{\widetilde{M}} = 2(\dot{M}\widetilde{M})^{\sim} = -\widetilde{B}$. As an element of even grade that is minus its own reverse in a 4D algebra, B must be a bivector. (From 6D onwards, this simple reasoning no longer holds; but B is still always a bivector.)

Depending on the nature of the motor M, its associated bivector B can be a linear velocity, angular velocity, or a combination of those. We will therefore denote B, neutrally, as the *rate* of the motion (in Screw Theory, this type of screw may be called a *twist*).

2.1.3 World Frame and Object Frame Derivatives

Let us summarize our results, in a notation which denotes the frames as subscripts: w for world, b for body.

In the world frame for a rate bivector B_w we have:

$$\dot{M} = -\tfrac{1}{2}\,B_w\,M. \tag{2.6}$$

In the body frame, with rate bivector $B_b = \widetilde{M}\,B_w\,M$, this derivative transforms to

$$\dot{M} = -\tfrac{1}{2}\,M\,B_b. \tag{2.7}$$

We will use whatever is more convenient to express our results. The two rates are related by:

$$B_w = M\,B_b\,\widetilde{M}.$$

The derivative of a *constant* element X_0 that was moved by a motor M to become $X_w = M\,X_0\,\widetilde{M}$ is

$$\dot{X}_w = X_w \times B_w. \tag{2.8}$$

If the element X is itself moving, such as a time-dependent X_b that moves to $X_w = MX_b\widetilde{M}$, then the derivative is more involved. The rate $\dot{X}_w$ of the motion is not simply the moved rate $\dot{X}_b$ by the naive pattern matching $\dot{X}_w = M\dot{X}_b\widetilde{M}$ (in a mistaken analogy to the correct identity $X_w = MX_b\widetilde{M}$). Rather, we compute

$$\begin{aligned}
\dot{X}_w &= \tfrac{d}{dt}(MX_b\widetilde{M}) \\
&= (MX_b\widetilde{M}) \times B_w + M\dot{X}_b\widetilde{M} \\
&= X_w \times B_w + (\dot{X}_b)_w.
\end{aligned} \tag{2.9}$$

Partly because of these extra terms, we will find the use of the world frame less convenient than the body frame when we develop dynamics.

2.1.4 Moving a PGA Point

We are particularly interested in *moving a point*, represented as a trivector X of the form $X = O + \mathbf{x}\mathcal{I}$. Its velocity should of course be the direction trivector (or vanishing point)

$$\dot{X} = \tfrac{d}{dt}(O + \mathbf{x}\mathcal{I}) = \dot{\mathbf{x}}\mathcal{I} = X \times B_w. \tag{2.10}$$

This is straightforward and compact in PGA, and usable in this form. If you would do a Euclidean split on the rate bivector $B_w = \mathbf{B}_w + \boldsymbol{\epsilon}\mathbf{v}_w$ to track linear and angular effects (as one usually does in the classical treatment) it leads to the more detailed expression $\dot{\mathbf{x}} = \mathbf{v}_w + \mathbf{x} \cdot \mathbf{B}_w$ for the velocity. We have placed all such correspondences to the classical formulae in Appendix A, this one in Section A.2. We do not need them to perform our computations, but they may reassure the reader that nothing is lost in the PGA treatment relative to the classic approach.

2.1.5 Transferring Rates: Preserve your Ideals

The transfer of rates between world frame and body frame is more subtle than you might suspect. For instance, take a point mass that is translating uniformly, i.e., it is moving with the motor $M = \exp(-\boldsymbol{\epsilon}\mathbf{v}_w t/2)M_0$, with $\mathbf{v}_w$ constant. The rate of this motion is $B_w = -2\dot{M}\widetilde{M} = (-2)(-\boldsymbol{\epsilon}\mathbf{v}_w/2)M\widetilde{M} = \boldsymbol{\epsilon}\mathbf{v}_w$, as expected.

You might think that in the body frame, the object is at rest and therefore has no translational rate. Yet computing, we find $B_b = -2\widetilde{M}\dot{M} = \widetilde{M}(\boldsymbol{\epsilon}\mathbf{v}_w)M = \widetilde{M}M\widetilde{M_0}(\boldsymbol{\epsilon}\mathbf{v}_w)M_0 = \widetilde{M_0}(\boldsymbol{\epsilon}\mathbf{v}_w)M_0 \equiv \boldsymbol{\epsilon}\mathbf{v}_b$. Thus the linear motion rate in the world frame is also felt in the body frame, with $\mathbf{v}_b = M_0\mathbf{v}_w\widetilde{M_0} = \widetilde{R}_0\mathbf{v}_w R_0$, the counter-rotated world velocity (for you can easily verify that the translational part T_0 of $M_0 = T_0R_0$ has no effect on the translation-invariant translation specification).

Of course B_b could not have been zero, since then the solution to $\dot{M} = -\frac{1}{2}MB_b$ would have been only a constant motor $M = M_0$, contradicting our actual world motion. The conceptual refinement we apparently have to make is that *ideal elements like* $\boldsymbol{\epsilon}\mathbf{v}$ *are also part of the body frame.* A frame for Euclidean motion description in 3D should be 6D, to specify all rotation and translation parameters. The former reside at the chosen location, the latter at infinity; but you need them both.

The classical picture of a 'velocity attaching at the centroid' (which would be zero within the body frame) is just not sufficiently precise: the ideal elements are felt wherever you are (though usually in a rotated manner if you changed your orientation). Though we may differ in origin, we should share our ideals: the algebra tells us so.

2.2 Moving Masses

Classically, the locational and orientational aspects of a motion have to be treated separately. By contrast, PGA can integrate them compactly, by encoding the local momentum as a join line, and the total momentum as a screw – both of which are dual bivectors. Only when we perform the (nonequivariant) Euclidean split does the separation between translation and rotation re-appear: it is an artefact of a *description* of Nature, not of Nature itself.

So we avoid the split here; if you need to see the correspondence to believe PGA, consult Appendix A; we provide pointers throughout.

2.2.1 The Momentum of a Mass Point is a Line

Momentum is a measure for the 'impetus' of a motion, defined as 'mass times speed'. Classically, momentum is separated into a linear momentum (mass times velocity) and an angular momentum (mass times cross product of rotation axis vector and location). Both are classically vectors; so adding them would confuse the two contributions. In PGA, we just have one dual bivector momentum (a join line), and the linear and angular momentum simply appear as (retrievable) components of it in a point split.

We form the momentum of a point X with mass m as the join line determined by its location and its change. So we join X to the displaced point $X + \dot{X}$ after one time unit passed; the result is the same as joining X to $\dot{X}$, since $X \vee X = 0$.

This join defines the *momentum blade* P for a physical point with mass m at location X, moving with velocity $\dot{X} = \dot{\mathbf{x}}\mathcal{I}$. (The P-based notation for momentum follows the tradition in classical physics, probably traceable to 'imPetus'.)

$$\text{momentum of point mass} \quad P \equiv m\,X \vee \dot{X} = m\,X \vee (\dot{\mathbf{x}}\mathcal{I}) = m\,\dot{\mathbf{x}} \cdot X.$$

(For the rewrite to the compact final form $\dot{\mathbf{x}} \cdot X$, see eq.(C.7).) The measure of the PGA momentum is still the same as usual: it is [mass $\times$ length/time], since the point X is a dimensionless quantity.

From the final form you can see that the momentum is an element with 1 dimension less than the d-blade of a point, so it is a $(d-1)$-blade in the PGA of d-dimensional space. In 3D PGA, the momentum is therefore a (dual) 2-vector: it represents the join line along which the motion takes place. In 2D PGA, the momentum is a 1-vector; that is indeed also the join line of motion, because in 2D space (where hyperplanes are lines), PGA 1-vectors represent lines.

$P = m\mathbf{v} \cdot X$ is the momentum join line that passes through the point X. We can consider that same physical quantity also from a point Y, at a relative position $\mathbf{r} = \mathbf{y} - \mathbf{x}$ relative to X. When we perform that point split to Y, we find:

$$\begin{aligned} P &= m\,\mathbf{v} \cdot X \\ &= m\,\mathbf{v} \cdot Y + m\,\mathbf{v} \cdot (X - Y) \\ &= m\,\mathbf{v} \cdot Y + m\,\mathbf{v} \cdot (\mathbf{r}\mathcal{I}) \\ &= m\,\mathbf{v} \cdot Y - (\mathbf{r} \wedge m\mathbf{v})\,\mathcal{I}. \end{aligned}$$

We thus find that when seen from Y, the momentum P at X automatically acquires an appropriate angular momentum (in 3D, it can be rewritten to the familiar vector expression $-(\mathbf{r} \wedge m\mathbf{v})\,\mathcal{I}_3 = \boldsymbol{\epsilon}\,(\mathbf{r} \times m\mathbf{v})$, with $\times$ the 3D cross product).

If you happen to take Y along the join line, then $\mathbf{r} \wedge \mathbf{v} = 0$, so the second term vanishes and $P = m\mathbf{v} \cdot X = m\mathbf{v} \cdot Y$. Therefore these join lines are not quite to be imagined like tangent vectors, they are not 'a velocity at a point': the PGA momentum P is the same for any other chosen point on the join line. There is a 'gauge freedom' of sliding along the line without affecting the value, which localized tangent vectors lack.

Note that the classical non-localized momentum vector $\mathbf{p} \equiv m\,\dot{\mathbf{x}}$ can be used to rewrite the PGA momentum to $P = \mathbf{p} \cdot X$: the classical momentum vector attached to the trivector point X (in 3D) to produce a join line. But we will not really want to revert to using the classical entity $\mathbf{p}$ by itself; it just is not sufficiently descriptive, since it does not indicate where in space the momentum occurs. That defect of $\mathbf{p}$ is the fundamental cause behind many classical rules for transferring inertial properties (like Steiner's theorem). None of those are needed with the PGA momentum P, which contains the spatial location X to provide a sufficiently localized characterization of the physical momentum.

2.2.2 Total Momentum of a Mass Point Set

Now consider a rigid body consisting of a set of discrete mass points $m_i X_i$. Its *total momentum* is naturally defined as the sum of the contributing momentum blades:

$$P \equiv \Sigma_i\, P_i. \tag{2.11}$$

However in 3D PGA, with its 4-dimensional representation space, this sum of 2-blades P_i is generally no longer a 2-blade: it is a general bivector, not factorizable by the $\wedge$-product. (And in more than 3 dimensions, the same thing happens, P is a $(d-1)$-*vector* in a $(d+1)$-dimensional space, and no longer guaranteed to be a $(d-1)$-*blade*.) Therefore the total momentum is no longer a join line (we might call it a *join screw*). By contrast, in 2D the total momentum remains line-like: 1-vector momenta add to produce the 1-vector of a 2D line.

2.2.3 The Inertia Map

We develop a more compact form of the total momentum for a point set subjected to a motor $M(t) = \exp(-B_w(t)/2)$, using $\dot{X}_i = X_i \times B_w$ of eq.(2.8).

$$\begin{aligned} P &= \Sigma_i\, P_i \\ &= \Sigma_i\, m_i\, X_i \vee \dot{X}_i \\ &= \Sigma_i\, m_i\, X_i \vee (X_i \times B_w) \\ &\equiv \mathsf{I}_w[B_w]. \end{aligned} \tag{2.12}$$

Here we recognized that the momentum depends linearly on the world frame rate bivector B_w, and we defined the *total inertia operator* $\mathsf{I}_w[\,]$ as that linear function,

$$\text{total inertia:} \quad \mathsf{I}_w[B_w] \equiv \Sigma_i\, m_i\, X_i \vee (X_i \times B_w). \tag{2.13}$$

This function $\mathsf{I}_w[\,]$ converts the rate bivector B_w into the momentum $(d-1)$-vector P, and is thus algebraically a kind of dualization in the $(d+1)$-dimensional PGA space. In 3D, this is not so noticeable, since then it converts a meet line bivector to a join line bivector. In 2D, the rate 2-vector B_w becomes a 1-vector momentum, clearly of a different grade.

The inertia $\mathsf{I}_w[\,]$ summarizes the essence of the mass distribution in the point cloud, for the purpose of dynamics. Different point clouds with the same inertia operator are equivalent from a dynamics point of view. For interactions like collisions between objects, more detailed aspects of the object shape are of course still relevant. When the mass distribution is continuous (a mass density), the sum is replaced by an integral.

2.2.4 Total Inertia Computed in the Body Frame

Since it is convenient to specify the properties of a body in its own frame (in physics, typically centered on the centroid and aligned with the principal axes of inertia), let us rewrite the total inertia $\mathsf{I}_w[\,]$ in those terms. So rather than computing the total world frame inertia $\mathsf{I}_w[B_w]$ as a function of B_w, we consider $\widetilde{M}\mathsf{I}_w[B_w]M$ as a function of $B_b = \widetilde{M}B_wM$. Let us denote that *total body inertia* by $\mathsf{I}_b[\,]$.

$$\text{total body inertia:} \quad \mathsf{I}_b[B_b] \equiv \widetilde{M}\,\mathsf{I}_w[M B_b \widetilde{M}]\,M. \tag{2.14}$$

We can then map to the classical entities by performing a point split in B_b and on $\mathsf{I}_b[B_b]$, relative to the centroid of the object. We emphasize that this splitting is *not* necessary for using the inertia computationally: the total momentum P can be computed straight from eq.(2.12), in world or body frame. But we will want to relate the PGA inertia expression to those of classical physics, if only to use the tables with inertial moments for commonly occuring shapes; there the inertia is customarily defined relative to the centroid. So let us bring the PGA momentum P in that point split form expression (without changing its value), by the following steps.

- **Fixed body reference point**
  We introduce a point fixed in the body to document the body's motion, and call it our origin O. When we track the motion of this point under motor M, it moves to $MO\widetilde{M}$.

- **Relative locations $\mathbf{r}_i$**
  In a chosen body reference frame at O, the points of the body at world frame locations $\mathbf{x}_i$ have locations relative to O, at relative vectors $\mathbf{r}_i$. So when expressed in the body frame, we have that
  $$X_i \equiv M\,R_i\,\widetilde{M} = M\,(O + \mathbf{r}_i\mathcal{I})\,\widetilde{M}.$$

- **Center of Mass**
  Let us now pick our frame such that its origin is the center of mass of the body. In that centroidal body frame, the relative vectors, weighted by mass, should add to zero:
  $$\text{body frame at center of mass:} \quad \Sigma_i m_r \mathbf{r}_i = \mathbf{0}.$$
  We can quickly check that the centroid C viewed as world frame point at location $\mathbf{c} = \Sigma_i m_i\mathbf{x}_i/\Sigma_i m_i$ (using position vectors $\mathbf{x}_i$ of the point X_i from the world frame origin) is indeed equal to the transformed origin O of the centroidal body frame:
  $$C \equiv O + \mathbf{c}\,\mathcal{I} = O + \frac{\Sigma m_i\mathbf{x}_i}{\Sigma_i m_i}\,\mathcal{I} = \frac{\Sigma m_i X_i}{\Sigma_i m_i} = \Sigma_i m_i\big(M\,(O + \mathbf{r}_i\mathcal{I})\,\widetilde{M}\big)/\Sigma_i m_i = M\,O\,\widetilde{M}. \tag{2.15}$$

- **Classical inertia**
  We expect the classical inertia in the body frame to make an appearance in our treatment. In the GA treatment of classical mechanics, this inertia map is a bivector-valued linear function on bivectors [20, 9], defined as:
  $$\text{classical GA inertia:} \quad \mathsf{I}_C[\mathbf{A}] \equiv \Sigma_i\, m_i\, \mathbf{r}_i \wedge (\mathbf{r}_i \cdot \mathbf{A}), \tag{2.16}$$

where $\mathbf{A}$ is a Euclidean 2-blade in the body frame (typically indicating the rotational axis and its velocity). Details on its geometric interpretation may be found in Section A.4, and Exercise 3 relates this bivector function $\mathsf{I}_C[\,]$ to the even more classical (we might even say 'antique') vector-based version of the inertia.

- In order to employ the classical inertia formula, the rate bivector B_b in the centroidal frame should be split relative to the centroid:

$$B_b \equiv \mathbf{B}_b + \boldsymbol{\epsilon}\mathbf{v}_b. \tag{2.17}$$

Note that in the centroidal frame, the point split coincides with a Euclidean split.

When we now execute this point split to the centroid (in detail through eq.(A.4) in Section A.3), we find that the PGA inertia $\mathsf{I}_b[]$ indeed contains a GA version of the classical inertia $\mathsf{I}_C[]$:

$$\mathsf{I}_b[B_b] \equiv \mathsf{I}_b[\mathbf{B}_b + \boldsymbol{\epsilon}\mathbf{v}_b] = m\,\mathbf{v}_b \cdot O - \mathsf{I}_C[\mathbf{B}_b]\,\mathcal{I}. \tag{2.18}$$

Thus $\mathsf{I}_b[B_b]$ contains components relating to the classical mass m in the *linear momentum* $m\mathbf{v}_b$, and includes the classical bivector inertia $\mathsf{I}_C[\mathbf{B}_b]$ as its non-Euclidean part in an *angular momentum* term. (For the world frame inertia map, this is discussed in more detail in Section A.8.)

The correspondence eq.(2.18) implies that we can use familiar techniques to compute the classical body inertia map $\mathsf{I}_C[]$ and in turn compute the PGA body inertia map $\mathsf{I}_b[]$. Those techniques might involve looking up the principal inertial moments of common shapes in a table, or using an SVD or diagonalization algorithm on the point data relative to the centroid, to determine the principal frame and its eigenvalues. Embedding such classical inertias into the PGA inertia gives you great advantages. notably linearity over object composition (which classical inertia does *not* obey, as we will emphasize in Section 2.2.7).

2.2.5 PGA Body Inertia in the Eigenbasis

In practice, we can specify the PGA total body frame inertia map $\mathsf{I}_b[\,]$ by stating what it does on the inertial body frame $\{\mathbf{E}_i\}_{i=1}^{\binom{d}{2}}$ extended with its PGA dual $\{\mathcal{I}\mathbf{E}_i\}_{i=1}^{d}$. When you have a discrete point distribution, this map can be established directly from its definition eq.(2.13).

For a more continuous distribution of mass, one can appeal to the classical way of computing or tabulating inertias, on an eigenbasis. Let us specify the steps in 3D space, with its 6-dimensional basis for the PGA bivectors: the Euclidean basis 2-blades $\{\mathbf{e}_{23}, \mathbf{e}_{31}, \mathbf{e}_{12}\}$ extended with the ideal basis 2-blades $\{\mathbf{e}_{01}, \mathbf{e}_{02}, \mathbf{e}_{03}\}$. The detailed description of the eigenbasis may be found in Section A.6.

1. Given a point cloud, we compute the classical eigenanalysis of its inertia matrix $\mathbf{a} \mapsto \mathsf{I}_C[\mathbf{a}\mathbf{I}_3]/\mathbf{I}_3$ (where we use the Euclidean pseudoscalar $\mathbf{I}_3$ to convert the arguments to vectors, to allow the convenient use of vector-based common eigenvector software). The eigenvectors $\mathbf{e}_k$ correspond directly to the eigenbivectors (eigenlines) $\mathbf{e}_{ij} \equiv \mathbf{e}_k\mathbf{I}_3$, with eigenvalues i_{ij}.

2. We then also determine the dual basis $\mathbf{e}_{0k}$ of ideal lines (simply by multiplying $\mathbf{e}_{ij}$ by $\mathcal{I}$). This the 'linear momentum' part of the total inertia, and the eigenvalues of the $\mathbf{e}_{0k}$ are all equal to the total mass $m \equiv \Sigma_i m_i$.

3. We only have to do steps 1 and 2 once, off-line, for a given rigid point cloud. At run time, given the body frame bivector B_b for which we want to compute $\mathsf{I}_b[B_b]$, we decompose B_b onto the bivector eigenbasis

$$B_b = \sum_{ij} [B_b]_{ij}\, \mathbf{e}_{ij} + \sum_k [B_b]_{0k}\, \mathbf{e}_{0k} \tag{2.19}$$

(where the square bracket indicates 'taking the coefficient' of the component indicated in its subscript). Then to obtain $\mathsf{I}_b[B_b]$, multiply those coefficients by the corresponding eigenvalues, but 'swapping' the coefficients of the eigenvectors:

$$\mathsf{I}_b[B_b] = \sum_{ij} m\, [B_b]_{0k}\, \mathbf{e}_{ij} + \sum_k \mathsf{i}_{ij}\, [B_b]_{ij}\, \mathbf{e}_{0k}, \tag{2.20}$$

where the index set ij cycles through $(23, 31, 12)$ while k cycles through $1, 2, 3$.

The computation of the PGA inertia is therefore not harder than that of the classical inertia; the most time-consuming part is the determination of the eigenbasis, as it always is.

If we wish to compute the total inertia $\mathsf{I}_w[B_w]$ for a world frame bivector B_w (which is $\mathsf{I}_w[B_w] = M\mathsf{I}_b[B_b]\widetilde{M}$), we can either compute it by going to the corresponding body bivector $B_b = \widetilde{M} B_w M$, picking up the eigenvectors, and transforming back the world frame; or we can perform a decomposition of B_w by projecting onto the transformed eigenbasis of bivectors $\{\, M\mathbf{e}_{ij}, M\mathbf{e}_{0k}\widetilde{M} \,\}$ and then multiply them by the corresponding (crossed) eigenvalues.

When we start doing dynamics, the inverse of the inertia map will also occur in our computations. It is simple to specify in the principal frame, using the same decomposition eq.(2.19) of its arguments, as

$$\mathsf{I}_b{}^{-1}[B_b] = \sum_{ij} \frac{1}{\mathsf{i}_{ij}}\, [B_b]_{0k}\, \mathbf{e}_{ij} + \sum_k \frac{1}{m}\, [B_b]_{ij}\, \mathbf{e}_{0k}. \tag{2.21}$$

The inertia of a single mass point is a bit of a degenerate case, since it leads to a non-invertible PGA inertia. Exercise D.10 looks into that.

2.2.6 The Inertia Map Dualizes Bivectors

We can make our notation fo d-dimensional inertia somewhat more compact by employing the Hodge dual (see Section A.6 for the derivation).

$$\mathsf{I}_b[\sum_{i=1}^{\binom{d+1}{2}} [B]_{ij}\, \mathbf{e}_{ij}] = \sum_{i=1}^{\binom{d+1}{2}} \lambda_{ij}\, [B]_{ij}\, \star\mathbf{e}_{ij}, \quad \text{with } \lambda_{ij} = \begin{cases} \mathsf{i}_{ij} & \text{for } i,j \neq 0 \\ m & \text{for } i = 0, j \neq 0. \end{cases} \tag{2.22}$$

Here m and i_{ij} are the total mass and the inertial eigenvalues in the principal basis directions $\mathbf{e}_{ij}$. To verify your understanding of this more general case, try defining the inertia map $\mathsf{I}_b[]$

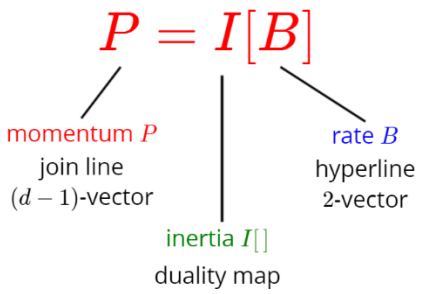


Figure 2.1: *The inertial duality map* $\mathsf{I}[]$ *takes a bivector rate* B *used to describe the kinematics of a motor, and uses the mass distribution of the object to map it to a dual bivector* P*, representing a momentum that dynamical forces can act on. It thus couples dynamics and kinematics, i.e., forces and motions, via the lopsidedness of the mass distribution.*

in 2D rather than 3D; this is Exercise D.9. We show how to implement 3D inertia compactly in Section 2.5.8.

The form of the inertia map $\mathsf{I}_b[\,]$ eq.(2.22), swapping the Euclidean and ideal aspects of its argument (with some weighting), is thus basically a weighted *Hodge dual*. Indeed, for a unit mass object with the identity matrix as its classical inertia, the inertia map $\mathsf{I}_b[\,]$ is even exactly *equal to the Hodge dual*. An example of such an object is a unit mass sphere with properly chosen radius, see Exercise D.12 (no, it is not a radius of 1!). [1]

As we will see, kinematics involves bivectors; dynamics involves dual bivectors. The connection between these two aspects is through the inertial properties of the objects involved. In 3D, where dual bivectors are representable on the same bivector basis, this structure is not so apparent. But it is very helpful to keep it in mind even then, since it allows you to keep track of where to include inertia in your equations.

2.2.7 PGA Inertia is Additive

From the definition of inertia eq.(2.12), we can easily see that the total inertia of the union of two bodies is the sum of their inertias. Say we have two objects X and Y consisting of points X_i and Y_i, respectively, moving with the same rate B_w, then

$$\begin{aligned}\mathsf{I}_w[B_w]_{X\cup Y} &= \Sigma_i m_i X_i \vee \dot{X}_i + \Sigma_i m_i Y_i \vee \dot{Y}_i \\ &= \Sigma_i m_i X_i \vee (X_i \times B_w) + \Sigma_i m_i Y_i \vee (Y_i \times B_w) \\ &= \mathsf{I}_w[B_w]_X + \mathsf{I}_w[B_w]_Y.\end{aligned}$$

There is nothing to prove, really; this is just the trivial (de)composition of a sum. You would not want a fundamental property like inertia to act differently.[2]

[1] The Hodge dual is the closest equivalent we can get of the projective dualization that was the original meaning for the 'P' in the name PGA. We point the interested reader to the balanced explanation in [17] employing the two dual spaces in full-fledged PGA. Since we decided to limit the algebra to one of the dual spaces, namely the vector space of planes, we will let 'P' stand for 'plane-based' (a term also used in [18]).

[2] PGA shares the property of 'additivity of inertia' with other 6D frameworks, such as Spatial Vector Algebra [16].

Yet for classical inertias, this simple additive property does not hold! Instead, the user needs to remember to apply the *parallel axis theorem* requiring the inclusion, by hand, of an additional term related to the displacement of the centroids of the two bodies X and Y. We show how this arises in Section A.7.

Similarly, if you would like to evaluate an inertia along a different axis, you have to add a relative term to the classical inertia map. By contrast, *in PGA inertia, we simply change the argument, not the mapping.* We can do this, because we can use an axis as an argument to the function; it is a legitimate element of the algebra.

Inertia in PGA is truly a *geometrical entity*, independent of arbitrary choices like an origin. It contains the necessary geometry inside it, as its capability to be evaluated on arbitrary axes shows; relative positions do not need to be supplied externally, their effects are automatically included in the map itself.

2.2.8 Geometry of the Inertia Map

If you need a visualization of the inertia map, let us consider 2D PGA, where join lines add simply (they remain join lines). In general, there is a (non-normalized) axis point denoted by the bivector rate B, relative to which an object moves (rotates, or translates when the axis point is ideal).

Consider an object X, made as a rigidly connected set of mass points $m_i X_i$. Each of those points is moved by the motion around the axis B in the direction of the join line $X_i \vee (X_i \times B)$, generating a momentum P_i when weighted by its mass m_i. To find the total momentum, those should all be added: in 2D they again produce a join line. This total momentum line passes through the center of mass (or is purely ideal if the motion is a translation). For different B, the momentum line $P = \mathsf{I}[B]$ is different, thus defining the linear map $\mathsf{I}[]$.

Since the inertia map is linear in its argument B, we are allowed to consider the axis point B as being split into convenient terms and then add their contributions. In 2D PGA, it is especially handy to see B as the centroid C plus an ideal point:

$$B = \omega C + \nu \mathbf{u}\mathcal{I}_2 = \omega C + \nu \boldsymbol{\epsilon}\mathbf{u}/\mathbf{I}_2,$$

where C and $\mathbf{u}$ are normalized (so $C^2 = -1$ and $\mathbf{u}^2 = 1$; remember that $\mathcal{I}_2 \equiv \boldsymbol{\epsilon}\mathbf{I}_2$). An object point X_i is decomposed as $X_i = C + \mathbf{r}_i\mathcal{I}_2$ in these C-based coordinates. Now we can tally the two contributions for these terms to the total inertia.

- The centroid term ωC in B contributes the sum of join line momenta for a pure rotation: this is the angular momentum of the object, and it can be written as $\omega \mathsf{i}\boldsymbol{\epsilon}$, i.e. as a purely ideal vector term of $P = \mathsf{I}[B]$. The value of the moment of inertia i of the object X is $\mathsf{i} = \sum_i m_i \mathbf{r}_i^2$, see Exercise D.1.

- The ideal term $\nu \mathbf{u}\mathcal{I}_2 = \nu\boldsymbol{\epsilon}(\mathbf{u}/\mathbf{I}_2) \equiv \boldsymbol{\epsilon}\mathbf{v}$ in B causes a translation of the centroid, and hence gives the linear momentum contribution to $\mathsf{I}[B]$. This is of the form $\mathbf{p} \cdot C$, in which we recognize the linear momentum $\mathbf{p} = m\mathbf{v} = m\nu\mathbf{u}/\mathbf{I}_2$, involving the total mass $m \equiv \Sigma_i m_i$, see Exercise D.2. Note that $\mathbf{v}$ is perpendicular to $\mathbf{u}$.

More algebraic details of this correspondence to the classically separate linear and angular momenta may be found in Section A.8.

Of course the inverse inertia map $\mathsf{I}^{-1}[\,]$ is also linear, and the results above specify it as well in 2D PGA, composing it from terms relative to the centroid C: $\mathsf{I}^{-1}[\mathbf{u}] = \boldsymbol{\epsilon}\,(\mathbf{u}/\mathbf{I}_2)/m$ and $\mathsf{I}^{-1}[\boldsymbol{\epsilon}] = C/\mathsf{i}$. The inverse inertia maps motion lines (in 2D PGA represented as vectors) to rates (bivectors in n-D PGA).

A momentum line at infinity is thus caused by a rotation axis at the centroid C. A finite momentum line is caused by an ideal translation 'axis' perpendicular to it. As we will see, forces directly affect the time derivatives of the momentum lines: so pushing at the centroid ('exerting a force') causes a translation, and pushing at infinity ('exerting a torque') causes a pure rotation around the centroid. A general force/torque causes a linear combination of the two.

The same principles apply in higher dimensions: we can always consider a general rate bivector as a sum of a rate at the centroid C plus an ideal term. The term at C gives a rotational inertia (parametrized by the available dimensionality for rotations which is $\binom{d}{2}$ in d-D PGA, with corresponding inertial eigenvalues i_{ij}) and represented by an ideal join line; the remaining ideal term of the rate gives a translational inertia m, for a d-dimensional linear momentum of the form $\mathbf{p} \cdot C$.

Note that the inertia map of an object seems tied to that object; especially the object's centroid plays a special role to bring it in a pleasant form. But this is merely affecting its decomposition, not its essence or value. The inertia maps of different objects can be added to provide the inertia of the total object – of which the new centroid is then automatically of similar representational relevance. But you do not need to compute that new centroid; the sum of inertias automatically and inherently incorporates it.

2.2.9 Derivative of the Inertia in the Body Frame

In Section 2.1 we saw that differential equations may look slightly different in body frame and world frame. In a body frame (which we will typically center on the centroid C, but this is not a requirement), the world frame momentum $P = P_w$ is experienced as $P_b = \widetilde{M}\,P_w\,M$. As we have shown, the inertia and therefore the momentum are easy to specify directly from the mass distribution of the points in its 'eigenframe' (which is a body frame rather than a world frame).

In Section 2.4.1 on Newtonian dynamics, we will need the derivative of the world frame momentum $P_w = \mathsf{I}_w[B_w]$, and it will again be handy to express the detailed computation in the chosen body frame. We have the tools to do so now. For the representation of B_w in the body frame as B_b, the masses m_i, the origin O and the relative vectors $\mathbf{r}_i$ in the body frame inertia $\mathsf{I}_b[\,]$ are constant, and therefore differentiation of this linear operator affects only the argument B_b.

$$\tfrac{d}{dt}\mathsf{I}_b[B_b] = \mathsf{I}_b[\dot{B}_b]. \tag{2.23}$$

By contrast, in the world frame the motors need to be differentiated as well, and that produces an extra commutator term by eq.(2.8).

$$\begin{aligned} \tfrac{d}{dt}\mathsf{I}_w[B_w] \equiv \tfrac{d}{dt}(M\,\mathsf{I}_b[B_b]\,\widetilde{M}) &= (M\,\mathsf{I}_b[B_b]\,\widetilde{M}) \times B_w + M\,\tfrac{d}{dt}\mathsf{I}_b[B_b]\,\widetilde{M} \\ &= M\left(\mathsf{I}_b[B_b] \times B_b + \mathsf{I}_b[\dot{B}_b]\right)\widetilde{M}. \end{aligned} \tag{2.24}$$

The body frame inertia has a special form which allows a minor additional simplification, requiring only the Euclidean part of B_b in the commutator product; see Exercise D.14. If you are interested in the form of the derivative of the inertia in the world frame, see Exercise D.16.

2.3 Forces, Torques and Hyperlines

Forces act on physical point masses, and so they attach at the point location and act in a Euclidean direction, with a certain magnitude. In PGA, the algebraic element corresponding to a geometrical 'join line' is therefore well-suited to characterize a force. Such a PGA join line is insensitive to translations along its direction, or rotations around it, but otherwise localized.

2.3.1 A Single 3D Force is a 2-Blade

In the PGA of d-dimensional space, the direction vector of the force $\mathbf{f}$ is represented as a vanishing point (or ideal point) represented by the d-blade $\mathbf{f}\,\mathcal{I}_d$. Here $\mathcal{I}_d$ is the pseudoscalar $\mathcal{I}_d \equiv \boldsymbol{\epsilon}\mathbf{I}_d$, and $\mathbf{I}_d$ the d-dimensional Euclidean pseudoscalar. The direction of the force is thus a trivector in 3D.

We combine location and direction in our *force line* (or line force?). Using the specification of a join line by location and direction (from eq.(1.5)) we find:

$$\text{force line:}\quad F = Q \vee (\mathbf{f}\,\mathcal{I}_d) = \mathbf{f} \cdot Q. \tag{2.25}$$

The point Q is dimensionless, but $\mathbf{f}$ and hence F has dimension measure of [force] = [mass $\times$ length/time^2] (we would use the SI unit Newton, with N = kg m/s^2).

When we consider the same force acting along the same geometrical join line, but would like to represent it relative to another location R at relative position $\mathbf{r} - \mathbf{q}$ to the point Q, we can simply perform a point split:

$$\begin{aligned} F &= \mathbf{f} \cdot Q \\ &= \mathbf{f} \cdot \big(R + (\mathbf{q} - \mathbf{r})\,\mathcal{I}\big) \\ &= \mathbf{f} \cdot R + \mathbf{f} \cdot \big((\mathbf{q} - \mathbf{r})\,\mathcal{I}\big) \\ &= \mathbf{f} \cdot R + \big(\mathbf{f} \wedge (\mathbf{q} - \mathbf{r})\,\big)\,\mathcal{I} \\ &= \mathbf{f} \cdot R - \big((\mathbf{r} - \mathbf{q}) \wedge \mathbf{f}\big)\,\mathcal{I} \\ &\equiv \mathbf{f} \cdot R - \mathbf{T}_R\,\mathcal{I} \qquad\qquad \text{(point split relative to } R\text{)} \end{aligned} \tag{2.26}$$

We thus find that the force part involving $\mathbf{f}$ simply transfers to the point R, but that at R we also feel a Euclidean *torque 2-blade* $\mathbf{T}_R \equiv (\mathbf{r} - \mathbf{q}) \wedge \mathbf{f}$ (in 3D, this is the GA counterpart of the classical vector torque $\boldsymbol{\tau}_R \equiv (\mathbf{q} - \mathbf{r}) \times \mathbf{f}$, by eq.(C.12): $\mathbf{T}_R\,\mathcal{I} = -\boldsymbol{\epsilon}\,\boldsymbol{\tau}_R$). The torque 2-blade $\mathbf{T}_R$ is also known as the *moment* of the force (though strictly classically one uses that term for the 1-vector $\boldsymbol{\tau}_R$ that is its Euclidean dual).[3] Note that the actual value of the entity F has *not* changed: it is only our split of it, relative to another point, that made the necessary torque appear there. You might view that split as a form of 'evaluating' F at R,

[3] We denoted $\mathbf{T}$ in bold font, since it is Euclidean, and in upper case since it is not a vector.

as the 'R-coordinates of F'. As we saw for PGA inertia, for PGA forces there is no need either for a 'parallel axis theorem' that you classically have to remember to apply when your attention shifts to another point R. The forque F automatically has the properties required.

2.3.2 Use the Forque!

We saw how the torque term appears when a force line is considered from a point not on it. Torques are therefore inherent in how forces are positioned in space, they are a natural aspect of force. Classically, the two terms in the Euclidean split relative to some origin are treated separately and therefore need distinct names. For PGA, they are merely aspects of one fundamental concept. In Screw Theory, this concept is called a 'wrench', but this term is not in general use – which presumably means that the concept itself is not alive. In an attempt to mend that, we propose to denote this unified, fluid representation of force and torque by the novel term '*forque*', since that much more strongly suggests the unification of the two familiar classical descriptors.

Although in eq.(2.26) we have made the torque term appear as the consequence of an offset force, it is also possible to apply a pure torque directly (for instance, using an actual physical wrench tool). Such a pure torque of the form $F = -\mathbf{T}\mathcal{I}$ is also subsumed in our term 'forque'.

In PGA, you should work with a forque F directly, rather than decompose it in terms of $\mathbf{f}$ and $\mathbf{T}$ (its classical force and torque constituents, as in eq.(2.26)). We will show the benefits of doing so! Still, you may need convincing that the simplicity of PGA contains the full physics; in Appendix A we therefore split the expressions in their force and torque parts, and the linear and angular motions, to show the correspondence to the familiar classical expressions. But the main text and the demos show that this is actually no longer required.

> **Historical note:** you may have read about Classical Mechanics done by means of geometric algebra before, and seen it done differently. In his seminal book [20], David Hestenes merges the two contributions of force and torque after the fact, forming $\mathbf{f} + \mathbf{T}$. He notes that since the two terms are vector and bivector, this shorthand is permitted; the individual parts can still be recovered. But the modern approach of PGA provides their integration immediately, and less arbitrarily, by recognizing the essential role that is played by geometrical lines. In 3D, it retains Hestenes' torque $\mathbf{T}$ as a bivector, but distinguishes two terms of the same grade by the presence of its algebra element $\mathcal{I}$ to form $F = \mathbf{f} \cdot O - \mathbf{T}\mathcal{I}$. Note the role of the trivector O (or any other point used in a point split) and pseudoscalar $\mathcal{I}$, to ensure that both terms are of the same grade, and to make the total element F become geometrically equivariant. Yet $\mathbf{f}$ and $\mathbf{T}$ are still separately retrievable, should you wish to, from the Euclidean and ideal parts.

Paradoxically, force and torque can be combined in PGA without mixing, in a manner that allows them to fluidly morph into each other when the focus of attention changes.

2.3.3 Total Forque (or Wrench)

Forques apply to objects since their constituent points may be sensitive to influences. For instance, charged masses would feel an electromagnetic forque (where the electric part may exert a force, and the magnetic part a torque). The effect that the forque has on the motion of the rigid object composed of such mass points is simply that the derivative of the PGA momentum (the velocity rates, both linear and angular) equals that total forque. The motion can be computed from this statement, by integrating the resulting differential equations.

Let us assume that we have a rigid object consisting of mass points $m_i Q_i$, with a forque F_i acting on the i-th mass point. Then we can add their result as the total *forque* F:

$$F = \Sigma_i F_i = \Sigma_i \, \mathbf{f}_i \cdot Q_i$$

If you need to see how this indeed adds the classical forces and torques as you would expect, consult the split in Section A.9.

The summation of the individual forques makes geometric (or physical) sense, but algebraically the total forque has changed character: whereas the individual forques F_i are dual 2-*blades*, the total forque is a dual 2-*vector*. The algebraic difference is that dual 2-*blades* can be factorized, but dual 2-*vectors* cannot (in 3D PGA and beyond). The total forque therefore no longer simply has an associated (join) line as the geometry of its action; the geometrical interpretation of its bivector is a finite line L paired with an orthogonal ideal line $L\mathcal{I}$ (in 3D; in more dimensions there are more terms). We mentioned how to retrieve that geometry equivariantly when we treated the screw split (or Chasles split) of bivectors in 1.3.5.

> **Physics Aside:** It is customary in treating the motion of a set of points to split the force $\mathbf{f}_i$ on a particle i into internal forces $\mathbf{f}_{ij}$ and external forces $\mathbf{f}'_i$, and (a) to assume that the internal forces obey Newton's third law: $\mathbf{f}_{ij} = -\mathbf{f}_{ji}$ and (b) that they are central (directed along the connection line of the points, so that $(\mathbf{x}_i - \mathbf{x}_j) \wedge \mathbf{f}_{ij} = 0$). The sum of all forces then equals the sum of the external forces, and the sum of all torques equals the external torques $\mathbf{T}'_i = \mathbf{q}_i \wedge \mathbf{f}'_i$:
>
> $$\Sigma_i \mathbf{f}_i = \Sigma_i \mathbf{f}'_i \quad \text{and} \quad \Sigma_i \mathbf{T}_i = \Sigma_i \mathbf{T}'_i.$$
>
> In this write-up on rigid bodies, we will not mention the internal forces anymore, and use the primeless notation for forces or external forces. But when your bodies are non-rigid, you will need to look deeper, with e.g. [20] or [1] as your guide.

We insert a remark on terminology for 3D. In the PGA of 3D space, the dual of a bivector is a bivector, so we can speak of the forque bivector (our original title was "May the Bivector Be with You"). That is OK, but it is actually helpful to bear in mind that it has a dual nature (dual to a rate bivector; from meet line to join line); that fact makes some formulas more easy to remember. *Forces are lines* in any dimension, and lines are dual bivectors. This also works unchanged in 2D (where forces are PGA vectors), and even in d-D. This uniformity greatly simplifies implementations. As some later examples will show, we can always implement a dynamics problem in the lowest dimension in which it can be defined, and then that description will be automatically valid in any higher dimension, and the program based on it will compute the correct solutions. Since a dimension-agnostic algebra thus allows for cleaner code, we will seize the opportunity to set up the algebra this way.

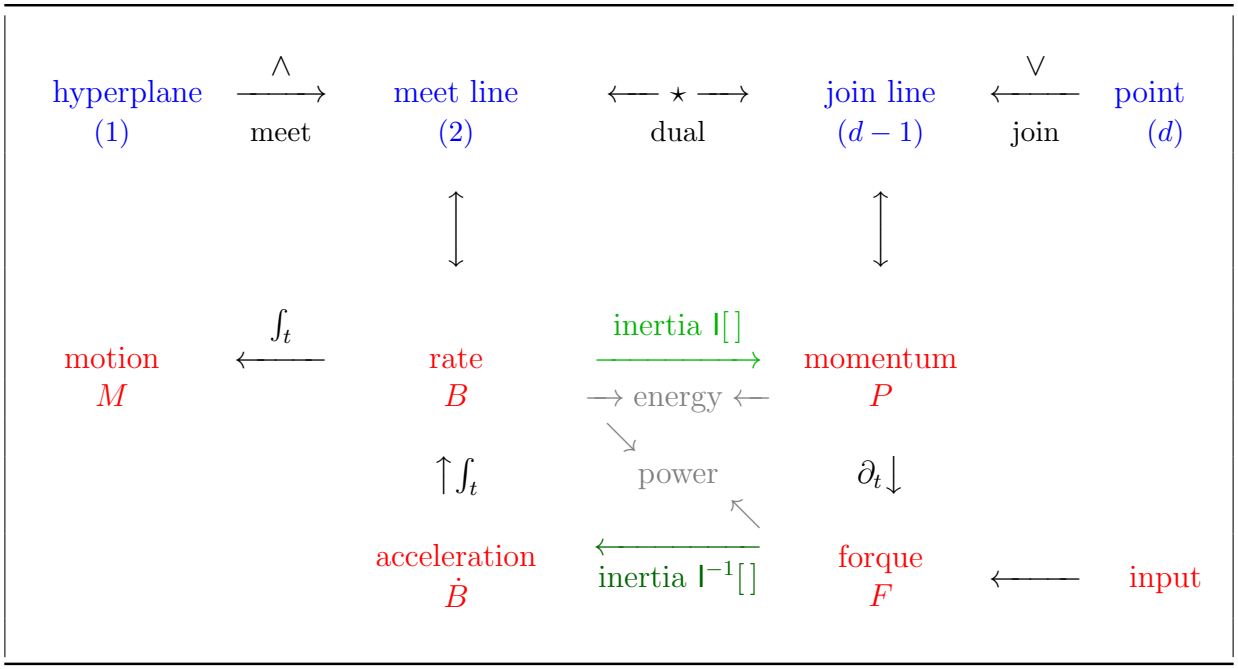


Figure 2.2: *Kinematics and dynamics are dual aspects of physical motion. They are coupled via the mass distribution as represented in the inertial duality map.*

2.4 PGA Dynamics: All in One

2.4.1 The Law of Motion

Newton's second law of motion states that the total force equals the time derivative of total momentum, and affects the motion in that manner. In the PGA formulation, this generalizes to the total PGA forque F and total PGA momentum P as:

$$\tfrac{d}{dt}P = F.$$

We now have all the ingredients to compute the motion of an object based on forces applied to it. We characterize the motion by a motor M; then the general outline is as sketched in Figure 2.2, chasing the input (a forque F) to the resulting motion M. Since we found it convenient to specify the properties of the body in the body frame (to avoid extra terms in the derivative according to eq.(2.24)), we can rewrite the equation of motion as

$$M\left(\mathsf{I}_b[B_b] \times B_b + \mathsf{I}_b[\dot{B}_b]\right)\widetilde{M} = F_w, \tag{2.27}$$

where F_w is the total forque (or wrench) in the world frame. In the body frame, this forque is felt as $F_b \equiv \widetilde{M}F_wM$. We therefore obtain two coupled equations for the motor M given the total forque F: the body bivector B_b satisfies

$$\text{dynamics equation:}\quad \dot{B}_b = \mathsf{I}_b{}^{-1}[B_b \times \mathsf{I}_b[B_b] + F_b], \tag{2.28}$$

and the motion M is related to the body bivector by eq.(2.7):

$$\text{kinematics equation:}\quad \dot{M} = -\tfrac{1}{2}\,M\,B_b. \tag{2.29}$$

Newton/Euler	$\dot{P}$	$=$	F
Inertia	$\mathsf{I}_b[B]$	$\equiv$	$\Sigma_i\, m_i\, X_i \vee (X_i \times B)$
Dynamics	$\dot{B}_b$	$=$	$\mathsf{I}_b{}^{-1}[B_b \times \mathsf{I}_b[B_b] + F_b]$
Kinematics	$\dot{M}$	$=$	$-\frac{1}{2}\, M\, B_b$

Table 2.1: *The equations of motion in PGA.*

We emphasize that these two equations *contain all forces and torques*, and both the *linear and angular motion* of the object. The dynamic object properties are encoded by the total body inertia $\mathsf{I}_b[\,]$, of which we copy the specification eq.(2.20) for completeness:

$$\mathsf{I}_b[\sum_{i=1}^{\binom{d+1}{2}} [B]_{ij}\, \mathbf{e}_{ij}] = \sum_{i=1}^{\binom{d+1}{2}} \lambda_{ij}\, [B]_{ij}\, \star\mathbf{e}_{ij}, \quad \text{with} \;\; \lambda_{ij} = \begin{cases} \mathsf{i}_{ij} & \text{for} \;\; i,j \neq 0 \\ m & \text{for} \;\; i = 0, j \neq 0. \end{cases} \tag{2.30}$$

with the $\mathbf{e}_{ij}$ and i_{ij} the eigenbivectors and eigenvalues of the body inertia map (in bivector form), and m the total mass of the body. That is all there is to dynamics in PGA.

The equations can be numerically integrated in an implementation, as we will see in the next Section. Gunn notes in [17] Section 9.5,

> When written out in full *(for 3D)*, this gives a set of 14 first-order ODE's. The solution space is 12 dimensions; the extra dimensions correspond to the normalization $M\widetilde{M} = 1$. At this point the solution continues as in the traditional approach, using standard ODE solvers. Our experience is that the cost of evaluating eq.(2.28)-eq.(2.29) is no more expensive than traditional methods.

Actually, the PGA speed is identical to the dual quaternion method - because they are algebraically isomorphic. Table 2.1 summarizes the body frame motion equations.

When an object has unit mass and equal unit principal inertias, the inertia map becomes identical to the Hodge dual. Explore the simple form the equations of motion then take in Exercise 13.

2.4.2 Geometric Intuition for the Motion Equation

We can now complete the geometrical insight of Section 2.2.8 to include the forques:

> The total momentum of a rigid object can be viewed as composed of two terms. A momentum line at infinity is caused by a rotation axis at the centroid C, and a finite momentum line by an ideal translation 'axis' perpendicular to it. Forces directly affect the time derivatives of the momentum lines: so pushing at the centroid ('exerting a force') causes a translation, and pushing at infinity ('exerting a torque') causes a pure rotation around the centroid. A general force/torque causes a linear combination of the two.

In any dimension, the velocity rate is a bivector, since it is the logarithm of a motor. The momentum is dual to the velocity, and proportional to the force. Forces are therefore always line-like; in all dimensions, these are represented as dual bivectors (so in 3D PGA a bivector, and in 2D PGA a vector).

2.5 Implementation

This section consists of partial text of `https://enki.ws/ganja.js/examples/pga_dyn.html` *(in January 2022). We recommend actually opening that webpage since it contains executable code, which is more convincing to see in action than to read about it. But we want to give at least the flavor of what the implementation looks like here on paper.*

Hopefuly we have been able to convince you that considering a problem in a dimension-agnostic way is of great value, and in fact provides a unique guide to the geometry. With the ability to formulate the mathematics in d-D comes the desire to have a true dimension agnostic implementation. There are several practical hurdles to be taken and we'll provide a short overview that should allow you to build your own dimension agnostic rigid body dynamics solver.

In the next section, we will go over the details of such an implementation, requiring nothing but a geometric algebra library. In the section after that, we focus on the 3-dimensional case and provide some details for an efficient implementation. We will use the `ganja.js` (`https://github.com/enkimute/ganja.js`) library for our implementation. It offers a syntax that is compatible with most other popular GA libraries when it comes to mathematical expressions - so you should have no trouble replacing it with the library of your choice, chosen from the list at `https://bivector.net/lib.html`.

2.5.1 Syntax

The examples in this text are all available as stash in the `ganja.js` coffeeshop at `https://enkimute.github.io/ganja.js/examples/coffeeshop.html`. They require no installation and you can easily Edit them, then Save and Run and share your own versions. Playing around with the existing examples is a great starting point to get a feel and we strongly encourage you to do so.

For those not accustomed to a geometric algebra library, Figure 2.3 gives a short overview of operators that should come in handy. Other operators like addition work as one would expect on arbitrary multivectors, and division is defined w.r.t. the geometric product. Note that the &, |, $\wedge$ binary operators keep their original bitwise functionality when both arguments are integers – we'll use that fact below.

2.5.2 Simulation setup

Rigid body simulators, or physics simulators in general are often complex pieces of code, and properly handling collision detection, integration errors, and large numbers of objects requires a lot of infrastructure that is not relevant to our current exploration. We refer the

syntax	operator	description
`a * b`	ab	Geometric Product
`a ^ b`	$a \wedge b$	"Wedge" or Outer Product
`a & b`	$a \vee b$	"Vee" or Regressive Product
`a | b`	$a \cdot b$	"Dot" or Inner Product
`~R`	$\tilde{R}$	Reverse. (i.e. inverse for rotor)
`!a`	a^*	Hodge Dual
`a.Grade(n)`	$\langle a \rangle_n$	Grade selection
`a.Normalized`	$\bar{a}$	Normalization
`E**a`	e^a	Exponential
`a.Log()`	$\log a$	Logarithm
`R >>> x`	RxR^{-1}	Sandwich product

Figure 2.3: *The notation of common GA operators in* `ganja.js`.

reader to the vast body of literature available on the subject and instead focus our attention on the geometric representations of forques, momenta, accelerations and velocities.

For anything not related to the representation we have tried to pick the simplest and most common solution. For collision detection (Section 2.6.1) this means a simple SAT algorithm is used, while for integration we pick the trivial forward Euler scheme. (Its problems are well understood).

The simple Euler integration scheme predicts a future state S_{t+h} by taking small steps h in the direction of the derivative $\dot{S}_t$ at time t:

$$S_{t+h} = S_t + h\,\dot{S}_t.$$

The simplicity of this technique is its main advantage, but it is not particularly good at energy conservation and misbehaves in the presence of stiff differential equations. Despite these disadvantages it is still widely used and allows us again to focus on the representation while keeping an integrator that is easy to implement in an environment that offers operator overloading. For each frame we take a number of fixed steps n of step size h into the future:

```
// Euler Integration
for (var i=0, h=0.01; i<n; ++i)
  state = state + h * dState(state);
```

When we need more precision, we can invoke the RK4 method.

2.5.3 The General d-dimensional Case

Before we can focus on dynamics in d dimensions, we need some tooling to create and visualize d-D meshes. We go over the techniques used in our demos in enough detail for you to recreate them in your favorite programming environment. Keep in mind that adding visualisations for the various elements of PGA is a must-have to debug and understand any algorithms you implement.

- **The Hypercube**
  The 'hyper' adjective is used to generalize 3D nomenclature to an arbitrary number of dimensions. A hyperplane is a plane in 3D, but a line in 2D and a volume in 4D. Similarly, a hypercube is a cube in 3D, a square in 2D and a tesseract in 3D. It is probably the easiest shape to generate in an arbitrary number of dimensions, so the ideal candidate for our toy examples.

- **Vertices**
  To visualize our cube we need, at a minimum, to render its vertices and edges. A hypercube in d-D has 2^d vertices v_i (a square has 4, a cube 8, a tesseract 16, etc ..). The individual coordinates can easily be deduced from the binary representation of the index i of each vertex v_i

  We can simply treat the first bit as the 'x' coordinate, the second as the 'y' and so on. With the number of dimensions expressed as 'd', the following piece of code will generate all the points of a d-dimensional hypercube:

```
var p = [...Array(2**d)]                                    // correct number of points
        .map((x,i)=>('000000'+i.toString(2)).slice(-d))     // convert to binary
        .map(x=>x.split('').map(v=>v-0.5))                  // split and center
        .map(x=>!(1e0 + x*[1e1,1e2,1e3,1e4,1e5].slice(0,d)));  // convert to PGA points
```

- **Edges**
  Equiped with the points, we still have to generate a list of edges. The edges we are interested in are the non-diagonal edges, and using the same encoding as above it is easy to see that those are the edges that differ only in one bit. We can use the binary AND & and XOR $\wedge$ operators to easily detect if two indices i, j differ in just one bit:

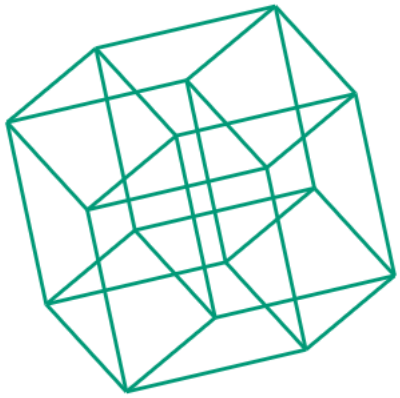
```
function is_edge(i,j) {
  var x = i^j;              // xor both together, all differing bits will be '1'
  return (x & (x-1)) == 0   // only if x has one bit, this returns true.
}
```

- **Visualizing**
  To visualize a cube in 3D, we typically use a perspective transformation. While `ganja.js` will default to an orthogonal projection for spaces of more than 3 dimensions, it is still quite easy to do a perspective transformation for your own visualisation

purposes. To do so, consider a camera at point c, and a screen s, then the projections of our hypercube points A_i are given by

$$A'_i = (A_i \vee c) \wedge s.$$

If the projection screen s aligns with the coordinate axes, the resulting points a'_i will have all but two coefficients zero, trivially enabling you to render them as 2D elements:



2.5.4 Motors in d dimensions

As we've learned in [13], isometries in any number of dimensions can all be called k-reflections. The $2k$- reflections thus represent the continuous (handedness preserving) transformations, and they can easily be generated using the exponential map.

$$M = e^{\alpha \bar{B}}.$$

While this function is well defined for arbitrary $B = \alpha \bar{B}$ in arbitrary algebras, for our purposes we will only need to consider the exponential of a 2-blade B (so that $B \wedge B = 0$), in this case the exponential map is particularly trivial

$$e^{\alpha \bar{B}} = \begin{cases} 1 + \alpha \bar{B} & \text{if } \bar{B}^2 = 0 \\ \cos\alpha + \bar{B}\sin\alpha & \text{if } \bar{B}^2 = -1 \\ \cosh\alpha + \bar{B}\sinh\alpha & \text{if } \bar{B}^2 = 1 \end{cases}$$

This allows us to easily generate translations (with pure ideal bivectors $\mathbf{e}_{0i},\ 0 < i$), and rotations (with pure Euclidean bivectors $\mathbf{e}_{ij},\ 0 < i \neq j$). The current position of our object will always be represented by a motor M, that transforms elements in the body frame X_b of our object to its place in the world frame X_w

$$X_w = M\, X_b\, \widetilde{M}$$

In the `ganja.js` code, such sandwiching is denoted by `M >>> X`; it is also correct for odd versors V, when there is a sign $(-1)^{(}\text{grade}(V)\text{grade}(X)$ involved.

2.5.5 Points

The origin itself is always the $(d-1)$-vector dual to the projective hyperplane $\mathbf{e}_0$ (which we labeled $\boldsymbol{\epsilon}$ in the mathy text), and so we can specify it in a dimension independent fashion as $\star\boldsymbol{\epsilon}$. Similarly a point at Euclidean position $\left[x_1, x_2, x_3, ...\right]$ can always be written as $\star(\boldsymbol{\epsilon} + x^i\mathbf{e}_i)$, a formulation which is independent of both the number of dimensions and the particular choice of basis. In our code, the Hodge dual is denoted by an exclamation point, so we obtain:

```
// Convert array of point coordinates x=[x1, x2, x3, ...] to a PGA point
function point(x) {
  return !(1e0 + x*[1e1, 1e2, 1e3, ...]);
}
```

2.5.6 Kinematics

To verify our simulation setup, let us simulate the kinematics of a uniform inertia hypercube (unit mass, unit inertia). A hypercube is a rotationally symmetric object, reducing our inertial duality to the standard Hodge dual. The current kinematic state is stored as a pair of multivectors $[M, B]$, where M is the motor that represents our current position and orientation and B is the bivector representing our current velocity, both specified in the body frame.

In the absence of external forces, the derivatives of these state variables are, for a uniform inertia:

$$\dot{M} = -\tfrac{1}{2}M\,B, \quad \dot{B} = \star^{-1}(B \times \star B),$$

where the $\times$ is the geometric algebra commutator product $A \times B = \frac{1}{2}(AB - BA)$. This translates to a remarkably simple state update (for uniform inertia):

```
function dState([M,B]) {
  return  [
    -0.5 * M * B,
    -0.5 * (B.Dual*B - B*B.Dual).UnDual
  ]
}
```

.

Putting everything together we can simulate the kinematics of our hypercube. Play with this as `https://enki.ws/ganja.js/examples/coffeeshop.html#irAoiHVhn` in the `ganja.js` coffeeshop; you should now recognize all the code. The motions are more involved than you might think. You can try to find settings that mimic the exact solutions depicted in Figure B.1.

2.5.7 Dynamics: Forques

When we consider external forces (forques!), we move into the field of Dynamics. In our PGA setup, forques are always along lines, and expressed in the body frame. Let us consider some simple cases.

- **Gravity**
  The easiest way to describe gravity in a dimension-independent fashion is to describe the acceleration it causes to a body in the world frame. Such an acceleration is always a bivector, and assuming gravity is acting along the negative Y direction, it simply is $-9.81\mathbf{e}_{02}$. (Actually, it might be smarter to use the $\mathbf{e}_{01}$-direction, since gravity exists from 1-D onwards; but our default visualization would plot this horizontally.)

  All we need to do to find the associated force in the body frame is to first move this ideal bivector to the bodyframe, and then dualize it to find the associated force line.

  $$F_g = \star\big(\widetilde{M}(-9.81\mathbf{e}_{02})M\big). \tag{2.31}$$

  The resulting force line will be purely Euclidean (see Section 2.4.2), a line through the center of mass of our object.

- **Spring**
  Hooke's law for a spring can be formulated in a similarly easy way. It is a line from a point on the body P_b to an attachment point A_w in the world frame, scaled with a spring constant k:

  $$F_H = k\left(P_b \vee (\widetilde{M}A_wM)\right).$$

- **Damping**
  The Euler integrator we are using is notoriously bad in energy preservation and tends to blow up in the presence of external forces. To mitigate this, we simply dissipate some of the energy at each step by creating a force inverse proportional to the current velocity bivector.

  $$F_d = \star(-\alpha B).$$

  This is not truly physics, but a commonly used trick in Euler integration, and the effect does not look unnatural.

Putting all this together, we can attach our hypercube to a string with forces encoded as:

```
// Our forques
var attach = (1-0.5e02) * p[2**d-1];
var F = (M,B)=>{
  var Gravity = !(~M >>> -9.81e02);
  var Hooke   = 12*( (~M >>> attach) & p[2**d-1] );
  var Damping = !(-0.25 * B);
  return (Gravity + Hooke + Damping);
}
```

.

The total code for a hypercube on a string, incorporating all that we treated so far, may be found in Figure 2.4. But it is better to play with it in the `ganja.js` coffeeshop (follow the link at `/ganja.js/examples/pga_dyn.html#Damping`), since you can then test your understanding by modifying it (Save and Run). See the (hyper)cube bounce!

2.5.8 Optimized Computations in the 3D Case

The examples above are, as far as we are aware, the first truly dimension independent implementations of rigid body dynamics. While this requirement is exactly what enabled this concise description, and provided a clear view on the requirements imposed by the geometry on the algebra (e.g. forces must be $(d-1)$-vectors), it is not a requirement that carries a lot of practical value. The special case of $d = 3$, and its representation of the Euclidean group is the one Nature selected, so it pays to take a closer look at the formulas in this particular case.

That's exactly what we will do in this section, focusing first on the normalisation of motors that, due to the nature of numerical integration, tend to drift away from the motor manifold. Additionally we show how the concise formulas above work out at a coefficient level to produce automatically exactly those scalar operations we used to handcraft in a not too distant past. Finally we show how to setup the inertial map and convert existing known inertia tensors to their bivector equivalent.

Basis

For the 3D case, we pick our basis so that it matches conventions from computer graphics, making it easier to integrate PGA methods in existing codebases. The basis we use for 3DPGA is the following:

$$1, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_0, \mathbf{e}_{01}, \mathbf{e}_{02}, \mathbf{e}_{03}, \mathbf{e}_{12}, \mathbf{e}_{31}, \mathbf{e}_{23}, \mathbf{e}_{032}, \mathbf{e}_{013}, \mathbf{e}_{021}, \mathbf{e}_{123}, \mathbf{e}_{0123}.$$

Renormalizing a Motor

A motor in PGA needs to satisfy the normalization condition $M\widetilde{M} = 1$, working this out in coefficients gives us the conditions on a coefficient level :

$$1 = M\widetilde{M} = m_{\emptyset}^2 + m_{12}^2 + m_{31}^2 + m_{23}^2 + 2\left(m_{\emptyset}m_{0123} - m_{01}m_{23} - m_{02}m_{31} - m_{03}m_{12}\right)\mathbf{e}_{0123} = 0,$$

(where $m_{\emptyset} \equiv \langle M \rangle_0$ is the scalar coeffient of the rotor). We see that the normalization condition in 3DPGA results in two conditions, for the coefficients of the scalar 1 and of the pseudoscalar $\mathbf{e}_{0123}$.

A multivector that has only scalar and quadvector parts is called a Study number, in our case isomorphic to the dual numbers with an inverse and square root that is well defined. Invoking some of our results on Study numbers from [6], we obtain for the normalization of a motor:

$$\overline{M} \equiv M/(M\widetilde{M}) = s(M + d(m_{23}\mathbf{e}_{01} + m_{31}\mathbf{e}_{02} + m_{12}\mathbf{e}_{03} - m_{\emptyset}\mathbf{e}_{0123})),$$

```
// Set the number of dimensions.
window.d = 4;

// Create a d-dimensional geometric algebra over the reals.
Algebra(d, 0, 1, ()=>{

  // Create a hypercube (square in 2D, cube in 3D, ...)
  // Start by defining its vertices.
  var p = [...Array(2**d)]
          .map((x,i)=>i.toString(2))              // [0, 1, 10, 11, 100, ...]
          .map(x=>('00000'+x).slice(-d))          // [000, 001, 010, 011, ...]
          .map(x=>x.split('').map(x=>x-0.5))      // [[-0.5, -0.5, -0.5], [-0.5, -0.5, 0.5], ...]
          .map(x=>!(1e0 + x*[1e1,1e2,1e3,1e4]))   // PGA points are dual vectors.

  // Now consider all vertex pairs and create edges for those
  // pairs that differ only in one coordinate.
  var e = p.map((a,i)=>p.map((b,j)=>
            i<=j||(i^j)&(i^j-1)?0:[a,b]   // note that &,^ here are bitwise ops since i,j are integer
          )).flat().filter(x=>x);

  // Physics state.
  var state  = [1, 1e12 + 1.3e13 + 0.5e24];

  // Our forques
  var attach = (1-0.5e02) * p[2**d-1];
  var F = (M,B)=>{
    var Gravity = !(~M >>> -9.81e02);
    var Hooke   = 12*( (~M >>> attach) & p[2**d-1] );
    var Damping = !(-0.25 * B);
    return (Gravity + Hooke + Damping);
  }

  // The derivative of the state.
  var dS = ([M,B])=>[
             -0.5*M*B,
             (F(M,B) - 0.5*(B.Dual*B-B*B.Dual)).UnDual
           ];

  // Render
  return this.graph(()=>{
    // Update the state
    for (var i=0; i<10; ++i)
      state = state + 1/600 * dS(state);
    return [ 0x007799, ...state[0] >>> e, attach, [attach, state[0]>>>p[2**d-1]] ];
  },{lineWidth:6,animate:1,scale:1.75});

})
```

Figure 2.4: *Dynamics code for a hypercube on a string, in gravity. Adapt* `window.d`...

with s and d computed as:

$$s = \frac{1}{\sqrt{m_\emptyset^2 + m_{12}^2 + m_{31}^2 + m_{23}^2}}, \quad d = s^2(m_\emptyset m_{0123} - m_{01}m_{23} - m_{02}m_{31} - m_{03}m_{12}),$$

all of which is straightforwardly transferred into code. Armed with this efficient orthogonal projection through renormalization, we are guaranteed to stay on the motor manifold even in the presence of numerical error.

Optimizing GA Expressions

To obtain optimal coefficient expressions from the concise and elegant mathematics above, an algebraic optimization (which can be automated using e.g. GAL [21] Grassmann.wl [4] or GAALOP [5]) is required. We illustrate the process for the derivative of the velocity bivector for the symmetric top. The GA formula we want to optimize is

$$\dot{B} = \star^{-1}(B \times \star B).$$

For a general bivector B with coefficients

$$B = b_{01}\mathbf{e}_{01} + b_{02}\mathbf{e}_{02} + b_{03}\mathbf{e}_{03} + b_{12}\mathbf{e}_{12} + b_{31}\mathbf{e}_{31} + b_{23}\mathbf{e}_{23}$$

we employed our soon to be released symbolic computation package `GAmphetamine` [7] to produce the expression:

$$\dot{B} = (b_{02}b_{12} - b_{03}b_{31})\,\mathbf{e}_{01} + (-b_{01}\,b_{12} - b_{03}b_{23})\,\mathbf{e}_{02} + (b_{01}\,b_{31} - b_{02}b_{23})\,\mathbf{e}_{03}.$$

Thus the expression for $\dot{B}$, involving two geometric products on a general bivector with several dualizitions actually simplifies to just 6 multiplications and 3 subtractions of scalar coefficients, for this case of uniform inertia.

As a second example let us consider the implementation of the forque of gravity. It required us to move a single fixed force in the global $\mathbf{e}_{02}$ direction into the body frame, according to $F_g = -9.81 \star(\widetilde{M}\mathbf{e}_{02}M)$ (which is eq.(2.31)). We calculated the optimal expression using the same `GAmphetamine` technique as above. This effectively results in:

$$2(m_\emptyset\, m_{23} + m_{12}m_{31})\,\mathbf{e}_{12} + (1 - 2(m_{12}^2 + m_{23}^2))\,\mathbf{e}_{31} + 2(m_{31} - m_\emptyset\, m_{12})\,\mathbf{e}_{23},$$

so in this case are left with just 9 multiplications and 4 additions.

It seems that with the right tooling, we can both have our lunch (geometric, concise, elegant, dimension-, metric- and basis-independent math), and eat it (optimal and efficient coefficient level expressions in all spaces).

Converting Inertia Tensors

We have so far focused on inertially symmetric objects of unit mass, which simplified our inertia tensor $\mathsf{I}_C[\,]$ to the standard PGA Hodge duality map. Modifying our set-up to properly incorporate full inertia is rather straightforward.

To convert a standard 3D inertia tensor to our inertial duality operator, we start from the diagonalized version of the inertia tensor. A list of 3D inertia tensors for a series of common shapes can be found on `https://en.wikipedia.org/wiki/List_of_moments_of_inertia#List_of_3D_inertia_tensors`.

Let us take the (rotational) inertia tensor of a solid cuboid of width w, height h, depth d, and mass m as example:

$$\mathsf{I}_C = \tfrac{m}{12}\begin{bmatrix} h^2+d^2 & 0 & 0 \\ 0 & w^2+d^2 & 0 \\ 0 & 0 & w^2+h^2 \end{bmatrix}.$$

We represent this inertia in 3D PGA as a bivector also including the translational inertia m, by eq.(2.20):

$$C = m\Big(\frac{h^2+d^2}{12}\,\mathbf{e}_{03} + \frac{w^2+d^2}{12}\,\mathbf{e}_{02} + \frac{w^2+h^2}{12}\,\mathbf{e}_{01} + \mathbf{e}_{12} + \mathbf{e}_{31} + \mathbf{e}_{23}\Big),$$

after which our inertial duality can be implemented using per coefficient multiplication:

```
var I = 1/12*mass*( (size[1]**2+size[2]**2)*1e01 +
                    (size[2]**2+size[0]**2)*1e02 +
                    (size[0]**2+size[1]**2)*1e03 + 12e12 + 12e13 + 12e23 ),
    A  = (x)=>x.Dual.map((x,i)=>x*(I[i]||1)),
    Ai = (x)=>x.map((x,i)=>x/(I[i]||1)).UnDual;
```

After this, simply make sure to use these maps `A` and `Ai` for the inertial duality in the function `dState` of Section 2.5.6. For these more detailed computations we need somewhat better precision, so it makes sense to replace the simple Euler integrator by RK4:

```
RK4 = (f,y,h)=>{
  var k1=f(y), k2=f(y+0.5*h*k1), k3=f(y+0.5*h*k2), k4=f(y+h*k3);
  return  y+(h/3)*(k2+k3+(k1+k4)*0.5);
};
```

and then to update the `State` by `State = RK4(dState,State,h);`.

2.6 Contact Forques

The previous examples were the mainstay of Classical Mechanics in a theoretical setting. In practice, there is more than one object, and it contacts others in collisions. That is where things get a bit ugly, since the phyiscs of contact involves material forces which are less easily modelled. The forces we have modeled so far in the implementation of Section 2.5 were being integrated numerically, and so they are bound to the time steps of the integration. This makes them not suited for modelling instant responses such as those required by Newton's third law. One option to resolve the issue is to model instantaneous collision response forces known as *impulses* explicitely. This allows those to bypass the integrator and update the velocity state directly. For the background of how collision is treated in this discrete time system, consult the description of the Wiki page on Collision Response [23]. It motivates why the 'impulse method' is a reasonable approximation. We basically take over from that page when the equations start.

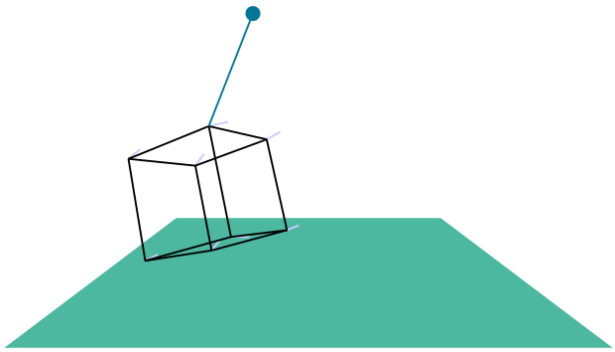


Figure 2.5: *A cube on a Hooke spring under gravity with contact.*

Let us have two bodies in a frictionless point/plane contact at Q at the time of collision. The local normal line at contact is $N = \mathbf{n} \cdot Q$, with $\mathbf{n}$ the local normal direction of the contact plane (the line N is given an orientation pointing 'out of' the plane contact object which is indexed with '+', and 'into' the point contact object which is indexed with '−'). A contact forque is generated, with associated contact impulse in the direction of the line N (note that this is actually a pure force, when we assume no friction). There is a proportionality factor j to be determined from the contact requirement on velocity before and after: only the normal component should change, it bounces back by a proportional 'coefficient of restitution' ρ.

The local velocities $V_\pm$ of the objects at the contact point are, before contact,[4]

$$V_\pm = Q \vee Q \times B_\pm.$$

After contact, the impulse J affects the velocity through the inertia of each object by $j\,\mathsf{I}_\pm^{-1}[N]$, so the velocities $V'_\pm$ after contact are given by

$$V'_\pm = Q \vee Q \times (B_\pm \mp j\,\mathsf{I}_\pm^{-1}[N]).$$

In the frictionless case we are assuming, the relationship between the velocities before and after is determined by how much the coefficient of restitution ρ affects the N-components of the velocities:

$$(V'_+ - V'_-) \cdot \widetilde{N} = -\rho\,(V_+ - V_-) \cdot \widetilde{N}$$

(we put in a reversion on N to avoid dimension-dependent signs). We can combine all equations to solve for the unknown j, to obtain:

$$j = -(1+\rho)\,\frac{\big(Q \vee Q \times (B_+ - B_-)\big) \cdot \widetilde{N}}{\big(Q \vee Q \times (\mathsf{I}_+^{-1}[N] - \mathsf{I}_-^{-1}[N]\big) \cdot \widetilde{N}} \tag{2.32}$$

With j thus found, replace the current $B_\pm$ of the state in which collision occurred with:

$$B_\pm \;\leftarrow\; B_\pm \pm j\,\mathsf{I}_\pm^{-1}[N] \tag{2.33}$$

[4]Let us make $\times$ have precedence over $\vee$, it avoids many brackets; since $Q \vee Q = 0$, it is not hard to remember that it applies.

and continue with whatever algorithm you used to solve the motion equations.

Since $N = \mathbf{n} \cdot Q$, one could write out eq.(2.32) for j in more detail and reduce both numerator and denominator to Euclidean dot products. That corresponds closer to our classical source [23]; see Section A.14 for details on this.

If the 'minus' object is infinitely heavy, its motion is not affected by the collision, so $V'_- = V_-$. Since it need not be static, its B_- is generally nonzero. In such a case, the motions are governed by the same equation eq.(2.32); you merely need to set $\mathsf{I}_-^{-1}[N] = 0$, to reflect the infinite inertia.

2.6.1 Implementation of Collision Response

Let us implement a simple situation of a convex object colliding with a plane. Then SAT collision detection is sufficient. Adopted to be dimension agnostic, the Separating Axis Test (SAT) algorithm tries to find a hyperplane that divides the space into two halves, each containing only points of one of the objects being tested for collision. It requires both objects to be convex, and considers each of the hyperplanes on the objects boundary as a potential separating axis. In our example, we are only testing for collisions between a moving object A and a fixed floor plane f. We can thus simply state that A is in collision with f if for any of the points A_i of A we have

$$A_i \vee f < 0.$$

Hence we can test if an object A that consists of a number of vertices A_i collides with our floor by finding at least one point for which the above expression is true.

```
// Find a point on the wrong side of the plane and return it.
function is_colliding(plane, points) {
  return points.find(point=>(point & plane)<0)
}
```

Using this, the code for updating the State is now changed using eq.(2.33):

```
var Q, N, Vm, rho=0.5, j;
for (var i=0; i<10; ++i) {
// Update the state
  var diff = 1/600 * dS(state);
  state = state + diff;
// Resolve collisions
  while (Q = colliding(state[0])) {
    state = state - diff;
    var [M,B] = state;          // shortcuts M and B for the state
    N = ((~M >>> floor) | Q);   // line at contact in body frame
    Vm = Q & comm(Q,B);         // Local velocity rate.
    j = -(1+rho) * (Vm | N) / ((Q & comm(Q, N.UnDual))|N);  // calculate impulse strenth j
    B.set( B + j*N.UnDual );    // update the velocity bivector.
    var diff = 1/600 * dS(state);
    state = state + diff;
  }
// Renormalize the state.
  state[0] = state[0].Normalized;
  state[1] = state[1].Grade(2);
}
```

Play with `https://enki.ws/ganja.js/examples/pga_dyn.html#Collision_Response` in the `ganja.js` coffeeshop, to see a cube bounce on a plane.

2.7 Kinetic Energy and Power

2.7.1 Duality of Rate and Momentum Bivectors

Dynamics is dual to kinematics. The bivector rates at which objects move can only be related to the (join) lines of forces through the inertia map; and we have seen that the inertia map is a form of dualization. It converts a bivector into a dual bivector, i.e., a $(d-1)$-vector (in the $(d+1)$-dimensional PGA, note that $(d+1)-2=d-1$). In 3D, the dual nature of the inertia map is a bit hidden, since then a $(d-1)$-vector is also a bivector. Even then, you can still sense its dual nature from the fact that in 3D the inertia map eq.(2.30) swaps Euclidean and ideal parts (with some scaling by mass and inertia).

So momentum P is dual to velocity rates B. This is a bit surprising (in classical linear motion, we view velocity $\mathbf{v}$ and momentum $\mathbf{p}=m\mathbf{v}$ as very similar elements of our vector algebra; as we have seen in PGA, the corresponding quantities $\boldsymbol{\epsilon}\mathbf{v}$ and $m\mathbf{v}\cdot Q$ are quite different). Forques, which are proportional to the time derivatives of momenta, are of course also dual to velocity rates: they are join lines (or join wrenches).[5]

The dual nature implies that there are combinations of rates and momenta, or rates and forques, that produce scalars. This leads us naturally to kinetic energy and power.

[5]The same dual distinctions occur in other 6D frameworks: screws and coscrews (for rates) and wrenches and co-wrenches (for forques) in Screw Theory; displacements and forces in Spatial Vector Algebra.

2.7.2 Kinetic Energy $T = \frac{1}{2}B \vee \mathsf{I}[B]$

The classical definition of kinetic energy of a mass point with velocity $\mathbf{v}$ is $\frac{1}{2}m\mathbf{v}^2$. In view of the above, let us rather view this as (half) the dot product of the velocity $\mathbf{v}$ and the momentum $\mathbf{p} = m\mathbf{v}$, and use that to define it in PGA.

There are several ways to produce a scalar from the rate B_b and the momentum $P = \mathsf{I}_b[B_b]$: we can take the join $B_b \vee \mathsf{I}_b[B_b]$, or the outer product $B_b \wedge \mathsf{I}_b[B_b]$ (which gives a scalar factor to the pseudoscalar quadvector $\mathcal{I}$).

These two methods are essentially the same construction, since meet and join are dually related. The outer product is perhaps more familiar, so we could define the PGA kinetic energy T through:

$$\text{kinetic energy } T\text{: } T\mathcal{I} \equiv \tfrac{1}{2}\, B_b \wedge \mathsf{I}_b[B_b]. \tag{2.34}$$

Section A.12 provides the connection to the classical formulation, and shows that both the linear and angular parts are included.

Alternatively and equivalently, we could define using the join:

$$\text{kinetic energy } T\text{: } T \equiv \tfrac{1}{2}\, B_b \vee \mathsf{I}_b[B_b]. \tag{2.35}$$

Being a scalar, the kinetic energy is invariant under the action of a motor, so either quantity has the same value in the world frame. (Of course this 'spatial-transformational invariance' does not imply that it is constant in time!)

2.7.3 Power $\Pi = B \vee F$

The work done on a point by a force involves the component of the force along the displacement it caused. The power Π, work per unit time, is classically computed from velocity and force as $\mathbf{v} \cdot \mathbf{f}$.

In PGA, where velocity and force at a point X are represented as bivector $B = \dot{\mathbf{x}} \cdot X$ and dual bivector $F = \mathbf{f} \cdot X$ we can compute this quantity directly from a meet or join expression. We define the *power* Π by:

$$\text{power } \Pi \text{ of total forque } F \text{ under motion by } B\text{: } \Pi\mathcal{I} = B \wedge F, \tag{2.36}$$

where we dropped the frame subscript since the expression is invariant under M. Or, if you prefer the join expression:

$$\text{power } \Pi \text{ of total forque } F \text{ under motion by } B\text{: } \Pi = B \vee F. \tag{2.37}$$

As for kinetic energy, we slightly prefer to compute with the meet (the familiar outer product) rather than the join.

> Our understanding of dynamics was much influenced by Gunn's thesis [17]. His purposeful non-metric and projective treatment makes it challenging reading, but we should connect our notation to his, for those interested. On page 110 of his thesis, Gunn defines the kinetic energy as $-\frac{1}{2}m\,\Gamma \vee \Pi$, with $\Gamma = V\mathcal{I}$ (with V the velocity). In his section 9.7, he defines total forque F as we do (and calls it Δ); of course $F = \dot{\Pi}$. Then the kinetic energy K has a derivative equal to the power

$\dot{K} = -B \vee F$ (denoted in [17] as $\dot{E} = -\Omega \vee \Delta$). The work done is then the integral of power times time. Gunn views $\Omega \vee \Delta$ as the *(virtual) work* – but this is per time unit, so it is actually power. Gunn has recently replaced the join expression by the somewhat more convenient dual expression $B \wedge \Delta$ (also used by [19]).

To convince yourself that the PGA power Π as defined by eq.(2.36) or eq.(2.37) is identical to the classical definition of the power of a linear and rotational motion, consult Section A.13.

2.7.4 Lagrangians

In this introductory tutorial, we have followed the Newton/Euler way of deriving the equations of motion directly from the geometry of the space in which the motion occurs. Lagrange developed a different way of deriving those equations, based on a lagrangian energy function $\mathcal{L}$ and its derivatives. This 'phase space method' is now often favored in expositions on Classical Mechanics, since it is better able to incorporate known symmetries, and as such also more natural for quantum mechanics. The lagrangian method is in principle coordinate-free, though often coordinates are used to express it. PGA should be very suitable to handle it (and the related Hamiltonian approach). Let us briefly denote how this can look in the free (forqueless) case.

We take the kinetic energy as the lagrangian in the forqueless case. We express it in the body frame, but drop the subscript to unclutter the equations.

$$\mathcal{L} = \tfrac{1}{2} B \vee \mathsf{I}[B].$$

This is a scalar quantity. In standard GA, many standard differentiation results exist for the scalar product, but not for the join $\vee$. We bring the lagrangian into 'scalar part' form, and observe that there are various symmetrical ways of doing so:

$$\mathcal{L} = \tfrac{1}{2}\, B \vee \mathsf{I}[B] = \tfrac{1}{2}\, \mathsf{I}[B] \vee B = \langle \tfrac{1}{2}\, B\, \mathcal{I}^r\, \mathsf{I}[B] \rangle = \langle \tfrac{1}{2}\, \mathcal{I}^r\, B\, \mathsf{I}[B] \rangle = \cdots,$$

where $\mathcal{I}^r$ is the reciprocal of the PGA pseudoscalar, $\mathcal{I}^r = \widetilde{\mathbf{I}}_d \boldsymbol{\epsilon}^r$. We thus had to invoke the 'dual space' to PGA to use this conversion technique, see [13].

We need to set up the lagrangian equations, which follows from the calculus of variations. Classically, variation is characterized through varying (generalized) coordinates, but in GA we can simply vary the motors of the orbits instead [9]. The lagrangian equations thus involve differentiating to the motor M corresponding to B. However, motors are normalized, so this would involve imposing a normalization condition. That can be incorporated using a Lagrange multiplier (as in the non-PGA treatment of GA lagrangian in [9] section 12.1.3), but we prefer to proceed more directly. We introduce a spinor ψ as the non-normalized M; then $B = -2\,\psi^{-1}\,\dot{\psi}$. Then invoking some standard differentiation results (see Exercise 21), we find that evaluating $\partial_\psi \mathcal{L} - \frac{d}{dt}\partial_{\dot{\psi}} \mathcal{L} = 0$ ultimately leads to:

$$\mathcal{I}^r\, \mathsf{I}[\dot{B}] = B \times (\mathcal{I}^r\, \mathsf{I}[B]) \;= \tfrac{1}{2} B\, \mathcal{I}^r\, \mathsf{I}[B] - \tfrac{1}{2} \mathcal{I}^r\, \mathsf{I}[B]\, B = \mathcal{I}^r\, (B \times \mathsf{I}[B]), \tag{2.38}$$

where we used that our lagrangian has a symmetry which allows us to substitute $\mathcal{I}^r B$ for $B\,\mathcal{I}^r$, so this is admissible in its solutions as well (even though in general $B \times \mathcal{I}^r \neq 0$). We thus indeed retrieve the PGA version of the forqueless Newton/Euler equations $\mathsf{I}[\dot{B}] = B \times \mathsf{I}[B]$ from the lagrangian method.

2.8 Connection to Other 6D Formalisms

We mentioned a few times that there are aspects to the PGA approach that are reminiscent of other frameworks to treat Newtonian dynamics, also recognizing that there is a 6D structure to the equations. We briefly treat three of them here, to show awareness and to point to bodies of literature that should contain interesting results in a usable format.

- **Screw Theory**
  In the nineteenth century, a group of British mathematicians (among them Ball, Study, Clifford) found a 6D formulation of classical mechanics in which we recognize much of the elements of 3D PGA, now known as Screw Theory. It was based on an algebraic encoding of lines, which as we have seen are central to modelling mechanics. The main difference between Screw Theory and PGA is that the basic elements are formulated in terms of pairs of 3D vectors (the Plücker coordinates), rather than as a single bivector. Special rules for the multiplication of these vector pairs, based on dual numbers (i.e., numbers of the form $a + b\epsilon$ with $\epsilon^2 = 0$), then need to be introduced, to obtain the proper algebra allowing one to use one 3D vector as an axis direction, and the other part as a moment.

  The modelling of the Newton-Euler equations then exhibits a similar unified pattern to that in PGA: there are screws called *twists* that are used to represent the velocity rates, and co-screws called *wrenches* used to model forces. The use of vector components feeds into a matrix-based representation (rather than the motors we use).

  Screw theory is of course effective, and has been used especially in robotics. But its reservation of vectors for the representation of screws makes those unavailable for the representation of geometrical objects. Therefore Screw Theory feels like an add-on, a dedicated framework for the treatment of motions only, to which one temporarily needs to transition to compute those aspects, and those only. It does not interact well with the modelling of objects.

- **Dual Quaternions**
  For an implementation, the motion operator description of Screw Theory boils down to the isomorphic structure of *dual quaternions*. In 3D, the 'quaternion' part is the known efficient algebra of 3D rotations; the 'dual' refers to dual numbers, of the form $a + b\epsilon$ with $\epsilon^2 = 0$. When you describe a motion state with coordinates of the form (q_1, q_2), to be interpreted as $q_1 + \epsilon q_2$ (with q_1 and q_2 quaternions), this data structure exhibits the right behavior at the coordinate level. Since quaternions are basic data types in many systems, this leads to efficient implementations.

  It is clear from literature, however, that many implementers do not really understand the geometrical structure of dual quaternions, and as a consequence perform nonsensical operations on them (additions, averaging, component selection, etc.), frequently ignoring the subtleties of non-commutation. Never perform operations at the coordinate level! PGA embeds the dual quaternions as its motors. Use those motors and their bivectors as elements, and perform only PGA operations between those entities – then you will not stray from the meaningful, and you will never need anything more to generate your code.

- **Spatial Vector Algebra**
  Roy Featherstone [15] also designed a 6D framework, called Spatial Vector Algebra. Compared to Screw Theory, it treats the elements more like unified 6D vectors than as 3D vector pairs. The framework encodes motions (displacements, rates) and forques in two 6D spaces with a dot product between them to produce the scalar quantity representing physical work. The separation into two 6D spaces accomplishes what PGA would view as the Hodge duality within the bivector space of 3D PGA, and it allows representing the inertia tensor as a symmetric matrix. It comes at a price: Spatial Vector Algebra does need to introduce two cross products, one for motion element times motion element and for motion element times spatial force element (which in PGA are both the same commutator product, applied to different types of bivectors).

  Featherstone explicitly observes the additivity of physical quantities like acceleration and inertia in the framework, and how that waives the need for parallel axis theorems. He shows [14] how using a 6D framework avoids having add a Coriolis term in the acceleration, yet produces correct physics. All these advantages are consequences of describing things in 6D, and also apply to PGA.

  But just like the screws in Screw Theory, spatial vectors are a special encoding of motions, disjoint from the representation of the geometrical objects that should be moved.

In our view, both frameworks lack the natural simplicity of PGA, following from the insight that we can use vectors to represent basic elements of geometry (planes), and then employ the automatically present bivectors and dual bivectors of the algebra to encode their motions. We believe that this integration with objects makes the algebra of motions more accessible, not a seemingly disjoint highly specialized single-purpose add-on, and that our demonstrations show this.

Many useful practical techniques have been developed in Screw Theory and Spatial Vector Algebra, in a manner and language that can be transferred to 3D PGA without too much trouble (and in that transferred form, they should work in any dimensionality). An example of this was given in our treatment of constrained motion in Section 2.9. We fully expect that the study of Spatial Vector Algebra and Screw Theory will yield a fruitful harvest for applied PGA, and that their re-incorporation into PGA will extend the scope of those earlier frameworks.

2.9 Constrained Motion

Often, objects are constrained in their motion, by being attached to other objects in some way - such as a rotational hinge. This modifies how much of the total forque exerted on the object can actually be used for the motion. Within the framework, we can treat this in a structural manner. We briefly suggest how this can be incorporated, taking our inspiration from the Spatial Vector Algebra approaches. A general GA approach to the same issues may be found in [19].

2.9.1 6D Approaches to Constraints

We treat an example given in [16], in the form of Figure 2.6. It involves a body 1 with inertia I_1, acted upon by a total forque F – but this object is connected to a second body 2 with inertia I_2, via a rotational axis L. As a consequence of this hinge, not all of F will move body 1, it will somehow have to drag the piggy-backed body 2 along.

The tutorial reference [16] solves this setup in both the classical 3D approach, and in the 6D Spatial Vector Algebra approach which that paper advertises. In its main text, it clearly spells out the advantages of the 6D approach, and graciously allows that those same advantages apply to all related 6D approaches. The paper is highly recommended, since all such considerations indeed also apply to the PGA of 3-dimensional space – which is 6D in its bivectors. Let us translate this Spatial Vector Algebra example literally into PGA, to show how to convert between the frameworks.

As we mentioned above, the main difference is that rather than the 6D vectors of Spatial Vector Algebra, PGA simply uses the 6-dimensional space of bivectors from 4D PGA. We then do not need to maintain two dual 6D spaces of vectors, one for displacements and one for forques, related by a dot product that produces the power, and each involving a slightly different commutator product. For us, one 6D space of bivectors with a single product suffices, and the PGA algebra does the administration of the modeled concepts automatically. As we have seen, doing so still maintains the physically relevant distinction between 'direct' bivector elements such as rates and '(Hodge) dual' bivector elements such as momenta and forques.

Let the hinge axis be characterized by the meet line 2-blade L (rather than the spatial vector $\mathbf{s}$ in Figure 2.6), and its forque (or wrench) transfer by F_L, a (dual) bivector. Then we can translate the first set of equations from the figure to PGA as

$$\begin{aligned} F - F_L &= \mathsf{I}_1 \dot{B}_1 \\ F_L &= \mathsf{I}_2 \dot{B}_2 \\ \dot{B}_2 &= \dot{B}_1 + \alpha L. \end{aligned}$$

The zero-work constraint for the forque is not just a dot product (as in the Spatial Vector Algebra framework), but expressed by the join with the meet line L:

$$L \vee F_L = 0. \tag{2.39}$$

The solution is then, following very analogous steps:

$$\begin{aligned} F_1 &= \mathsf{I}_2[\dot{B}_1 + \alpha L] \\ 0 &= L \vee \mathsf{I}_2[\dot{B}_1 + \alpha L] \\ \alpha &= -\frac{L \vee \mathsf{I}_2[\dot{B}_1]}{L \vee \mathsf{I}_2[L]} \\ F_L &= \mathsf{I}_2[\dot{B}_1 - L\,\frac{L \vee \mathsf{I}_2[\dot{B}_1]}{L \vee \mathsf{I}_2[L]}] \\ F &= \big(\mathsf{I}_1 + \mathsf{I}_2 - \frac{L}{L \vee \mathsf{I}_2[L]}\, L \vee \mathsf{I}_2\big)[\dot{B}_1]. \end{aligned} \tag{2.40}$$

The outcome completely corresponds with S39 in Figure 2.6, both in outcome and derivation.

Solving a Two-Body Dynamics Problem Using Spatial Vectors

We are given a rigid-body system consisting of two bodies, B_1 and B_2, connected by a revolute joint [S2]. The bodies have inertias of I_1 and I_2, respectively, and they are initially at rest. The joint's rotation axis is s. A force f is applied to B_1, causing both bodies to accelerate. The problem is to calculate the acceleration of B_1 as a function of f (Figure S2).

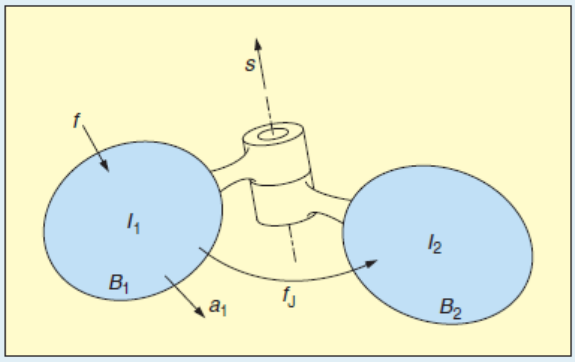


Figure S2. Problem diagram using spatial vectors.

Solution

Let a_1 and a_2 be the accelerations of the two bodies, and let f_J be the force transmitted from B_1 to B_2 through the joint. The net forces acting on the two bodies are therefore $f - f_J$ and f_J, respectively, and their equations of motion are

$$f - f_J = I_1 a_1 \tag{S33}$$

and

$$f_J = I_2 a_2. \tag{S34}$$

There are no velocity terms because the bodies are at rest. The joint permits B_2 to accelerate relative to B_1 about the axis specified by s; so, a_2 can be expressed in the form

$$a_2 = a_1 + s\alpha, \tag{S35}$$

where α is the joint acceleration variable. Again, there are no velocity terms because the bodies are at rest. This motion constraint is implemented by f_J, which is the joint constraint force, so f_J must satisfy

$$s^T f_J = 0, \tag{S36}$$

i.e., the constraint force does no work in the direction of motion allowed by the joint.

Given (S33)–(S36), the problem is solved as follows. First, substitute (S35) into (S34), giving

$$f_J = I_2 (a_1 + s\alpha). \tag{S37}$$

Substituting (S37) into (S36) gives

$$s^T I_2 (a_1 + s\alpha) = 0,$$

from which we get the following expression for α:

$$\alpha = -\frac{s^T I_2 a_1}{s^T I_2 s}. \tag{S38}$$

Substituting (S38) back into (S37) gives

$$f_J = I_2 \left(a_1 - \frac{s s^T I_2}{s^T I_2 s} a_1 \right),$$

and substituting this equation back into (S33) gives

$$\begin{aligned} f &= I_1 a_1 + I_2 a_1 - \frac{I_2 s s^T I_2}{s^T I_2 s} a_1 \\ &= \left(I_1 + I_2 - \frac{I_2 s s^T I_2}{s^T I_2 s} \right) a_1 . \end{aligned}$$

The expression in brackets is nonsingular and may therefore be inverted to express a_1 in terms of f:

$$a_1 = \left(I_1 + I_2 - \frac{I_2 s s^T I_2}{s^T I_2 s} \right)^{-1} f. \tag{S39}$$

Reference

[S2] R. Featherstone. (2010). Spatial vector algebra [Online]. Available: http://users.cecs.anu.edu.au/~roy/spatial/

COURTESY OF R. FEATHERSTONE, 2010.

Figure 2.6: *From [16]: the dynamics of a robot joint employing the Spatial Vector Algebra formalism. The PGA solution can follow this almost exactly at the algebraic level. Featherstone's work is a great inspiration for more such solutions.*

2.9.2 PGA Constraint Geometry

The Spatial Vector Algebra approach of Figure 2.6 looks somewhat more straightforward, since it can use a dot product in the solution rather than the unfamiliar join product $\vee$. Expressing the dot product as a transpose then combines well with the matrix encoding of the inertias. The price for this dot product is the need to maintain two 6-dimensional spaces, for motion rates and for forques. Numerically, the computation on the coefficients is exactly the same in both frameworks, and equally efficient.

To write the solution eq.(2.40) more compactly, we can define a special scalar product ruled by the inertia I_2 of object 2, to be used between two bivectors X and Y. In this, I_2 plays the role of a metric.

$$X *_2 Y \equiv \widetilde{X} \vee \mathsf{I}_2[Y]. \tag{2.41}$$

Then $L *_2 L$ measures an 'inertial norm' of the line.

We now define the linear bivector-valued bivector function $\mathsf{R}_L[\,]$ as

$$\begin{aligned}\mathsf{R}_{L_2}[X] &\equiv X - \frac{L}{\widetilde{L} \vee \mathsf{I}_2[L]}\, \widetilde{L} \vee \mathsf{I}_2[X] \\ &= X - \frac{L}{L *_2 L}\, L *_2 X \\ &= X - L^{-1}\,(L *_2 X).\end{aligned}$$

We recognize this as the *rejection* of the quantity X by the axis L, in terms of that new scalar product based on the inertia of body 2. Then the expression for F can be written compactly in terms of the linear maps I_1, I_2 and R_{L_2} as

$$F = (\mathsf{I}_1 + \mathsf{I}_2 \circ \mathsf{R}_{L_2})[\dot{B}_1],$$

indicating that part of the acceleration gets absorbed by L, and only the rejected remainder feels the inertia of body 2. The given F has to cause that motion, as well as the motion of body 1. We solve for $\dot{B}_1$ in closed form as

$$\dot{B}_1 = \big(\mathsf{I}_1 + \mathsf{I}_2 \circ \mathsf{R}_{L_2}\big)^{-1}[F]. \tag{2.42}$$

This is thus the rate acceleration of body 1, modified from what F would give by the presence of body 2 through the axis joint L.

There is a great contrast of either of these two compact solutions (in Spatial Vector Algebra, or in PGA) with the much more involved classical approach. The differences are very well enumerated in [16] for Spatial Vector Algebra when it shows the classical solution explicitly; and they apply just as well to PGA.

2.9.3 Constraints in General

For general constraints, we would propose to convert the algebraic methods from Spatial Vector Algebra [15] to PGA, since they are the best around, and in their 6D framework very closely related to PGA.

General constraints between two bodies can then be modelled locally as linear conditions on the relative rate bivector, requiring it to lie (locally) in a linear subspace S of the bivector space which may be represented by a k-blade S, for k constraints. If we provide the bivector space with a basis $\mathbf{s}_i$, then the constraint space S is a linear combination of those basis vectors.

$$(B_2 - B_1) \wedge S = 0.$$

Both B_i and S are meet lines in this equation.

The algebraic and computational processing of such constraints should closely follow the methods of Spatial Vector Algebra. It may be that the ability to combine constraints using meet and join provides some additional geometrical insights, but this remains to be explored. Certainly the matrix methods of Spatial Vector Algebra should be mimicked in PGA implementations, since they are highly efficient. Details of those may be found in [15]; a treatment of multiple constraints in a GA setting is [19].

2.10 Wrapping Up (for Now)

We hope to have demonstrated, on paper and on your screen, that PGA is a very attractive framework for dynamics. Its effective 6D motion description at the (dual) bivector level was found before, as the key algebraic structures behind Screw Theory and Spatial Vector Algebra, but the way in which PGA integrates that physical motion with the representation of the objects that actually do the moving is truly unique and useful.

2.10.1 Dimension-Agnostic by Complementary Representation

We have moreover found that the specification of typical situations (like the demo of a damped Hooke spring under gravity) can be done in a manner that is dimension-independent: literally the same program works in any dimensionality. The actually required dimension is conveyed in the initial conditions (such as where the starting state is relative to a gravity direction), not in the program's operations.

The reason that this could be done is that PGA represents all situations in a dual manner, i.e., by orthogonal complementation. When you state that gravity works in a direction $\mathbf{e}_1$, we encode that by using the bivector $\boldsymbol{\epsilon} \wedge \mathbf{e}_1$. For us, the vector $\mathbf{e}_1$ is a plane with normal vector $\mathbf{e}_1$; this exists in all higher dimensions. Only if you want to draw it do you care how many dimensions to fill; but this is display, not algebra. The dual description by a normal vector is sufficient; the actual plane will fill all the other dimensions $\mathbf{e}_2, \mathbf{e}_3, \cdots$ of any space properly, and as required. This holds whether you realized that you would go to higher dimensions when you designed your program, or not (we show an example of this in the tricycle demo below). There is no need to adapt that first low-dimensional description as we find ourselves in more dimensions later on. (This contrasts with the common usage of having the vertical direction be the 'last coordinate': $\mathbf{e}_2$ in 2D, $\mathbf{e}_3$ in 3D - which such inconvenient conventions for the most important problem dimension, it is no wonder that a program for 2D does not extend easily to 3D...)

The effect of this dual representation extends beyond the forces – it also helps the kinematics.

When we need to develop a demo of a tricyle moving in a plane, we first draw a 2D picture in [13], as in the left picture in Figure 1.2. We find the steering pivoting around a point by angle ϕ, and naturally find the center of rotation as the meet of the lines $f(\phi)$ and r of the front and rear axles. That center of rotation is a point; and because lines are our vectors in 2D, the meet $r \wedge f(\phi)$ is a 2-vector, and the rotation around the center of rotation is generated by a motor that is its exponential $\exp(r \wedge f(\phi)\, t)$. All this can be expressed through $\boldsymbol{\epsilon}$, $\mathbf{e}_1$ and $\mathbf{e}_2$, as the normal vectors of the 2D lines involved (which is a dual, i.e., orthogonal specification). Running a tricycle is a 2D problem.

Having done the essential analysis in 2D, we wanted to implement in 3D, to have a nicely realistic tricycle. In 3D, rotations require an *axis*, not a point. We can find this axis by setting up vertical planes ruled by the lines of the front and rear axles, and intersect those to get the rotation axis through the center of rotation. This would appear to be a new, and somewhat involved, re-computation of the 2D case. But it is neither.

In PGA, the specification in 2D of the axle lines r and $f(\phi)$ was done by their line normals, in 2D PGA involving the vectors $\mathbf{e}_1, \mathbf{e}_1$ and $\boldsymbol{\epsilon}$. Those same normal vectors, reinterpreted in 3D PGA, now represent the corresponding vertical planes. In that 3D interpretation of the algebraic computation, the meet $r \wedge f(\phi)$ is literally the intersection of two planes r and $f(\phi)$, a 2-vector which is a 3D axis of rotation. The 3D motor is thus $\exp(r \wedge f(\phi)\, t)$. It is exactly the same PGA formula as what we already computed in the 2D case, in form and contents, involving only the vectors $\mathbf{e}_1$ and $\mathbf{e}_2$ of the original 2D specification. In 3D, it generates the motion parallel to that 2D plane as embedded in 3D – just as we wanted. Exactly the same algebraic element obtains the correct geometric meaning adjusted to the available dimensionality.

So if you use the complementary specification, solving a problem in the lowest dimension in which it can sensibly be defined also solves it for all higher dimensions. It solves it *literally*: the program for 2D is identical to the program for 3D; only the graphics display part of the code needs to be aware of the dimension in which you ultimately desire to convey the computed results.

2.10.2 Covariantly Moving Objects

In [13], we clarified that PGA is 'Plane-based Geometric Algebra', and we had a different reason for motivating the use of (dual) planes, rather than points, as the basic elements of our description of Euclidean geometry.

We first adopted the Kleinian general view of seeing the motions ('symmetries') of a geometry as fundamental (so, *not* the moving objects themselves); then we were inspired by Cartan-Dieudonné to encode motions as multiple reflections. We recalled that Clifford's geometric product can perform reflections in an equivariant manner by sandwiching. And with the realization that *Euclidean motions can be made by multiple reflections in planes*, it became natural to set things up as the geometric algebra of a vector space in which the *algebraic vectors are geometric planes*. That gave us code that was capable of moving any element that could be made as a linear combination of geometric products.

In a second step, we then realized that the invariant and equivariant elements of Euclidean geometry could be made from the symmetric and anti-symmetric combination of their invariants under the geometric product. The latter defined the meet of planes (which produces a 3D line as the meet of two planes, a 3D point as the meet of three planes, etc.). We also found it convenient to define a dual operation to the meet, which allowed us to compose elements as the join (for instance, a line in d-D as the join of two points, themselves represented as d-vectors). Deep down, those derived 'products' are all indeed expressible as linear combinations of geometric products; therefore elements made as meet and join of (hyper)planes move (anti-)equivariantly.[6] That greatly simplifies code: only the universally applicable operations of sandwiching and projection remain.

2.10.3 PGA: Two for the Price of One

It is a pleasant surprise that the two design criteria: *dimension-free specification of physical situations* (such as: the instantaneous motion of a mass points occurs along a line, whatever the dimension), and *equivariant representation of moving elements* (founded in the Cartan-Dieudonné view), both can be met by the same PGA approach. The former leads to dual bivectors as the way to encode momenta and forques, the latter leads to (hyper)planes as basic descriptors. The two coexist in PGA by its inherently dual structure. Since meet, join and sandwich product are all founded in the same geometric product, one obtains a unified computational framework that can be used immediately to implement the physical situation (as we demonstrated in our examples).

2.10.4 Other Physical Geometries

We have focussed on Euclidean geometry, since it is obviously useful in description and depiction of the physical world at our human scale. But these same principles of organizing one's representation extend beyond that.

1. Charles Gunn [17] has shown how to use the structure of PGA to encode dynamics in a metric-free way (by pinpointing the non-metric duality underlying it, rather than using the metric duality we used). This extends Euclidean dynamics of the PGA $\mathbb{R}^{d,0,1}$ naturally to elliptic dynamics in the PGA $\mathbb{R}^{d+1,0,0}$ and hyperbolic dynamics in the PGA $\mathbb{R}^{d,1,0}$. In all these PGAs, the equations of motion are the same, by judicious use of projective duality.

2. The kinematics of conformal geometry is treated in a different algebra called CGA (conformal geometric algebra). But CGA can easily be seen as an extension of PGA, where we use dual spheres rather than dual planes as vectors. This requires one more orthogonal representational dimension, but planes are still there, as spheres passing through infinity. So CGA is fully backwards compatible with PGA. The converse point of view is articulated well in [8]. Conformal dynamics has not been developed yet, but could (and we would not be surprised to find Maxwell taking over from Newton there).

[6] It was not mentioned in [13], but the join transforms anti-equivariantly: it requires a minus sign for odd reflections (relative to reflecting a meet element). This is actually related to its orientation type, which is internal. More about this in a future version of [13].

3. Space-time algebra (STA) has the Minkowski space as its playground; its motors cause the Lorentz transformations as its kinematics. A PGA version of STA (to be called STAP) is being developed by Martin Roelfs; spin promises to be integrated very nicely.

4. The treatment of other physical symmetries, such as SU(3), can similarly benefit from the Cartan-Dieudonné treatment, see [22]. Once in GA, an orthogonal factorization of the Lie algebra is simply one of the natural bivector decompositions; but it produces a new result for the analysis of Gell-Mann operators.

5. Projective geometry in 3D attains a canonical form in the geometric algebra $\mathbb{R}^{3,3}$, of the space of lines and screws: projective transformations become versors [12]. It should be possible to integrate this naturally with PGA, where meet lines are its bivectors. This remains to be done.

6. Some GA proposals, such as the representation of quadrics [2], do not put the motions central, but instead appear to be satisfied to represent the objects by themselves by meet and join. This leads to very high-dimensional algebras of which the symmetries are much more extensive than the usual collineations – and these are then not really used at all. We feel that such a 'waste of space' confirms that the 'object-first' method to design an algebra is the wrong way around...

We find it very refreshing to dive into these other geometries with these novel organizational principles in mind. Many advanced results can be better understood in their light; other results can be simplified to their essence; and some completely new results are beginning to be found. The PGA of Euclidean geometry and Newtonian physics is sufficiently rich and concrete to serve as a guiding model for inspiration.

Appendix A

Correspondence to Vector CM

When writing this tutorial text, we found ourselves checking in detail that the very lean PGA expressions indeed contain all vector relationships (and more) in the usual way of teaching Classical Mechanics. This was originally interwoven with the PGA explanations of the main text, which actually obscured the subject. But since many of our readers may have the same atavistic need of establishing the relationships with 'the way we learned it', we decided to enclose such material anyway. It forms the next two appendices of this tutorial. Read them in correspondence with the PGA material (in the PDF, click a link, and when done Alt-RightArrow to return), and you will hopefully find that it clarifies both frameworks.

A.1 Breaking Equivariance by Splitting

Classical mechanics uses an algebra of vectors that is not capable of expressing things without assuming an origin. Unfortunately, that origin may vary with circumstances or needs (from world origin to body centroid, for different bodies) and it is not in the formulas but only in the text; therefore the vector-based formulas are incomplete by themselves, and require following certain conventions to apply them correctly (like the addition of extraneous terms by hand, using facts like the parallel axis theorem).

But changes of viewpoint are quite easy in the equivariant geometric approach of PGA, and all is fully encoded in the algebraic expression of the geometrical entities. We see the classical formulas appear as certain factors of terms within the invariant expressions, when we pick suitably chosen Euclidean splits and point splits corresponding to the classical changes of viewpoint. PGA moreover allows us to work directly with the physical quantities, without splitting them.

This chapter collects the various splits of the PGA quantities to show that and how the classical formulation is subsumed; you then clearly see how nothing is lost, but much is gained by the invariant formulation.

A.2 Velocity

We may compute the time derivative of a point as eq.(2.10) by commutation with a general bivector of the Euclidean split form $B_w = \mathbf{B}_w + \boldsymbol{\epsilon}\mathbf{v}_w$ according to eq.(2.8):

$$\begin{aligned}\dot{X} &= X \times B_w \\ &= (O + \mathbf{x}\mathcal{I}) \times (\mathbf{B}_w + \boldsymbol{\epsilon}\mathbf{v}_w) \\ &= \tfrac{1}{2}(O\,\boldsymbol{\epsilon}\mathbf{v}_w - \boldsymbol{\epsilon}\mathbf{v}_w\,O + \mathbf{x}\mathcal{I}\,\mathbf{B}_w + \mathbf{B}_w\,\mathbf{x}\mathcal{I}) \\ &= (\mathbf{v}_w + \mathbf{x}\cdot\mathbf{B}_w)\,\mathcal{I}. \end{aligned} \tag{A.1}$$

This should be equal to $\dot{X} = \dot{\mathbf{x}}_w\mathcal{I}$ of eq.(2.10), so we find that, in terms of the world frame rate bivector $B_w = \mathbf{B}_w + \boldsymbol{\epsilon}\mathbf{v}_w$, the velocity vector of the point motion is

$$\dot{\mathbf{x}}_w = \mathbf{v}_w + \mathbf{x}\cdot\mathbf{B}_w. \tag{A.2}$$

This structure conveys the classical meaning of the Euclidean split components of velocity rate $B_w = \mathbf{B}_w + \boldsymbol{\epsilon}\mathbf{v}_w$: a translational motion by $\mathbf{v}_w$, and a rotating motion by $\mathbf{B}_w$ converted to classical 3D vector notation, $\mathbf{x}\cdot\mathbf{B}_w$ is the cross product $\mathbf{b}_w \times \mathbf{x}$ of rotation axis vector $\mathbf{b}_w\mathbf{I}_3 \equiv \mathbf{B}_w$ and location vector $\mathbf{x}$, which is the familiar classical expression for rotational speed. Check: $\mathbf{x}\cdot\mathbf{B}_w = \mathbf{x}\cdot(\mathbf{b}_w\mathbf{I}_3) = (\mathbf{x}\wedge\mathbf{b}_w)\,\mathbf{I}_3 = \mathbf{b}_w \times \mathbf{x}$ (where the $\times$ denotes the classical cross product).

A.3 Body Inertia

In eq.(2.14) we defined the PGA inertia $\mathsf{I}_b[]$. Let us see how this relates to the classical inertia in GA bivector form, which is

$$\mathsf{I}_C[\mathbf{B}_b] \equiv \Sigma_i\, m_i\, \mathbf{r}_i \wedge (\mathbf{r}_i \cdot \mathbf{B}_b). \tag{A.3}$$

We introduce the Euclidean split $B_b = \widetilde{M}B_wM = \mathbf{B}_b + \boldsymbol{\epsilon}\mathbf{v}_b$ for the bivector experienced in the body frame, as in Section 2.2.4.

$$\begin{aligned}\mathsf{I}_b[B_b] &\equiv \widetilde{M}\left(\Sigma_i\, m_i\, X_i \vee (X_i \times (MB_b\widetilde{M})\right) M \\ &= \Sigma_i\, m_i\, (\widetilde{M}X_iM) \vee \left((\widetilde{M}X_iM) \times B_b\right) \\ &= \Sigma_i\, m_i\, (O + \mathbf{r}_i\mathcal{I}) \vee \left((O + \mathbf{r}_i\mathcal{I}) \times B_b\right) \\ &= \Sigma_i\, m_i\, O \vee \left(O \times B_b\right) + 0 + \Sigma_i\, m_i\, (\mathbf{r}_i\mathcal{I}) \vee \left((\mathbf{r}_i\mathcal{I}) \times B_b\right) \\ &= m\, O \vee (O \times B_b) + \Sigma_i\, m_i\, (\mathbf{r}_i\mathcal{I}) \vee \left((\mathbf{r}_i\mathcal{I}) \times \mathbf{B}_b\right) \\ &= m\, O \vee \left(O \times (\boldsymbol{\epsilon}\mathbf{v}_b)\right) - \Sigma_i\, m_i\, \mathbf{r}_i \wedge (\mathbf{r}_i \cdot \mathbf{B}_b)\,\mathcal{I} \\ &= m\, \mathbf{v}_b \cdot O - \mathsf{I}_C[\mathbf{B}_b]\,\mathcal{I}. \end{aligned} \tag{A.4}$$

We see that $\mathsf{I}_b[B_b]$ contains components relating to the classical mass in the momentum $m\mathbf{v}_b$, and includes the classical bivector inertia $\mathsf{I}_C[\mathbf{B}_b]$ as its non-Euclidean part. Note that eq.(A.4) is both a 'Euclidean split' (see eq.(1.6)) and a 'point split' relative to the origin $O = C$ of the body frame (see eq.(1.8)).

We will also need the inverse of the body inertia. From the above, this may be specified as:

$$\mathsf{I}_b{}^{-1}[\mathbf{v}\cdot O - \mathbf{B}\mathcal{I}] = \boldsymbol{\epsilon}\mathbf{v}/m + \mathsf{I}_C^{-1}[\mathbf{B}], \tag{A.5}$$

where we parametrized the Euclidean split of its argument in a particularly useful form.

A.4 Classical Inertia as Bivector Map

The GA version of the classical inertia operator [20, 9],

$$\mathbf{B} \;\;\rightarrow\;\; \mathsf{I}_C[\mathbf{B}] \equiv \Sigma_i\, m_i\, \mathbf{r}_i \wedge (\mathbf{r}_i \cdot \mathbf{B}),$$

converts a Euclidean bivector $\mathbf{B}$ into another Euclidean bivector $\mathsf{I}_C[\mathbf{B}]$. So it is a mapping that changes a PGA origin hyperline (or axis) into another origin hyperline. The consequence is that when we turn an object around the axis $\mathbf{B}$, the lopsidedness of the object actually contributes to the angular momentum along a transformed axis. Every mass point contributes to this conversion, but the heavier points further away from the centroid affect it more.

Figure A.1 provides an illustration of the contribution of a single mass point with mass m_i at relative location $\mathbf{r}_i$ (relative to the body centroid at $\mathbf{c}$, through which $\mathbf{B}$ passes in this classical Euclidean setup). In the expression $\mathbf{r}_i \wedge (\mathbf{r}_i \cdot \mathbf{B})$, the input line $\mathbf{B}$ is brought to lie in the plane orthogonal to position vector $\mathbf{r}_i$, by tilting it in the plane $\mathbf{r}_i \cdot \mathbf{B}$ passing through $\mathbf{B}$ and the position vector $\mathbf{r}_i$. The magnitude of the contribution to the remapping by the mass m_i is proportional to $m_i\mathbf{r}_i^2$; so (for a fixed direction) the distance $\|\mathbf{r}_i\|$ of the mass point to the centroid is even more relevant than its mass m_i in determining how much the input line $\mathbf{B}$ changes due to the mass point: points away from the rotation axis contribute more.

A small note on the 'naturalness' of PGA: in the usual 3D GA treatment in a 3D vector space, we always identify the 2-blade $\mathbf{B}$ with an invariant plane (since 2-blades represent planes through the origin in that algebra), and then have to make the argument that this rotation plane is more natural as a characterization of a rotation than the usual rotation axis (which is then dual to it). In 3D PGA, the bivector $\mathbf{B}$ is immediately the representation of a meet line through the origin (it is the meet of two hyperplanes), so we can revert to simply seeing the 2-blade $\mathbf{B}$ as the representation of the invariant rotation *axis* rather than of the invariant rotation *plane*. In 3D, the exponential of the invariant axis is the rotation motor around it. PGA also clears up the situation for 2D, where a rotation still keeps a 2-dimensional subspace invariant. Now its rotation rate 2-blade is the PGA representation of an invariant *2D point*, the center of rotation. Thus PGA has no need to introduce the artificial concept of an extraneous 3D axis to describe rotations in 2D: a 2D axis is a point, automatically, and you do not need more to describe the motion it generates by straightforward exponentiation.

A.5 Eigenstructure of the Classical Inertia $\mathsf{I}_C[\,]$

The seemingly involved classical inertia map in the body frame, $\mathsf{I}_C[\mathbf{B}] \equiv \Sigma_i\, m_i\, \mathbf{r}_i \wedge (\mathbf{r}_i \cdot \mathbf{B})$, can be characterized compactly. That fact is actually the reason why we went to the body frame: it is easy to specify inertia there when you know your object. This $\mathsf{I}_C[\,]$ is a linear mapping, and it is symmetric, in the sense that $\mathbf{A} \cdot \mathsf{I}_C[\mathbf{B}] = \mathbf{B} \cdot \mathsf{I}_C[\mathbf{A}]$. (Exercise 4 in Section D asks

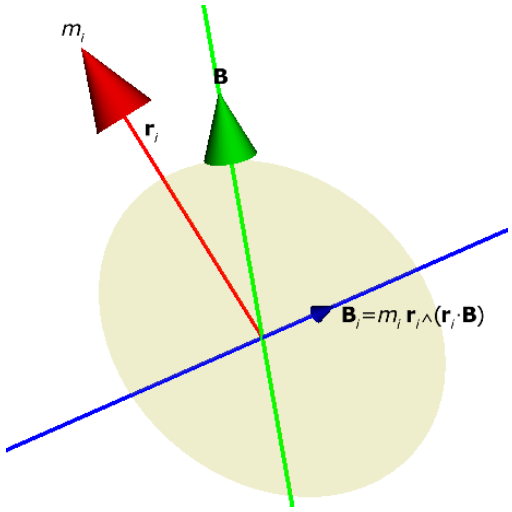


Figure A.1: *The geometry of the classical inertia map in 3D. A mass m_i at relative location $\mathbf{r}_i$ rotating around an axis $\mathbf{B}$ gives a contribution $\mathbf{B}_i = m_i\,\mathbf{r}_i \wedge (\mathbf{r}_i \cdot \mathbf{B})$ to the inertia map $\mathsf{I}_C[\mathbf{B}]$, perpendicular to $\mathbf{r}_i$ in the common plane through $\mathbf{r}_i$ and $\mathbf{B}$ (denoted in yellow). Adding these contributions for all points gives the angular momentum axis, which therefore usually differs in its direction from the rotational axis $\mathbf{B}$.*

you to show this.) Standard linear algebra then informs us that we can perform a spectral decomposition on $\mathsf{I}_C[\,]$: we can find three orthogonal 2-blades (i.e., PGA origin lines) $\mathbf{E}_j$ that are eigenblades of $\mathsf{I}_C[\,]$, and we can find their eigenvalues i_j. (Here our notation in 3D is that $\mathbf{E}_3 = \mathbf{e}_1\mathbf{e}_2$, and cyclic.) The corresponding equation,

$$\mathsf{I}_C[\mathbf{E}_j] = \mathsf{i}_j\mathbf{E}_j,$$

can be solved by standard linear algebra techniques (notably by viewing $\mathsf{I}_C[\,]$ as a matrix on a vector space formed by the coordinate 2-blades and performing an eigenanalysis).

Then in this frame adapted to $\mathbf{E}_j$, the inertia map can be written as an eigenvalue-weighted sum of components:

$$\mathsf{I}_C[\mathbf{B}] = \Sigma_j\,\mathsf{i}_j\,(\mathbf{B}\cdot\widetilde{\mathbf{E}}_j)\,\mathbf{E}_j.$$

(In LA terms, the matrix of $\mathsf{I}_C[\,]$ can be *diagonalized*.) For the particular form of the map $\mathsf{I}_C[\,]$, all eigenvalues are non-negative, and so the minimum inertia is achieved for a $\mathbf{B}$ aligned with the $\mathbf{E}_j$ of the smallest eigenvalue i_j. This works unchanged in d-dimensional space; $\mathsf{I}_C[]$ is always a linear symmetric map, with a d-dimensional eigenbasis of orthogonal 2-vectors.

A.6 Full Eigenstructure of the Body Inertia

In PGA, the classical body inertia $\mathsf{I}_C[\,]$ term is only the rotational part of the total inertia $\mathsf{I}_w[\,]$. But in full PGA, the inertia map also contains a translational consequence to having mass moving: it is called (linear) momentum. (In d-dimensional PGA, there are d more

translations, to augment the $\binom{d}{2}$ directional inertias, for a total of $\binom{d+1}{2}$ bivectors.) Let us study the body frame version $\mathsf{I}_b[B_b]$ of that total PGA inertia $\mathsf{I}_w[\,]$ and its eigenstructure.

As we derived in eq.(A.4), the total PGA inertia in the body frame at the centroid looks like the map

$$B_b = \mathbf{B}_b + \boldsymbol{\epsilon}\mathbf{v}_b \quad \mapsto \quad \mathsf{I}_b[B_b] \equiv m\mathbf{v}_b \cdot O - \mathsf{I}_C[\mathbf{B}_b]\,\mathcal{I}. \tag{A.6}$$

This PGA inertia is *not* a symmetric map, and therefore it cannot be diagonalized on an eigenbasis. However, a modified form of diagonalization still applies.

Using the eigenstructure of the classical inertia, the principal eigenblades $\mathbf{E}_k$ of the classical inertia operator $\mathsf{I}_C[\,]$ become, in the total body inertia transformed to a weighted version of their dual bivectors:

$$\mathsf{I}_b[\mathbf{E}_k] = -\mathsf{i}_k\,\mathbf{E}_k\mathcal{I} = \mathsf{i}_k\ \star\mathbf{E}_k. \tag{A.7}$$

We used the Hodge dual $\star(\,)$ to rewrite the result in a general form. This *Hodge dual* of an element is defined through $X(\star X) = \mathbf{X}\widetilde{\mathbf{X}}\mathcal{I}$, where $\mathbf{X}$ is the Euclidean factor in X. For a normalized bivector E, this implies $E \star E = \mathcal{I}$, so if E is a Euclidean bivector $\mathbf{E}_i$, the Hodge dual is equal to $\star\mathbf{E}_i = -\mathbf{E}_i\mathcal{I}$, and for a bivector of the form $\boldsymbol{\epsilon}\mathbf{e}_i$, we get $\star(\boldsymbol{\epsilon}\mathbf{e}_i) = \mathbf{e}_i\mathbf{I}_d$.

Due to the factor $\mathcal{I}$ which the $\mathbf{E}_k$ accrue, they are no longer eigenelements of the total body inertia mapping. You really would not expect them to be in d-D, since the grades of input and output of the inertia map differ (3D is confusing: bivector in, bivector out). But the $\mathbf{E}_k$ are clearly still useful in specifying it compactly.

The remaining (non-Euclidean) bivectors in PGA are of the form $\boldsymbol{\epsilon}\mathbf{e}_i$ (ideal meet line rates, generating translational motion). These are mapped by the total PGA inertia eq.(A.6) to $(d-1)$-blades, and this can again be written as mapping to a multiple of their Hodge dual:

$$\mathsf{I}_b[\boldsymbol{\epsilon}\mathbf{e}_k] = m\,\mathbf{e}_k \cdot O = m\ \star(\boldsymbol{\epsilon}\mathbf{e}_k), \tag{A.8}$$

Thus we see from eq.(A.7) and eq.(A.8) that both for angular rates and linear rates, the inertial map effectively can be viewed as a weighted Hodge dual in $\mathbb{R}^{d,0,1}$. It converts 2-vectors to $(d-1)$-vectors, weighing them with aspects of the mass distribution. We therefore may call $\mathsf{I}_b[]$ the *inertia duality map*.

Let us show the 3D inertia map in a bit more detail. In 3D PGA, $\mathsf{I}_b[\mathbf{E}_1\mathcal{I}_3] = \mathsf{I}_b[\boldsymbol{\epsilon}\mathbf{e}_1] = m\,\mathbf{e}_1 \cdot \mathbf{I}_3 = m\,\mathbf{e}_{23} = m\mathbf{E}_1$, and cyclic, and $\mathsf{I}_b[\mathbf{E}_1] = \mathsf{i}_1\,\mathsf{I}_b[\mathbf{e}_2\mathbf{e}_3] = \mathsf{i}_1\,\boldsymbol{\epsilon}\mathbf{e}_1 = \mathsf{i}_1\mathbf{E}_1\mathcal{I}_3$ and cyclic. Taking a body frame basis $\{\mathbf{E}_1, \mathbf{E}_2, \mathbf{E}_3, \mathbf{E}_1\mathcal{I}_3, \mathbf{E}_2\mathcal{I}_3, \mathbf{E}_3\mathcal{I}_3\} = \{\mathbf{e}_{23}, \mathbf{e}_{31}, \mathbf{e}_{12}, \mathbf{e}_{01}, \mathbf{e}_{02}, \mathbf{e}_{03}\}$, the linear 3D inertia map has a matrix that looks like two off-diagonal diagonal 3×3 matrices:

$$\begin{bmatrix} 0 & 0 & 0 & m & 0 & 0 \\ 0 & 0 & 0 & 0 & m & 0 \\ 0 & 0 & 0 & 0 & 0 & m \\ \mathsf{i}_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathsf{i}_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathsf{i}_3 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \mathsf{O} & m\mathsf{I} \\ \mathsf{I_C} & \mathsf{O} \end{bmatrix}$$

Its off-diagonal diagonal structure clearly reveals that it is a duality mapping in 3D: *meet lines become join lines* (and vice versa). These considerations motivate the computation of the PGA inertia in the body frame by the algorithm spelled out in Section 2.2.5. You may want to see how it looks in 2D, using exercise D.9.

A.7 Parallel Axis Theorem: No Longer Needed

In the use of the classical inertia in a vector-based framework, you need the *Parallel Axis Theorem* (also known as Steiner's Theorem):

> Moving the origin axis (relative to which we computed the inertia $\mathsf{I}_C[\mathbf{B}_b]$) to pass instead through a point with location vector $\mathbf{q}$ in the local frame increases the inertia $\mathsf{I}_C[\mathbf{B}_b]$ by an additional term $m\,\mathbf{q}\wedge(\mathbf{q}\cdot\mathbf{B}_b)$.

The classical GA proof is simple, so we repeat it. In the definition of $\mathsf{I}_C[\mathbf{B}] \equiv \Sigma_i\, m_i\,\mathbf{r}_i\wedge(\mathbf{r}_i\cdot\mathbf{B})$, we summed over points with locations $\mathbf{r}_i$ relative to the centroid, and $\Sigma_i\mathbf{r}_i = \mathbf{0}$. Now consider $\mathbf{r}'_i \equiv \mathbf{r}_i - \mathbf{q}$, local vectors measured relative to a point Q. Then the classical inertia $\mathsf{I}_C[\mathbf{B}_b]'$ of the object turning around that point Q with the same angular velocity $\mathbf{B}_b$ is

$$\begin{aligned}
\mathsf{I}_C[\mathbf{B}_b]' &\equiv \Sigma_i\, m_i\,\mathbf{r}'_i\wedge(\mathbf{r}'_i\cdot\mathbf{B}_b)\\
&= \Sigma_i\, m_i\,(\mathbf{r}_i-\mathbf{q})\wedge((\mathbf{r}_i-\mathbf{q})\cdot\mathbf{B}_b)\\
&= \Sigma_i\, m_i\,\big(\mathbf{r}_i\wedge(\mathbf{r}_i\cdot\mathbf{B}_b) - \mathbf{q}\wedge(\mathbf{r}_i\cdot\mathbf{B}_b) - \mathbf{r}_i\wedge(\mathbf{q}\cdot\mathbf{B}_b) + \mathbf{q}\wedge(\mathbf{q}\cdot\mathbf{B}_b)\big)\\
&= \mathsf{I}_C[\mathbf{B}_b] + \mathbf{q}\wedge(\mathbf{q}\cdot\mathbf{B}_b). \qquad \text{(A.9)}
\end{aligned}$$

If we would like to compose inertias of a composite object, we can use the classical inertias of the individual parts, but we will have to make sure we add terms that correspond to how they are displaced relative to the new centroid. This cumbersome necessity is common practice in classical mechanics classes: when you want another viewpoint, you change the mapping. Seems natural.

In PGA, there is no longer any need for a parallel axis theorem for inertias: if you evaluate the current body inertia $\mathsf{I}_b[\,]$ on the new displaced axis, which is $\mathbf{B}_b - \boldsymbol{\epsilon}\,(\mathbf{q}\cdot\mathbf{B}_b)$ rather than $\mathbf{B}_b$ (in the Euclidean split encoding that classical usage), it provides the correct result. *In PGA inertia, we simply change the argument, not the mapping.* Let us show that in detail to convince you how eq.(A.9) is embedded inherently in PGA; for clarity in the comparison, let us take a pure rotation rate $\mathbf{B}_b$ (no translational velocity).

$$\begin{aligned}
\mathsf{I}_b[T_{\mathbf{q}}\mathbf{B}_b\widetilde{T}_{\mathbf{q}}] &= \mathsf{I}_b[\mathbf{B}_b - \boldsymbol{\epsilon}(\mathbf{q}\cdot\mathbf{B}_b)]\\
&= -m\,(\mathbf{q}\cdot\mathbf{B}_b)\cdot O - \mathsf{I}_C[\mathbf{B}_b]\,\mathcal{I}\\
&= -m\,(\mathbf{q}\cdot\mathbf{B}_b)\cdot(O+\mathbf{q}\mathcal{I}) - \mathsf{I}_C[\mathbf{B}_b]\,\mathcal{I}\\
&= m\,(\mathbf{q}\cdot\mathbf{B}_b)\cdot O + m\,(\mathbf{q}\cdot\mathbf{B}_b)\cdot(\mathbf{q}\mathcal{I}) - \mathsf{I}_C[\mathbf{B}_b]\,\mathcal{I}\\
&= m\,(\mathbf{q}\cdot\mathbf{B}_b)\cdot O - \big(\mathsf{I}_C[\mathbf{B}_b] + m\,\mathbf{q}\wedge(\mathbf{q}\cdot\mathbf{B}_b)\big)\,\mathcal{I}.
\end{aligned}$$

We have performed the point split to the new point Q. It shows that for the new values of the classical momentum and inertia, there is a contribution $m\,(\mathbf{q}\cdot\mathbf{B}_b)$ to the translational momentum, since the centroid now moves at the velocity $-\mathbf{q}\cdot\mathbf{B}_b$) and an additional contribution $m\,\mathbf{q}\wedge(\mathbf{q}\cdot\mathbf{B}_b)$ to the angular momentum. Thus the inertia term of the parallel axis has appeared automatically, merely by evaluating the PGA inertia $\mathsf{I}_b[]$ on the new axis $T_{\mathbf{q}}\mathbf{B}_b\widetilde{T}_{\mathbf{q}}$, and insisting on a classical split representation of its outcome.

The PGA inertia maps are valid anywhere; their specific outcome for a different situation is determined by the inherently spatial nature of the arguments they are given. Thus the PGA

body inertia map $\mathsf{I}_b[\,]$ itself does *not* need to change to accommodate a change of viewpoint; it has all viewpoint aspects embodied in its ability to deal with fully geometrical arguments (like axes). When you want the inertia for a different velocity state, you just give that new state as argument to the mapping. Enjoy this natural simplicity!

A.8 Linear and Angular Momentum

Classically, linear and angular momenta are treated separately. In PGA they are integrated in the momentum bivector P of eq.(2.11). That turns out to be convenient, and it is what is used to evaluate the dynamics. However, you may be curious how this new concept corresponds to the classical description. In this section, we show that the point split of the total momentum P relative to the centroid C is

$$P = \mathbf{p} \cdot C - \mathbf{L}\mathcal{I}. \tag{A.10}$$

Here $\mathbf{p}$ is the classical linear momentum vector and $\mathbf{L}$ is the GA/classical angular momentum bivector, both in the world frame. In 3D, you may also write the last term in terms of the angular momentum *vector* $\mathbf{l} \equiv \mathbf{L}/\mathbf{I}_3$ as $-L\mathcal{I}_3 = \boldsymbol{\epsilon}\mathbf{l}$. Note that a single mass point only has a linear momentum $\mathbf{p} \cdot C$, see Exercise D.10.

We recommend against this needless splitting of the total inertia in PGA, but it is educational to do it just once. The outcome clearly shows the additional precision of PGA (since $\mathbf{p} \cdot C$ is the momentum line passing through the point C; classically, such positional aspects as C needed to be stated explicitly and separately in natural language in the text, rather than within the algebra). Yet it also shows how the classical parts are individually retrievable from the total momentum. (One can retrieve $\mathbf{p}$ from the Euclidean part $\mathbf{p} \cdot O$ of P, then remove $\mathbf{p} \cdot (\mathbf{c}\mathcal{I})$ from the ideal part to obtain $\mathbf{L}\mathcal{I}$ and hence the Euclidean quantity $\mathbf{L}$.) We have thus not lost anything by moving to PGA, but actually gained unification and precision.[1]

Let us derive eq.(A.10) in a number of steps.

- **Euclidean split conversion:** First, we use eq.(A.4) to express the world inertia in terms of the classical inertia $\mathsf{I}_C[\,]$ as

$$P = \mathsf{I}_w[B_w] = M\left(m\,\mathbf{v}_b \cdot O - \mathsf{I}_C[\mathbf{B}_b]\,\mathcal{I}\right)\widetilde{M}, \tag{A.11}$$

  employing the body quantities $\mathbf{B}_b$ and $\mathbf{v}_b$ in the Euclidean split $B_b = \mathbf{B}_b + \boldsymbol{\epsilon}\mathbf{v}_b$. Their relationship to the Euclidean split for the world $B_w = \mathbf{B}_w + \boldsymbol{\epsilon}\mathbf{v}_w$ is rather subtle, it is not simply $\mathbf{B}_w = M\mathbf{B}_b\widetilde{M}$ and $\mathbf{v}_w = M\mathbf{v}_b\widetilde{M}$, since M introduces $\boldsymbol{\epsilon}$-containing elements. Rather, the conversions are related to the (rather unnatural) split of the motor $M = T_{\mathbf{c}}R$, as an origin rotation R and the translation $T_{\mathbf{c}}$ to the centroid. (This is clearly non-equivariant since it involves the origin; but it is commonly used in classical

[1] We show in Section 2.8 that Screw Theory and the Spatial Vector Algebra framework unify the two momentum components as well; they do so by extending the representational *vector* spaces to distinguish them, rather than by simply employing the *bivector* space of their algebras.

texts, including the GA-bivector treatment before PGA.)

$$\begin{aligned} B_w &= M B_b \widetilde{M} \\ &= T_{\mathbf{c}}\,(R\mathbf{B}_b\widetilde{R})\,\widetilde{T}_{\mathbf{c}} + M\boldsymbol{\epsilon}\mathbf{v}_b\widetilde{M} \\ &= R\mathbf{B}_b\widetilde{R} - \boldsymbol{\epsilon}\,\mathbf{c}\cdot(R\mathbf{B}_b\widetilde{R}) + M\boldsymbol{\epsilon}\mathbf{v}_b\widetilde{M} \\ &= R\mathbf{B}_b\widetilde{R} + \boldsymbol{\epsilon}\left(R\mathbf{v}_b\widetilde{R} - \mathbf{c}\cdot(R\mathbf{B}_b\widetilde{R})\right) \\ &= \mathbf{B}_w + \boldsymbol{\epsilon}\mathbf{v}_w. \end{aligned} \tag{A.12}$$

So the Euclidean split in the world frame relates to the Euclidean split in the body frame as

$$\mathbf{B}_w = R\mathbf{B}_b\widetilde{R} \tag{A.13}$$

$$\mathbf{v}_w = R\mathbf{v}_b\widetilde{R} - \mathbf{c}\cdot(R\mathbf{B}_b\widetilde{R}). \tag{A.14}$$

The velocity of the centroid is then:

$$\dot{\mathbf{c}} = R\mathbf{v}_b\widetilde{R} = \mathbf{v}_w + \mathbf{c}\cdot\mathbf{B}_w,$$

as expected, see eq.(A.2).

- **Linear momentum p:** The classical linear momentum is the mass times the velocity of the centroid $C = MO\widetilde{M}$. Hence the term $m\,\mathbf{v}_b\cdot O$ in eq.(A.11) transforms to the world frame quantity

$$M\,(m\,\mathbf{v}_b\cdot O)\,\widetilde{M} = m\mathbf{v}_w\cdot C = \mathbf{p}\cdot C.$$

This is the PGA representation of the line that has the direction of the classical world momentum vector $\mathbf{p}$, and passes through C.

- **Angular momentum:** The classical inertia $\mathsf{I}_C[\,]$ is computed in eq.(A.11) in a term that is multiplied by the null pseudoscalar $\mathcal{I}$. That destroys all translational parts, and is thus purely rotational.

$$\begin{aligned} M\,(\mathsf{I}_C[\mathbf{B}_b]\,\mathcal{I})\,\widetilde{M} &= T_{\mathbf{c}}R\,\mathsf{I}_C[\mathbf{B}_b]\,\widetilde{R}\widetilde{T}_{\mathbf{c}}\,\mathcal{I} \\ &= R\,\mathsf{I}_C[\mathbf{B}_b]\,\widetilde{R}\,\mathcal{I}. \end{aligned} \tag{A.15}$$

The outcome indicates how a rotational rate $\mathbf{B}_b$ in the body frame becomes the angular component of the total momentum. The resulting quantity is therefore recognizable as ($\mathcal{I}$ times) the GA/classical *angular momentum* 2-blade $\mathbf{L}$ in the world frame.

We thus find the relationship $P = \mathbf{p}\cdot C - \mathbf{L}\,\mathcal{I}$ between the total PGA momentum P and the classical constituents $\mathbf{p}$ and $\mathbf{L}$ – augmented by their customarily hidden need to specify the point C.

A.9 Total Forque, but Split

We can split the total forque of eq.(2.27) in its force and torque parts, if you need that to confirm that its definition make sense.

$$F = \Sigma_i F_i = \Sigma_i \, \mathbf{f}_i \cdot Q_i = \Sigma_i \, (\mathbf{f}_i \cdot O - \mathbf{T}_i \mathcal{I}) = \mathbf{f} \cdot O - \mathbf{T}_O \mathcal{I}, \tag{A.16}$$

where $\mathbf{f} = \Sigma_i \mathbf{f}_i$ is the *total force* and $\mathbf{T}_O \equiv \Sigma_i \mathbf{T}_i = \Sigma_i \mathbf{q}_i \wedge \mathbf{f}_i$ the *total torque* (relative to O). Thus this point split of F relative to the origin O shows the classical GA form of the total torque, in terms of the position vectors $\mathbf{q}_i$ of the masses relative to that origin. We emphasize that it is completely unnecessary to split the forque F before using it.

A.10 Newton and Euler Indeed Included

The PGA equation of motion in Table 2.1 are so compact that the reader may need convincing that they do indeed contain the classical equations (unless you are already familiar with similar 6D frameworks, such as Spatial Vector Algebra [16], then you would be disappointed if PGA did not!). So let us decompose them, merely for this purpose, into equations governing the linear motion (Euler's first law) and angular motion (Euler's second law). We do emphasize that this is unnecessary, eq.(2.28)-eq.(2.30) are complete, and solvable as they are – as our implemenations of Section 2.5 show!

A.10.1 Body Frame Equations

In essence the equation for $\dot{B}_b$ in the body frame is the straightforward

$$F_b = \dot{P}_b = \dot{\mathsf{I}}_b[B_b] = \mathsf{I}_b[\dot{B}_b]. \tag{A.17}$$

To establish the correspondence with the classical equations, we split forque and inertia into their Euclidean and ideal parts relative to the origin O.

$$\begin{aligned} \mathbf{f}_b \cdot O - \mathbf{T}_b \mathcal{I} &= \dot{\mathsf{I}}_b[\mathbf{B}_b + \boldsymbol{\epsilon}\mathbf{v}_b] \\ &= \mathsf{I}_b[\dot{\mathbf{B}}_b + \boldsymbol{\epsilon}\dot{\mathbf{v}}_b] \\ &= m\,\dot{\mathbf{v}}_b \cdot O - \mathsf{I}_C[\dot{\mathbf{B}}_b]\,\mathcal{I}. \end{aligned}$$

By Theorem 1 in Appendix C.4, we are allowed to just equate the obviously corresponding parts, leading to the familiar classical Newton/Euler equations in the body frame

$$\text{Euler's first law:} \quad m\,\dot{\mathbf{v}}_b = \mathbf{f}_b \tag{A.18}$$

$$\text{Euler's second law:} \quad \mathsf{I}_C[\dot{\mathbf{B}}_b] = \mathbf{T}_b, \tag{A.19}$$

In this form, the Euler equations of motion separate the translational and rotational aspects, which remain united in the PGA approach.

A.10.2 World Frame Equations

It is perhaps more common to represent these equations in the world frame (especially the first one). But we have to be careful, since merely applying the motor M does not perform the necessary conversion: splitting into Euclidean and non-Euclidean is not an equivariant operation (i.e., the transform of the Euclidean part is not the Euclidean part of the transform).

Let us first transform the force F_b and (to correspond to the classical approach) study the effect of the rotational part R and translational part $T_{\mathbf{c}}$ of $M = T_{\mathbf{c}}R$ separately. We define $\mathbf{f}_w \equiv R\,\mathbf{f}_b\,\widetilde{R}$ and $\mathbf{T}_{wC} = R\,\mathbf{T}_{bO}\,\widetilde{R}$, to improve readability. Then

$$\begin{aligned}
F_w &= MF_b\widetilde{M} && \text{(unsplit)} \\
&= T_{\mathbf{c}}R\,(\mathbf{f}_b \cdot O - \mathbf{T}_{bO}\,\mathcal{I})\,\widetilde{R}\widetilde{T}_{\mathbf{c}} \\
&= (T_{\mathbf{c}}R\,\mathbf{f}_b\widetilde{R}\widetilde{T}_{\mathbf{c}}) \cdot C - (R\mathbf{T}_{bO}\widetilde{R})\,\mathcal{I} \\
&= (T_{\mathbf{c}}\mathbf{f}_w\widetilde{T}_{\mathbf{c}}) \cdot C - \mathbf{T}_{wC}\,\mathcal{I} \\
&= \mathbf{f}_w \cdot C - \mathbf{T}_{wC}\,\mathcal{I} && \text{(point split for } C\text{)} && \text{(A.20)} \\
&= \mathbf{f}_w \cdot O - (\mathbf{c} \wedge \mathbf{f}_w + \mathbf{T}_{wC})\,\mathcal{I}. && \text{(Euclidean split)} && \text{(A.21)}
\end{aligned}$$

We presented two forms of the final result: eq.(A.20) relative to the newly displaced centroid C where the force attaches, and eq.(A.21) relative to the arbitrary origin O in the world frame. It is clear that the former is simpler, and we used it to define the *world frame torque* $\mathbf{T}_{wC}$ *relative to* C as $\mathbf{T}_{wC} \equiv R\mathbf{T}_{bO}\widetilde{R}$. But note that we departed from the strict separation between Euclidean and non-Euclidean to do so (in writing the point split for C rather than O). In the truly classical treatments, one tends to work in what we called the Euclidean split, relative to the arbitrary origin (as in eq.(A.21))). One then experiences an augmented torque $\mathbf{T}_{wC} + \mathbf{c} \wedge \mathbf{f}_w$ merely due to the offset of O to C.

So much for the forque part. We also need to convert the right hand side motion term $\dot{\mathsf{l}}_b[B_b] = \mathsf{l}_b[\dot{B}_b]$ of eq.(A.17) to the world frame. This is done by applying M to $\mathsf{l}_b[\,]$, but we also should convert its argument $\dot{B}_b$ and relate it to $\dot{B}_w$. In general, derivatives transform not straightforwardly between body frame and world frame (as eq.(2.9) shows), but the derivative of the rate bivector is fortunately the (unique) exception, since $B_w \times B_w = 0$:

$$\dot{B}_w = \tfrac{d}{dt}(MB_b\widetilde{M}) = B_w \times B_w + M\dot{B}_b\widetilde{M} = M\dot{B}_b\widetilde{M}.$$

So $\dot{B}_b$ simply transforms by applying the versor M, as you might expect from naive pattern matching with $B_w = MB_b\widetilde{M}$ (OK, you were right this time, but be careful in general!).

In order to apply eq.(A.4), we need to split the argument $B_b = \widetilde{M}B_wM$ into its Euclidean and ideal parts. This is not simply the M-transform of the Euclidean and ideal parts of $\dot{B}_b$; rather they mix according to

$$\begin{aligned}
\widetilde{M}\dot{B}_wM &= \widetilde{M}(\dot{\mathbf{B}}_w + \boldsymbol{\epsilon}\dot{\mathbf{v}}_w)M \\
&= \widetilde{R}\widetilde{T}_{\mathbf{c}}(\dot{\mathbf{B}}_w + \boldsymbol{\epsilon}\dot{\mathbf{v}}_w)T_{\mathbf{c}}R \\
&= \widetilde{R}\big(\dot{\mathbf{B}}_w + \boldsymbol{\epsilon}\,(\dot{\mathbf{v}}_w + \mathbf{c} \cdot \dot{\mathbf{B}}_w)\big)R \\
&= \widetilde{R}\dot{\mathbf{B}}_wR + \boldsymbol{\epsilon}\,\widetilde{R}(\dot{\mathbf{v}}_w + \mathbf{c} \cdot \dot{\mathbf{B}}_w).
\end{aligned}$$

So the world frame rotational accelaration is experienced in the body frame as $\widetilde{R}\dot{\mathbf{B}}_w R$, and the world frame accelaration $\boldsymbol{\epsilon}\mathbf{v}_w$ as $\boldsymbol{\epsilon}\,\widetilde{R}(\dot{\mathbf{v}}_w + \mathbf{c}\cdot\dot{\mathbf{B}}_w)R$. Using that, we compute in terms of the world frame quantities

$$\begin{aligned}
\mathsf{I}_w[\dot{B}_w] &= M\,\mathsf{I}_b[\widetilde{M}\dot{B}_w M]\,\widetilde{M} \\
&= M\,\mathsf{I}_b[\widetilde{R}\dot{\mathbf{B}}_w R + \boldsymbol{\epsilon}\,\widetilde{R}(\dot{\mathbf{v}}_w + \mathbf{c}\cdot\dot{\mathbf{B}}_w)R]\,\widetilde{M} \\
&= m\,M\big((\widetilde{R}(\dot{\mathbf{v}}_w + \mathbf{c}\cdot\dot{\mathbf{B}}_w)R)\cdot O\big)\widetilde{M} - R\,\mathsf{I}_C[\widetilde{R}\dot{\mathbf{B}}_w R]\,\widetilde{R}\mathcal{I} \\
&= m\,(\dot{\mathbf{v}}_w + \mathbf{c}\cdot\dot{\mathbf{B}}_w)\cdot C - R\,\mathsf{I}_C[\widetilde{R}\dot{\mathbf{B}}_w R]\,\widetilde{R}\mathcal{I}. \quad \text{(point split for } C\text{)}
\end{aligned} \tag{A.22}$$

This is decomposed to show what happens translationally at the centroid C, and rotationally in the ideal plane. According to Appendix C.4, it can thus be equated term for term to eq.(A.20) and we obtain

$$\text{Euler's first law:} \qquad m\,(\dot{\mathbf{v}}_w + \mathbf{c}\cdot\dot{\mathbf{B}}_w) = \mathbf{f}_w \tag{A.23}$$

$$\text{Euler's second law:} \qquad R\,\mathsf{I}_C[\widetilde{R}\dot{\mathbf{B}}_w R]\,\widetilde{R} = \mathbf{T}_{wC}, \tag{A.24}$$

The expression for the linear acceleration looks involved; it contains more than just $\dot{\mathbf{v}}_w$; but this is the correct formula that includes the spatial information. These 'spatial accelerations' are additive, without the need to include angular velocity terms like the Coriolis force.[2] For a comparison with the classically used acceleration, see [14] (that text is actually an analysis within 6D Spatial Vector Algebra but it equally applies to other 6D frameworks like PGA).

As you see, this world frame view of eq.(A.23) and eq.(A.24) is slightly more involved than the unified body frame equations of eq.(2.28) and eq.(2.29). In PGA, the body frame view is preferred; the resulting solution for B_b is just as easily integrated to the motor M by solving $\dot{M} = -\frac{1}{2}MB_b$ as it would be to solve M from B_w by $\dot{M} = -\frac{1}{2}B_w M$.

A.11 Summarizing the Atavism

We have repeatedly shown how the integrated PGA concepts embody the classical results when considered in the various splits. However, we stopped at the classical GA approach, in which angular momenta and 3D torques are at least already bivectors. For the 3D case, we now go a bit further, and connect to the antique vector approach still taught in almost all physics classes. We will adopt commonly used symbols for the entities, so that its correspondence with PGA becomes more obvious. In the end, we will see that very little actually needs to be added to the antique way to become PGA; in fact, the geometric elements that are mentioned in all texts ('axial vectors are different', 'mind relative to what point you calculate') simply are attached as actually algebraic computational factors, making correct computations automatic. It is a small step, but it elevates the classical approach to the equivariant PGA framework which automatically incorporates the explicitly required antique tricks (like the parallel axis theorem). This again confirms that PGA provides the proper data structures to represent dynamics quantities.

[2] This is very similar to how the combination of PGA inertias does not require Steiner's 'parallel axis theorem' which the composition of classical inertias needs.

PGA $\mathbb{R}^{3,0,1}$	decomposition into classical terms			split
kinematics rate		rotation	translation	
B	$=$	$\boldsymbol{\omega} \cdot R$	$+ \quad \boldsymbol{\epsilon} \wedge \mathbf{v}$	point R
bivector	$=$	$\boldsymbol{\omega} \cdot O$	$+ \quad \boldsymbol{\epsilon} \wedge (\mathbf{v} + \mathbf{r} \times \boldsymbol{\omega})$	origin
body properties		mass	inertia	
$\mathsf{I}[B]$	$=$	$m\,\mathbf{v} \cdot C$	$+ \quad \boldsymbol{\epsilon} \wedge I\boldsymbol{\omega}$	centroid
dual bivector	$=$	$m\,\mathbf{v} \cdot O$	$+ \quad \boldsymbol{\epsilon} \wedge \big(I\boldsymbol{\omega} + m\,\mathbf{c} \times (\mathbf{v} + \boldsymbol{\omega} \times \mathbf{c}) \big)$	origin
momentum		linear	angular	
P	$=$	$\mathbf{p} \cdot C$	$+ \quad \boldsymbol{\epsilon} \wedge \boldsymbol{\ell}$	centroid
dual bivector	$=$	$\mathbf{p} \cdot O$	$+ \quad \boldsymbol{\epsilon} \wedge \big(\boldsymbol{\ell} + \mathbf{c} \times \mathbf{p} \big)$	origin
dynamics forque		force	torque	
F	$=$	$\mathbf{f} \cdot R$	$+ \quad \boldsymbol{\epsilon} \wedge \boldsymbol{\tau}$	point R
dual bivector	$=$	$\mathbf{f} \cdot O$	$+ \quad \boldsymbol{\epsilon} \wedge \big(\boldsymbol{\tau} + \mathbf{r} \times \mathbf{f} \big)$	origin

Table A.1: *For 3D PGA, we display the transcription of the basic entities B, $\mathsf{I}_C[\,]$, P and F (in red) in terms of elements of classical mechanics (in blue). The classically disparate pieces are recognizable; they can only be added to form a coordinate-independent entity by explicitly appending the point at which they occur (which in classical physics is in the text, not in the formula), and meeting with the null vector $\boldsymbol{\epsilon}$ (the hyperplane at infinity). Those two algebraic measures (in gray) effectively unify axial vectors and regular vectors, into distinguishable additive bivectors, combining into one invariant quantity.*

- *Rates*
  In the Euclidean split, our rate $B = \mathbf{B} + \boldsymbol{\epsilon}\mathbf{v}$ consists of two parts: a purely Euclidean 2-blade $\mathbf{B}$ for the rotational aspect, and a translational part parametrized by a velocity vector $\mathbf{v}$. Classically, one would characterize the rotational velocity by a vector $\boldsymbol{\omega}$ along the rotation axis. For a counterclockwise $\mathbf{B} = \mathbf{e}_1 \wedge \mathbf{e}_2$, one would have $\boldsymbol{\omega} = \mathbf{e}_3$, so we can relate the two through
  $$\mathbf{B} = \boldsymbol{\omega}\,\mathbf{I}_3.$$
  This prepares us for expressing various splits using $\boldsymbol{\omega}$:
  $$\begin{aligned} B &= \mathbf{B} + \boldsymbol{\epsilon}\,\mathbf{v} \\ &= \boldsymbol{\omega}\cdot\mathbf{I}_3 + \boldsymbol{\epsilon}\,\mathbf{v} && \text{(origin split)} \\ &= \boldsymbol{\omega}\cdot(\mathbf{I}_3 + \mathbf{q}\,\mathcal{I}_3) + \boldsymbol{\epsilon}\,\mathbf{v} - \boldsymbol{\omega}\cdot(\mathbf{q}\,\mathcal{I}_3) \\ &= \boldsymbol{\omega}\cdot Q + \boldsymbol{\epsilon}\,\big(\mathbf{v} - (\boldsymbol{\omega}\wedge\mathbf{q})\,\mathbf{I}_3\big) \\ &= \boldsymbol{\omega}\cdot Q + \boldsymbol{\epsilon}\,\big(\mathbf{v} + \boldsymbol{\omega}\times\mathbf{q}\big), && \text{(point split } Q = O + \mathbf{q}\mathcal{I}) \end{aligned}$$
  where we used the $\times$ notation for the 3D cross product $\mathbf{a}\times\mathbf{b} \equiv (\mathbf{b}\wedge\mathbf{a})\,\mathbf{I}_3$.
  This was actually a special case, since we made the axis of $\boldsymbol{\omega}$ pass through the origin O. In Table A.1, we show the more natural formulation where the $\boldsymbol{\omega}$-axis passes through an arbitrary point R, and we then develop its split as seen from the arbitrary origin O. This swaps some signs.

- *Inertia*
  For the inertia, we introduced the classical inertia based on a Euclidean bivector $\mathbf{B}$, producing a bivector $\mathsf{I}_C[\mathbf{B}]$. In the antique formulation, one would feed the inertia map the axial vector $\boldsymbol{\omega} \equiv \mathbf{B}/\mathbf{I}_3$ and expect an axial vector as its outcome. Let us denote that linear map as I_c (italic I, small c); you can read it as a matrix if you prefer. Then that map I_c is defined through $\mathsf{I}_C[\mathbf{B}] = \mathsf{I}_C[\boldsymbol{\omega}\,\mathbf{I}_3] \equiv (I_c\boldsymbol{\omega})\,\mathbf{I}_3$.
  In the equations, we always have the product with $\mathcal{I}_3 = \boldsymbol{\epsilon}\,\mathbf{I}_3$, so we can replace $\mathsf{I}_C[\mathbf{B}]\,\mathcal{I}$ by:
  $$\mathsf{I}_C[\mathbf{B}]\,\mathcal{I} = \mathsf{I}_C[\mathbf{B}]\,\boldsymbol{\epsilon}\,\mathbf{I}_3 = \boldsymbol{\epsilon}\,I_c\boldsymbol{\omega}\,\mathbf{I}_3^2 = -\boldsymbol{\epsilon}\,I_c\boldsymbol{\omega}.$$
  An expression like $\mathbf{c}\cdot\mathbf{B}$ is converted as:
  $$\mathbf{c}\cdot\mathbf{B} = \mathbf{c}\cdot(\boldsymbol{\omega}\mathbf{I}_3) = (\mathbf{c}\wedge\boldsymbol{\omega})\,\mathbf{I}_3 = \boldsymbol{\omega}\times\mathbf{c}.$$
  And also
  $$(\mathbf{a}\wedge\mathbf{b})\,\mathcal{I}_3 = (\mathbf{a}\wedge\mathbf{b})\,\boldsymbol{\epsilon}\,\mathbf{I}_3 = \boldsymbol{\epsilon}\,(\mathbf{a}\wedge\mathbf{b})\,\mathbf{I}_3 = -\boldsymbol{\epsilon}\,(\mathbf{a}\times\mathbf{b}).$$
  With those results, the displaced inertia in the origin split is converted to:
  $$-\big(\mathsf{I}_C[\mathbf{B}] + m\,\mathbf{c}\wedge(\mathbf{v} + \mathbf{c}\cdot\mathbf{B})\big)\,\mathcal{I}_3 = \boldsymbol{\epsilon}\,\Big(I_c\boldsymbol{\omega} + m\,\mathbf{c}\times(\mathbf{v} + \boldsymbol{\omega}\times\mathbf{c})\Big).$$
  That is the familiar antique expression, which includes the traditional 'inertial forces' due to the frame change.

- *Forques*
  The antique treatment of a force as a vector $\mathbf{f}$ integrates well with the local description $\mathbf{f} \cdot Q$ of a force at a location Q – though in the usual treatment, the Q is in the text, and not in the formula.

  Torque is classically viewed as one of those axial vectors $\boldsymbol{\tau}$, whereas in GA we see it as a bivector $\mathbf{T}$ (which is an axis in 3D PGA, the meet of two planes). Conversion is simply by the right-hand rule $\mathbf{T} = \boldsymbol{\tau}\,\mathbf{I}_3$, so that

  $$-\mathbf{T}\,\mathcal{I}_3 = -\boldsymbol{\epsilon}\,\mathbf{T}\mathbf{I}_3 = \boldsymbol{\epsilon}\,\boldsymbol{\tau}.$$

  Therefore again, the embedding into 3D PGA merely requires a locational post-operation $\cdot Q$ on the classical linear vector, and a prefactor $\boldsymbol{\epsilon}\wedge$ on the classical axial vector.

So basically, the procedure to atavistically revert to antique mechanics is: replace all PGA symbols for bivectors by their classical counterparts for vectors, and change postmultiply by $\mathcal{I}_3$ into premultiply by $\boldsymbol{\epsilon}\wedge$ (and then extract the Euclidean factor as the angular part). Oh, and drop the all-important location of where things are defined, and hide that part somewhere in the text accompanying your classical equation.

A.12 Kinetic Energy

Let us see how the PGA definition of kinetic energy $T\mathcal{I} = \frac{1}{2} B_b \wedge \mathsf{I}_b[B_b]$ of eq.(2.34) contains the classical results, by performing a Euclidean split on it in terms of $B_b = \mathbf{B}_b + \boldsymbol{\epsilon}\mathbf{v}_b$.

$$\begin{aligned} B_b \wedge \mathsf{I}_b[B_b] &= (\mathbf{B}_b + \boldsymbol{\epsilon}\mathbf{v}_b) \wedge (m\,\mathbf{v}_b \cdot O - \mathsf{I}_C[\mathbf{B}_b]\,\mathcal{I}) \\ &= m\,\mathbf{B}_b \wedge (\mathbf{v}_b \cdot O) + m\,(\boldsymbol{\epsilon}\mathbf{v}_b) \wedge (\mathbf{v}_b \cdot O) - \mathbf{B}_b\,(\mathsf{I}_C[\mathbf{B}_b]\mathcal{I}) \\ &= 0 + (m\,{\mathbf{v}_b}^2 - \mathbf{B}_b \cdot \mathsf{I}_C[\mathbf{B}_b])\,\mathcal{I} \\ &= \left(\tfrac{1}{2} m\,{\mathbf{v}_b}^2 + \tfrac{1}{2}\,\Sigma_{k=1}^{3}\,\mathsf{i}_k\,[\boldsymbol{\omega}]_k^2\right)(2\,\mathcal{I}), \end{aligned} \tag{A.25}$$

where we used that $\mathbf{B}_b = \boldsymbol{\omega}\mathcal{I}_3$. Thus the PGA expression indeed contains the classical expression for kinetic energy as the sum of the classical translational energy and the rotational energy, and hence defines the scalar kinetic energy T.

A.13 Power

Performing a Euclidean split of the arguments on the PGA power defined by $\Pi\,\mathcal{I} = B_w \wedge F_w$ of eq.(2.36)confirms the classical result in more detail:

$$\begin{aligned} B_w \wedge F_w &= (\mathbf{B}_w + \boldsymbol{\epsilon}\mathbf{v}_w) \wedge (\mathbf{f}_w \cdot O - \mathbf{T}_{wO}\,\mathcal{I}) \\ &= (\boldsymbol{\epsilon}\mathbf{v}_w) \wedge (\mathbf{f}_w \cdot O) - \mathbf{B}_w \wedge (\mathbf{T}_{wO}\,\mathcal{I}) \\ &= (\mathbf{v}_w \cdot \mathbf{f}_w - \mathbf{B}_w \cdot \mathbf{T}_{wO})\,\mathcal{I} && \text{(A.26)} \\ &= \big(\mathbf{v}_w \cdot \mathbf{f}_w - \mathbf{B}_w \cdot (\mathbf{x} \wedge \mathbf{f}_w)\big)\,\mathcal{I} && \text{(A.27)} \\ &= \big((\mathbf{v}_w + \mathbf{x} \cdot \mathbf{B}_w) \cdot \mathbf{f}_w\big)\,\mathcal{I} \\ &= \big(\dot{\mathbf{x}}_w \cdot \mathbf{f}_w\big)\,\mathcal{I} && \text{(A.28)} \end{aligned}$$

since $\mathbf{T}_{wO} = \mathbf{x} \wedge \mathbf{f}_w$ at the location X and eq.(A.2) gives $\dot{\mathbf{x}}_w \equiv \mathbf{v}_w + \mathbf{x} \cdot \mathbf{B}_w$. We thus retrieve as scalar factor the usual expression of displacement times force. The intermediate expression eq.(A.27) shows that the torque works on the angular velocity $\mathbf{B}_w$, and the force vector $\mathbf{f}_w$ on the translational velocity $\mathbf{v}$. But all this is implicitly unified in the forque expression.

A.14 Contact in Classical GA Form

Rewriting of eq.(2.32) can be based on:

$$Q \vee Q \times A = Q \vee Q \times (\mathbf{A} + \boldsymbol{\epsilon}\mathbf{a}) = Q \vee \big((\mathbf{a} + \mathbf{q} \cdot \mathbf{A})\,\mathcal{I}\big) = (\mathbf{a} + \mathbf{q} \cdot \mathbf{A}) \cdot Q$$

and the dot product of join lines at a point which simplifies to $(\mathbf{v} \cdot Q) \cdot (\mathbf{w} \cdot Q)^{\sim} = \mathbf{v} \cdot \mathbf{w}$. The former is based on a Euclidean split, so it requires the specification of an (arbitrary) origin to give A and Q coordinates. The inverse inertia maps may be split in each body frame to involve the classical inertias of each object by eq.(A.5), and then needs to address the common point Q from each centroid.

$$\mathsf{I}_b{}^{-1}[N] = \mathsf{I}_b{}^{-1}[\mathbf{n} \cdot Q] = \mathsf{I}_b{}^{-1}[\mathbf{n} \cdot C + (\mathbf{n} \wedge (\mathbf{q} - \mathbf{c}))\mathcal{I}] = \boldsymbol{\epsilon}\mathbf{n}/m + \mathsf{I}_C^{-1}[\mathbf{n} \wedge (\mathbf{q} - \mathbf{c})],$$

so each term in the denominator of the expression eq.(2.32) for j becomes of the form:

$$\begin{aligned}
\big(Q \vee Q \times \mathsf{I}^{-1}[N]\big) \cdot \widetilde{N} &= \big(\mathbf{n}/m + \mathsf{I}_C^{-1}[\mathbf{n} \wedge (\mathbf{q} - \mathbf{c})] \cdot Q\big) \cdot (\mathbf{n} \cdot Q)^{\sim} \\
&= 1/m + (\mathbf{q} - \mathbf{c}) \cdot \mathsf{I}_C^{-1}[\mathbf{n} \wedge (\mathbf{q} - \mathbf{c})] \cdot Q\big) \cdot (\mathbf{n} \cdot Q)^{\sim} \\
&= 1/m + (\mathbf{n} \wedge (\mathbf{q} - \mathbf{c})) \cdot \mathsf{I}_C^{-1}[\mathbf{n} \wedge (\mathbf{q} - \mathbf{c})],
\end{aligned}$$

for the appropriate m, $\mathsf{I}_C[]$ and $\mathbf{c}$. This basically retrieves the formula for j in [23], though in the more symmetrical d-dimensional GA bivector form (rather than the vector form which holds only in 3D).

Appendix B

Hand Computation of Simple Cases

B.1 Solving the Free Motion Equation

In simple situations one can solve the equations of motion by hand. Let us show some of the techniques involved, by considering increasingly complicated settings in the next few sections. If you are into Computer Graphics, you may skip these sections, since RK4 (or another preferred numerical integration method) is your tool. But if you are doing classical mechanics, as study or profession, the techniques will be of interest. (Many of the versor integration techniques actually transfer to quantum mechanics, done the GA way, so it is good practice to see them in this tangible classical setting.)

An apology: in the current writeup, we show the correspondence with the classical equations by solving the PGA equations in split form. This is *not* in the spirit of PGA, we plan to work out unsplit solution methods in the future.

To develop some experience with the integration of equations eq.(2.28)-eq.(2.30), let us focus on the simplest example: a freely moving object. Without external forces, we find that we effectively need to solve:

$$\dot{B}_b = \mathsf{I}_b{}^{-1}[B_b \times \mathsf{I}_b[B_b]].$$

The specific off-diagonal block diagonal form of the total inertia leads to an almost independent treatment of the translational and angular part of the motion, as we can see by writing out the equation by means of the Euclidean split into Euclidean and ideal terms for $B_b = \mathbf{B}_b + \boldsymbol{\epsilon}\mathbf{v}_b$.

$$\begin{aligned} \dot{B}_b &= \mathsf{I}_b{}^{-1}[B_b \times \mathsf{I}_b[B_b]] \\ &= \mathsf{I}_b{}^{-1}[m\,(-\mathbf{v}_b \cdot \mathbf{B}_b) \cdot O - \mathbf{B}_b \times \mathsf{I}_C[\mathbf{B}_b]\,\mathcal{I}] \\ &= \boldsymbol{\epsilon}\,(-\mathbf{v}_b \cdot \mathbf{B}_b) + \mathsf{I}_C^{-1}[\mathbf{B}_b \times \mathsf{I}_C[\mathbf{B}_b]], \end{aligned} \tag{B.1}$$

using some details spelled out in Exercises D.14 and D.15, and the inverse inertia from eq.(A.5). Since this should equal $\dot{B}_b = \dot{\mathbf{B}} + \boldsymbol{\epsilon}\dot{\mathbf{v}}_b$, we have in the free motion case:

$$\begin{cases} \dot{\mathbf{v}}_b = -\mathbf{v}_b \cdot \mathbf{B}_b \\ \dot{\mathbf{B}} = \mathsf{I}_C^{-1}[\mathbf{B}_b \times \mathsf{I}_C[\mathbf{B}_b]], \end{cases} \tag{B.2}$$

The angular part only depends on $\mathbf{B}_b$, the translational part also contains a rotational element (since when the object frame rotates, so do translational velocities measured relative to it).

These equations should be solved for $\dot{B}_b$ in each of the cases, and then the motor can be obtained by integrating $\dot{M} = -\frac{1}{2}\, M\, \dot{B}_b$.

B.2 Free Spherical Top

Let us first take the simplest of the simplest: a uniform spherical object with mass m. Its inertial eigenvalues are all identical[1] (call them i), so that $\mathsf{I}_C[\mathbf{B}_b] = \mathsf{i} \star \mathbf{B}_b$. Therefore the contribution of $\mathbf{B}_b \times \mathsf{I}_C[\mathbf{B}_b]$ is zero in this spherical case. In the body frame, the differential equation for B_b then reduces to

$$\dot{B}_b \equiv \dot{\mathbf{B}}_b + \boldsymbol{\epsilon}\dot{\mathbf{v}}_b = -\boldsymbol{\epsilon}\,(\mathbf{v}_b \cdot \mathbf{B}_b). \tag{B.3}$$

Let us unclutter our equations by momentarily dropping the body subscripts on $\mathbf{B}_b$ and $\mathbf{v}_b$; we are working the body frame until further notice.

For the Euclidean part $\mathbf{B}$, the equation $\dot{\mathbf{B}} = 0$ integrates to a constant $\mathbf{B} = \mathbf{B}_0$. Then we find from the ideal part that $\dot{\mathbf{v}} = -\mathbf{v} \cdot \mathbf{B}_0$ implying:

$$\mathbf{v} = \widetilde{M}_{\mathbf{B}_0}\, \mathbf{v}_0\, M_{\mathbf{B}_0}, \quad \text{with} \quad M_{\mathbf{B}_0} \equiv e^{-\mathbf{B}_0 t/2}, \tag{B.4}$$

with $\mathbf{v}_0$ an integration constant vector. Therefore the total body frame bivector B_b is the rather ungainly

$$B_b = \mathbf{B}_0 + \widetilde{M}_{\mathbf{B}_0}\,(\boldsymbol{\epsilon}\mathbf{v}_0)\, M_{\mathbf{B}_0}. \tag{B.5}$$

We need to find M by integrating $\dot{M} = -\frac{1}{2} M B_b$ (see eq.(2.28)), but with the time-dependence of B_b this requires some work. Note that $\mathbf{B}_0$ is invariant under $M_{\mathbf{B}_0}$, i.e., $M_{\mathbf{B}_0}\mathbf{B}_0\widetilde{M}_{\mathbf{B}_0} = \mathbf{B}_0$, so we can write B_b in the form $B_b = \widetilde{M}_{\mathbf{B}_0}\,(\mathbf{B}_0 + \boldsymbol{\epsilon}\mathbf{v}_0)\, M_{\mathbf{B}_0}$. This suggests considering the differential equation for the motor $M_{\mathbf{B}_0} M \widetilde{M}_{\mathbf{B}_0}$ instead.

$$\begin{aligned}
&\tfrac{d}{dt}(M_{\mathbf{B}_0} M \widetilde{M}_{\mathbf{B}_0}) = \\
&\quad = \quad (M_{\mathbf{B}_0} M \widetilde{M}_{\mathbf{B}_0}) \times \mathbf{B}_0 + M_{\mathbf{B}_0} \dot{M} \widetilde{M}_{\mathbf{B}_0} \\
&\quad = \quad -\tfrac{1}{2}\mathbf{B}_0\,(M_{\mathbf{B}_0} M \widetilde{M}_{\mathbf{B}_0}) + \tfrac{1}{2}\,(M_{\mathbf{B}_0} M \widetilde{M}_{\mathbf{B}_0})\mathbf{B}_0 \\
&\qquad -\tfrac{1}{2} M_{\mathbf{B}_0} M \big(\mathbf{B}_0 + \widetilde{M}_{\mathbf{B}_0}(\boldsymbol{\epsilon}\mathbf{v}_0) M_{\mathbf{B}_0}\big) \widetilde{M}_{\mathbf{B}_0} \\
&\quad = \quad -\tfrac{1}{2}\mathbf{B}_0\,(M_{\mathbf{B}_0} M \widetilde{M}_{\mathbf{B}_0}) - \tfrac{1}{2}(M_{\mathbf{B}_0} M \widetilde{M}_{\mathbf{B}_0})\,\boldsymbol{\epsilon}\mathbf{v}_0.
\end{aligned}$$

It follows from this by direct integration that $M_{\mathbf{B}_0} M \widetilde{M}_{\mathbf{B}_0} = M_{\mathbf{B}_0} M_0 M_{\mathbf{v}}$, with $M_{\mathbf{v}} = \exp(-\frac{1}{2}\boldsymbol{\epsilon}\mathbf{v}_0 t/2)$ and M_0 an integration constant.

Restoring the body subscripts, the differential equation for the body frame B_b we found in eq.(B.5) is thus solved by

$$M = M_0 M_{\mathbf{v}_b} M_{\mathbf{B}_{b0}} = M_{\mathbf{v}_w} M_0 M_{\mathbf{B}_{b0}} = e^{-(\boldsymbol{\epsilon}\mathbf{v}_{w0})t/2}\, M_0\, e^{-\mathbf{B}_{b0} t/2},$$

[1] It does not have to be a sphere, any object with this property of equal principal inertias will do, such as a cube of uniform density.

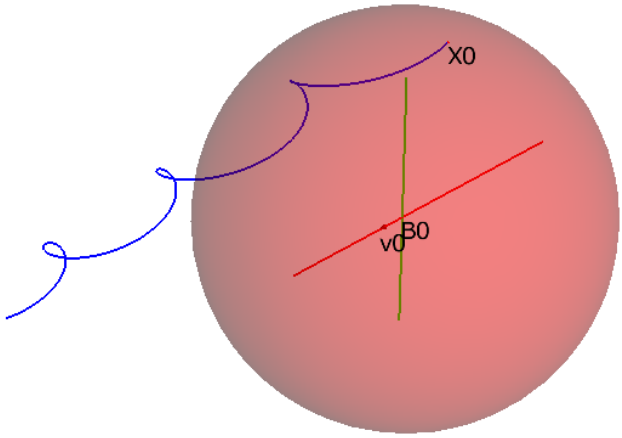


Figure B.1: *The motion of a point on a freely moving sphere: a uniform rotation around a uniformly translating centroid.*

where $\boldsymbol{\epsilon}\mathbf{v}_{w0} = M_0(\boldsymbol{\epsilon}\mathbf{v}_{b0})\widetilde{M}_0$ (note that $\mathbf{v}_{w0} = M_0\mathbf{v}_{b0}\widetilde{M}_0$ would not make $\mathbf{v}_{w0}$ Euclidean). Reading from the right, we see how the object with initial pose M_0 is rotated as an angular rate $\mathbf{B}_b$ in the body frame, then moved by M_0 to the world frame, and finally translated with world frame velocity $\mathbf{v}_{w0}$. We can learn from such processing that a rotating vector $\mathbf{v}_b$ in the body frame (as we obtained from the body frame integration of $\dot{B}_b$ in eq.(B.5)) is perhaps more easily seen as a constant vector $\mathbf{v}_{w0}$ in the world frame. Apparently, that is what eq.(B.3) implied.

During the derivation, we found that there are two constants of motion. In the body frame, there is $\mathbf{B}_{b0}$, which because of $\mathbf{L}_b = \mathsf{i}\mathbf{B}_{b0}$ can also be written in terms of the body angular momentum $\mathbf{L}_b$. In the world frame, there is the conserved quantity $\mathbf{v}_{w0}$, which because of $\mathbf{p}_w = m\mathbf{v}_{w0}$ can be expressed in the linear momentum $\mathbf{p} \equiv \mathbf{p}_w$ of the object's translation. We can also express the body angular momentum in the world frame through the line $L_w = M_0\mathbf{L}_b\widetilde{M}_0$, but the factor of $\mathcal{I}$ in the inertia term implies that only the ideal part contributes; the angular motion direction is a location-independent quantity. So it is better to define the purely Euclidean 2-blade $\mathbf{L} \equiv \mathbf{L}_w$ through $\mathcal{I}\mathbf{L}_w = M(\mathcal{I}\mathbf{L}_b)\widetilde{M}$.

To summarize, the spherical top moves with the motor

$$\begin{aligned} M &= e^{-(\boldsymbol{\epsilon}\mathbf{p}/m)\,t/2}\, M_0\, e^{-(\mathbf{L}_b/\mathsf{i})\,t/2} \\ &= e^{-(\boldsymbol{\epsilon}\mathbf{p}/m)\,t/2}\, e^{-(\mathbf{L}/\mathsf{i})\,t/2}\, M_0, \end{aligned} \tag{B.6}$$

in terms of its world frame constants of motion $\mathbf{p}$ and $\mathbf{L}$ and its material properties m and i. Here M_0 is the initial state of its world frame (in both position and orientation) relative to the body eigenframe. This is illustrated in Figure B.1. Note that the two exponentials in eq.(B.6) cannot be merged by addition of their bivectors, since those do not generally commute.

B.3 Free Rotationally Symmetric Top

Having used the spherical top to develop our basic techniques, we now handle the rotationally symmetric top. For such a top, two principal eigen-inertias are identical (see [1] Ch. 28, for pretty arguments about symmetries of inertias); let us pick i_3 as the odd one out, and set $\mathsf{i}_1 = \mathsf{i}_2 = \mathsf{i}$. We then have, as the body frame inertia,

$$\mathsf{I}_C[\mathbf{B}_b] = \mathsf{i}\,\mathbf{B}_b + (\mathsf{i}_3 - \mathsf{i})\,[\mathbf{B}_b]_3\,\mathbf{E}_3, \tag{B.7}$$

where $[\mathbf{B}_b]_3 \equiv -\mathbf{B}_b * \mathbf{E}_3$ denotes the third coordinate of $\mathbf{B}_b$ on the inertia eigenbasis. We will be working in the body frame for the remainder of this section, so let us drop the body subscripts for $\mathbf{B}_b$ and $\mathbf{v}_b$, to unclutter our equations.

We can use eq.(B.7) to set up the explicit differential equation for the body frame bivector B_b.

$$\begin{aligned}
\dot{B}_b &= \mathsf{I}_b{}^{-1}[B_b \times \mathsf{I}_b[B_b]] \\
&= -\boldsymbol{\epsilon}\,(\mathbf{v}\cdot\mathbf{B}) + \mathsf{I}_C^{-1}[\mathbf{B}\times\mathsf{I}_C[\mathbf{B}]] \\
&= -\boldsymbol{\epsilon}\,(\mathbf{v}\cdot\mathbf{B}) - \mathsf{I}_C^{-1}[(\mathsf{i}-\mathsf{i}_3)[\mathbf{B}]_3\,\mathbf{B}\times\mathbf{E}_3] \\
&= -\boldsymbol{\epsilon}\,(\mathbf{v}\cdot\mathbf{B}) - (1-\mathsf{i}_3/\mathsf{i})[\mathbf{B}]_3\,\mathbf{B}\times\mathbf{E}_3.
\end{aligned}$$

So the equation to be solved for B_b is:

$$\boldsymbol{\epsilon}\dot{\mathbf{v}} + \dot{\mathbf{B}} = -\boldsymbol{\epsilon}(\mathbf{v}\cdot\mathbf{B}) - \mathbf{B}\times\big((1-\mathsf{i}_3/\mathsf{i})\,[\mathbf{B}]_3\,\mathbf{E}_3\big) \tag{B.8}$$

We are allowed to consider the Euclidean and $\boldsymbol{\epsilon}$-parts separately, by Appendix C.4.

- It follows from the Euclidean part of eq.(B.8) that $\dot{\mathbf{B}}$ has a zero $\mathbf{E}_3$ component (due to the commutator product). Therefore $\mathbf{B}$ has a constant third component $[\mathbf{B}]_3\mathbf{E}_3$ as a constant of motion. We define the constant body frame bivector $\mathbf{A}$ (with the 'A' a mnemonic for the symmetry Axis) as

  $$\mathbf{A} = (1-\mathsf{i}_3/\mathsf{i})\,[\mathbf{B}]_3\,\mathbf{E}_3. \tag{B.9}$$

  (We will find a more compact expression for $\mathbf{A}$ below, after we have derived the constancy of the angular momentum.) Then the differential equation for $\mathbf{B}$ is the Euclidean part of eq.(B.8), i.e., $\dot{\mathbf{B}} = \mathbf{B}\times(-\mathbf{A})$. It is solved by

  $$\mathbf{B} = \widetilde{M}_{\mathbf{A}}\,\mathbf{B}_0\,M_{\mathbf{A}}, \quad \text{with} \quad M_{\mathbf{A}} \equiv e^{-\mathbf{A}t/2},$$

  with integration constant $\mathbf{B}_0$ the rotational angular velocity in the body frame at time 0. We see that $\mathbf{A}$ is interpreted as the constant component of the angular velocity around the symmetry axis $\mathbf{E}_3$, but that the object may also be rotating simultaneously around the other axes. (Below, we will see that eq.(B.7) is actually the conserved angular momentum $\mathbf{L}_b$; then $\mathbf{A}$, including its inertia-dependent scale factor, can be compactly characterized as 'the component of the angular momentum $\mathbf{L}_b$ along the symmetry axis'.)

- From the ideal part of eq.(B.8), we find the familiar equation

$$\boldsymbol{\epsilon}\,\dot{\mathbf{v}} = -\boldsymbol{\epsilon}\,(\mathbf{v}\cdot\mathbf{B}).$$

This may look like the ideal part of eq.(B.3) in the spherical top, but now we cannot just integrate it, since we have just seen that $\mathbf{B}$ is not constant: it is of the form $\mathbf{B} = M_{\mathbf{A}}\mathbf{B}_0\widetilde{M}_{\mathbf{A}}$, with $M_{\mathbf{A}}$ a time-varying motor. So let us massage the equation for $\dot{\mathbf{v}}$ by putting that motor $M_{\mathbf{A}}$ into the quantity to be determined:

$$\begin{aligned} &\tfrac{d}{dt}\big(M_{\mathbf{A}}(\boldsymbol{\epsilon}\mathbf{v})M_{\mathbf{A}}\big) && \text{(B.10)}\\ &= \big(M_{\mathbf{A}}(\boldsymbol{\epsilon}\mathbf{v})\widetilde{M}_{\mathbf{A}}\big)\times\mathbf{A} + M_{\mathbf{A}}(\boldsymbol{\epsilon}\dot{\mathbf{v}})\widetilde{M}_{\mathbf{A}}\\ &= \big(M_{\mathbf{A}}(\boldsymbol{\epsilon}\mathbf{v})\widetilde{M}_{\mathbf{A}}\big)\times\mathbf{A} + M_{\mathbf{A}}\big(\boldsymbol{\epsilon}(-\mathbf{v}\cdot\mathbf{B})\big)\widetilde{M}_{\mathbf{A}}\\ &= \big(M_{\mathbf{A}}(\boldsymbol{\epsilon}\mathbf{v})\widetilde{M}_{\mathbf{A}}\big)\times\mathbf{A} - (M_{\mathbf{A}}(\boldsymbol{\epsilon}\mathbf{v})\widetilde{M}_{\mathbf{A}})\cdot\mathbf{B}_0\\ &= -\big(M_{\mathbf{A}}(\boldsymbol{\epsilon}\mathbf{v})\widetilde{M}_{\mathbf{A}}\big)\times(\mathbf{B}_0-\mathbf{A}). && \text{(B.11)}\end{aligned}$$

The relevant quantity is apparently the constant bivector $\mathbf{B}_0 - \mathbf{A}$. We notice from eq.(B.7) that it can be written as $\mathsf{I}_C[\mathbf{B}_0]/\mathsf{i}$. The constant $\mathsf{I}_C[\mathbf{B}_0]$ is in fact the *angular momentum* $\mathbf{L}_b$ *in the body eigenframe* at time zero – and therefore, at any time. So let us denote

$$\mathbf{B}_0 - \mathbf{A} \equiv \mathbf{L}_b/\mathsf{i} \qquad \text{(B.12)}$$

Then eq.(B.11) integrates to $M_{\mathbf{A}}\mathbf{v}\widetilde{M}_{\mathbf{A}} = e^{(\mathbf{L}_b/\mathsf{i})t/2}\mathbf{v}_{w0}\,e^{-(\mathbf{L}_b/\mathsf{i})t/2}$. The integration constant is a constant velocity $\mathbf{v}_{w0}$ in the world frame. In fact, $m\mathbf{v}_{w0}$ is the classical *linear momentum vector* $\mathbf{p}$ and the derivation shows that linear momemtum is conserved under free motion of the symmetric top. The integration thus results in

$$\boldsymbol{\epsilon}\mathbf{v} = \widetilde{M}_{\mathbf{A}}\,\widetilde{M}_{\mathbf{L}_b}\,(\boldsymbol{\epsilon}\mathbf{v}_{w0})\,M_{\mathbf{L}_b}\,M_{\mathbf{A}}, \quad \text{with} \quad M_{\mathbf{L}_b} \equiv e^{-(\mathbf{L}_b/\mathsf{i})t/2}. \qquad \text{(B.13)}$$

The motor $M_{\mathbf{L}_b}M_{\mathbf{A}}$ produces a flowerlike orbit for any point of the object when viewed in a world frame at its centroid, 'an $\mathbf{A}$-rotation within an $\mathbf{L}_b$-rotation', see Figure B.2.

- Now that we realize that $\mathbf{L}_b$ is constant, we can write the expression eq.(B.9) for $\mathbf{A}$ more compactly, in terms of the preserved angular momentum $\mathbf{L}_b$ or the preserved rotation rate $\mathbf{B}_0$ around the symmetry axis:

$$\begin{aligned}\mathbf{A} &= \mathbf{B}_0 - \mathbf{L}_b/\mathsf{i}\\ &= (1-\mathsf{i}_3/\mathsf{i})\,[\mathbf{B}_0]_3\,\mathbf{E}_3\\ &= (\mathsf{i}_1-\mathsf{i}_3)\,[\mathbf{L}_b]_3\,\mathbf{E}_3. && \text{(B.14)}\end{aligned}$$

The constant rotation rate $\mathbf{A}$ around the symmetry axis is thus simply proportional to the $\mathbf{E}_3$-axis component of either $\mathbf{L}_b$ or $\mathbf{B}_0$.

- Combining the results for the rotational and translational parts, we have found B_b as a function of time:

$$B_b = \mathbf{B} + \boldsymbol{\epsilon}\mathbf{v} = \widetilde{M}_{\mathbf{A}}\big(\mathbf{B}_0 + \widetilde{M}_{\mathbf{L}_b}(\boldsymbol{\epsilon}\mathbf{p}/m)\,M_{\mathbf{L}_b}\big)\,M_{\mathbf{A}}.$$

in terms of its constants of motion $\mathbf{A}$, $\mathbf{L}_b$ and $\mathbf{p}$, and where $\mathbf{B}_0 = \mathbf{L}_b/\mathsf{i} + \mathbf{A}$.

Finally, we need to find the actual motor M from the body frame bivector B_b by integrating

$$\dot{M} = -\tfrac{1}{2}\, M\, B_b.$$

The time-varying nature of B_b makes this non-trivial. Let us first play the same trick as before, wrapping the disturbance around the variable:

$$\begin{aligned}
&\tfrac{d}{dt}(M_{\mathbf{A}} M \widetilde{M}_{\mathbf{A}}) = \\
&= (M_{\mathbf{A}} M \widetilde{M}_{\mathbf{A}}) \times \mathbf{A} + M_{\mathbf{A}} \dot{M} \widetilde{M}_{\mathbf{A}} \\
&= (M_{\mathbf{A}} M \widetilde{M}_{\mathbf{A}}) \times \mathbf{A} - \tfrac{1}{2}(M_{\mathbf{A}} M \widetilde{M}_{\mathbf{A}})\,(M_{\mathbf{A}}\, B_b \widetilde{M}_{\mathbf{A}}) \\
&= (M_{\mathbf{A}} M \widetilde{M}_{\mathbf{A}}) \times \mathbf{A} - \tfrac{1}{2}(M_{\mathbf{A}} M \widetilde{M}_{\mathbf{A}})\,(\mathbf{B}_0 + M_{\mathbf{L}_b}(\boldsymbol{\epsilon}\mathbf{v}_{w0})\, \widetilde{M}_{\mathbf{L}_b}) \\
&= -\tfrac{1}{2}\mathbf{A}\,(M_{\mathbf{A}} M \widetilde{M}_{\mathbf{A}}) - \tfrac{1}{2}(M_{\mathbf{A}} M \widetilde{M}_{\mathbf{A}})\,(\mathbf{B}_0 - \mathbf{A} + \widetilde{M}_{\mathbf{L}_b}(\boldsymbol{\epsilon}\mathbf{v}_{w0})\, M_{\mathbf{L}_b}) \\
&= -\tfrac{1}{2}\mathbf{A}\,(M_{\mathbf{A}} M \widetilde{M}_{\mathbf{A}}) - \tfrac{1}{2}(M_{\mathbf{A}} M \widetilde{M}_{\mathbf{A}})\,(\mathbf{L}_b/\mathsf{i} + \widetilde{M}_{\mathbf{L}_b}(\boldsymbol{\epsilon}\mathbf{v}_{w0})\, M_{\mathbf{L}_b}) \\
&= -\tfrac{1}{2}\mathbf{A}\,(M_{\mathbf{A}} M \widetilde{M}_{\mathbf{A}}) - \tfrac{1}{2}(M_{\mathbf{A}} M \widetilde{M}_{\mathbf{A}})\,(\mathbf{L}_b/\mathsf{i}) - \tfrac{1}{2}(M_{\mathbf{A}} M \widetilde{M}_{\mathbf{A}})\,(\widetilde{M}_{\mathbf{L}_b}(\boldsymbol{\epsilon}\mathbf{p}/m)\, M_{\mathbf{L}_b}).
\end{aligned}$$

where we used that $\mathbf{L}_b$ is invariant under the rotation by $M_{\mathbf{L}_b}$. In Appendix C.5 we show the straightforward technique to integrate this form. We then find from eq.(C.15) that it integrates to involve a motor for the bivector $\boldsymbol{\epsilon}\mathbf{p}/m$.

$$M_{\mathbf{A}} M \widetilde{M}_{\mathbf{A}} = M_{\mathbf{A}}\, M_{\mathbf{p}}\, M_0\, M_{\mathbf{L}_b} \quad \text{with} \quad M_{\mathbf{p}} \equiv e^{-(\boldsymbol{\epsilon}\mathbf{p}/m)t/2}.$$

Ultimately, the motor of the symmetric top is thus:

$$\text{symmetric top motor:} \quad M = M_{\mathbf{p}}\, M_0\, M_{\mathbf{L}_b}\, M_{\mathbf{A}} = M_{\mathbf{p}}\, M_{\mathbf{L}}\, M_0\, M_{\mathbf{A}}. \tag{B.15}$$

In the second expression, we defined the Euclidean 2-blade of constant angular momentum $\mathbf{L} \equiv \mathbf{L}_w$ in the world frame through $\mathcal{I}\mathbf{L}_w = M_0(\mathcal{I}\mathbf{L}_b)\widetilde{M}_0$, and its motor as $M_{\mathbf{L}_w} = \exp(-(\mathbf{L}/\mathsf{i})t/2)$. We could similarly move the motion constant $\mathbf{A}$ to its world frame representation.

An example of the motion of an object point under such a symmetric top motor is indicated in Figure B.3. It is of course essentially a translated version of the kind of motions depicted in Figure B.2, moved by the linear momentum. Setting $M_0 = 1$, we can visualize the total motion as a top spinning around its symmetry axis with body angular velocity $\mathbf{A}$, and that axis then spinning around the body angular momentum axis $\mathbf{L}_b$. The latter is called the *precession* of the axis of rotation. Depending on the relative values of the object parameters i and i_3, and the motion constants $\mathbf{L}$ and $\mathbf{A}$, points on the top trace out different curves. Note that if the inertias are all equal, so $\mathsf{i}_3 = \mathsf{i}$, only the rotation caused by the angular momentum $\mathbf{L}$ remains, as expected – this is the 'spherical' top of Section B.2.

> **Physical aside.** This natural motion of a free top is rather surprising (would you not expect the rotation to happen purely around the angular momentum, as in the spherically symmetric case?), and sufficiently counterintuitive that many computer-generated movies set in space get their rotational spacecraft motions wrong in this respect. But of course, if they would do it correctly, the audience might feel that it was wrong – the dilemma of realistic rendering...

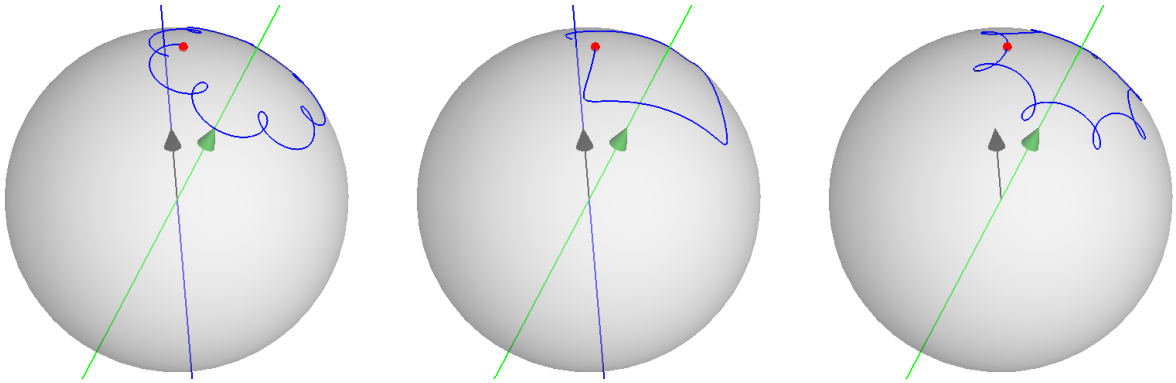


Figure B.2: *Implementation of the solution eq.(B.15) for the free symmetric top in 3D. A point* $\mathbf{x}$ *(red) on a free spinning symmetric top, close to its symmetry axis (in gray). The principal moments of inertia are* $\mathsf{i} = 1$, $\mathsf{i}_3 = 3$. *The angular momentum 2-blade* $\mathbf{L}_b$ *is indicated by its dual axis (green). Values of* $(1 - \mathsf{i}_3/\mathsf{i})$ *vary from* -4 *to* 2 *to* 5.

The above derivation extends the GA treatment of the symmetric top in Doran & Lasenby [9]. They solve it by a combination of classical GA at the body frame, and *a priori* physical insight (realizing the constancy of $\mathbf{L}$ and $\mathbf{p}$ at the start). As usual in physics texts, they treat the rotation only, as the most interesting part of the motion. Not having PGA or CGA at the time, they would not have been able to treat translation on the same footing anyway. We now see that, in the end, the top just translates by a final translation motor, so the classical separation of rotation and translation is (of course) fully justified. We just wanted to show that the complete derivation can be done in PGA, and how the motion constants $\mathbf{A}$, $\mathbf{L}$ and $\mathbf{p}$ are then forced upon us algebraically – there is no need for physical insights or experience to solve the equations.

B.4 Non-symmetric Free Top

The equations of the general free top moved by a motor $M = \exp(-B(t)/2)$ are still $\dot{\mathsf{I}}_w[B_w] = 0$. Let us work out the equations in the body frame; then $\dot{\mathsf{I}}_b[B_b] = 0$ becomes, in an object moved by the motor M:

$$0 = \tfrac{d}{dt}\mathsf{I}_b[B_b] = \mathsf{I}_b[\dot{B}_b] + \mathsf{I}_b[B_b] \times B_b,$$

by eq.(2.24). A natural choice for the body frame of the top is to take its origin at the centroid and the principal inertia directions as our frame directions. Then B_b is the purely Euclidean $\mathbf{B}_b$, and

$$0 = -\dot{\mathsf{I}}_b[B_b] = \mathsf{I}_C[\dot{\mathbf{B}}]\,\mathcal{I} + (\mathsf{I}_C[\mathbf{B}_b]\,\mathcal{I}) \times \mathbf{B}_b = (\mathsf{I}_C[\dot{\mathbf{B}}] + \mathsf{I}_C[\mathbf{B}_b] \times \mathbf{B}_b)\,\mathcal{I},$$

which due to the Euclidean nature of the factor of $\mathcal{I}$ (see Section C.1) gives

$$\mathsf{I}_C[\dot{\mathbf{B}}] = \mathbf{B}_b \times \mathsf{I}_C[\mathbf{B}_b].$$

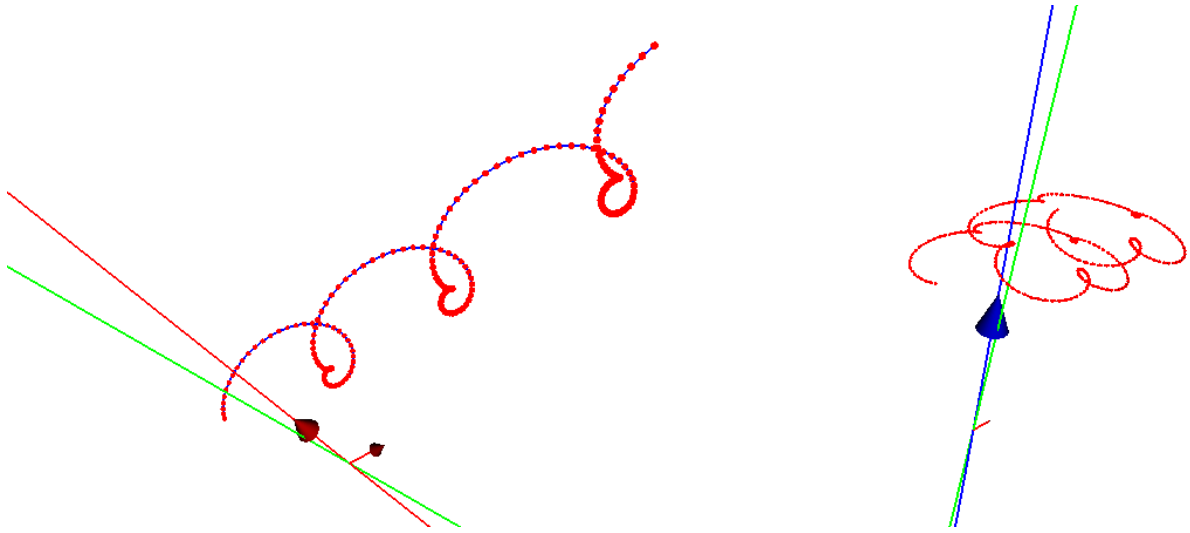


Figure B.3: *Implementation of the solution eq.(B.15) for the free symmetric top. A free symmetric top, with linear momentum. Two orbits with different choices of the constants of motion.*

We then obtain the two familiar differential equations whose solution produces the motor M:

$$\begin{cases} \dot{M} = -\frac{1}{2}\, M\, \mathbf{B}_b \\ \dot{\mathbf{B}} = \mathsf{I}_C^{-1}[\,\mathbf{B}_b \times \mathsf{I}_C[\mathbf{B}_b]\,] \end{cases} \tag{B.16}$$

We are not going to solve these, but instead point the interested reader to the implementation in the Ganja Coffeeshop at `enkimute.github.io/ganja/js/examples`. There the equations are solved numerically by the RK4 method, so you can see the free top tumble over the screen. Again, you can also roll you own ganja and change the sizes of the principal inertias to study the effect on the motion.

Especially interesting is the demonstration of the instability of the second principal axis, by specifying an initial rotation that is close to it: this is an illustration of the 'tennis racket theorem' (aka the *Dzhanibekov effect*). It is fairly simple to derive that when the principal inertias are ordered $\mathsf{i}_1 \geq \mathsf{i}_2 \geq \mathsf{i}_3 \geq 0$, the second rotation axis is unstable. For we compute from the orientational Euler equation in eq.(B.16):

$$\begin{aligned} \dot{\mathbf{B}} &= \dot{B}_{b1}\,\mathbf{E}_1 + \dot{B}_{b2}\,\mathbf{E}_2 + \dot{B}_{b3}\,\mathbf{E}_3 \\ &= \mathsf{I}_C^{-1}[(B_{b1}\,\mathbf{E}_1 + B_{b2}\,\mathbf{E}_2 + B_{b3}\,\mathbf{E}_3) \times (\mathsf{i}_1\, B_{b1}\,\mathbf{E}_1 + \mathsf{i}_2\, B_{b2}\,\mathbf{E}_2 + \mathsf{i}_3\, B_{b3}\,\mathbf{E}_3)] \\ &= \mathsf{I}_C^{-1}[(\mathsf{i}_2 - \mathsf{i}_3)\, B_{b2}B_{b3}\,\mathbf{E}_1 + (\mathsf{i}_3 - \mathsf{i}_1)\, B_{b3}B_{b1}\,\mathbf{E}_2 + (\mathsf{i}_1 - \mathsf{i}_2)\, B_{b1}B_{b2}\,\mathbf{E}_3] \\ &= \frac{\mathsf{i}_2 - \mathsf{i}_3}{\mathsf{i}_1}\, B_{b2}B_{b3}\,\mathbf{E}_1 + \frac{\mathsf{i}_3 - \mathsf{i}_1}{\mathsf{i}_2}\, B_{b3}B_{b1}\,\mathbf{E}_2 + \frac{\mathsf{i}_1 - \mathsf{i}_2}{\mathsf{i}_3}\, B_{b1}B_{b2}\,\mathbf{E}_3. \end{aligned} \tag{B.17}$$

Thus defining $\omega_1 = \frac{\mathsf{i}_2 - \mathsf{i}_3}{\mathsf{i}_1}$, and cyclic, the equations read

$$\begin{aligned} \dot{B}_{b1} &= \omega_1\, B_{b2}B_{b3} \\ \dot{B}_{b2} &= \omega_2\, B_{b3}B_{b1} \\ \dot{B}_{b3} &= \omega_3\, B_{b1}B_{b2}. \end{aligned} \tag{B.18}$$

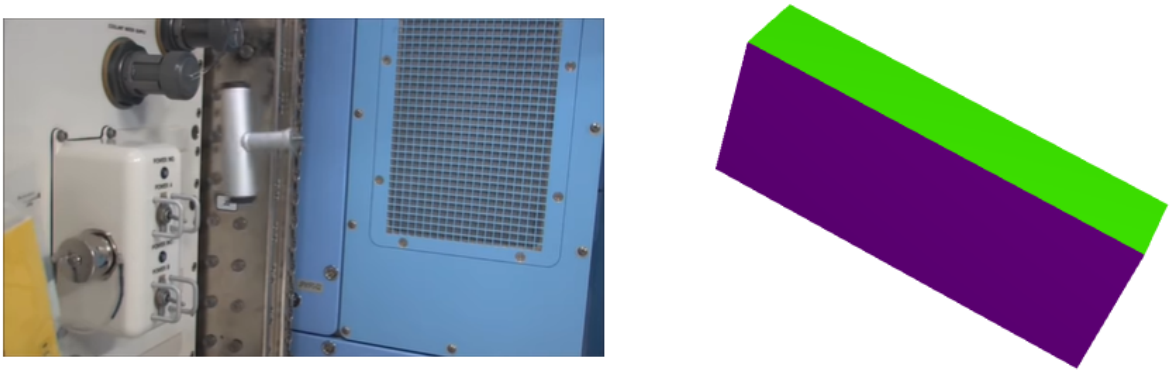


Figure B.4: *Playing with the Dzhanibekov/tennis racket effect on a free top in* `ganja.js`*, see* *`https://enki.ws/ganja.js/examples/pga_dyn.html#Dzhanibekov`*.

This appears symmetrical; but note that while ω_1 and ω_3 are positive, ω_2 is negative. If the object thus happens to be spinning with, say, positive values of B_{b1}, B_{b2}, B_{b3}, it immediately reduces the angular velocity B_{b2} and increases B_{b1} and B_{b3}. This shows that $\mathbf{E}_2$ is not a stable axis for rotations, unlike the axes $\mathbf{E}_1$ and $\mathbf{E}_3$. You can play with this example numerically, by eliminating the Gravity and Damping forques from the demo in Section 2.5.8.

B.5 Inertial Intuition

If all these subtleties already happen in free motion, without forques, how much more involved will the general case be? As Doran & Lasenby (in [9], pg.74) phrase the phenomena (our italics):

> In general, the total angular momentum will not lie in the same plane as the angular velocity. This is one reason why rigid-body dynamics can often seem quite counterintuitive. When we see a body rotating, our eyes naturally pick out the angular *velocity* by focusing on the vector the body rotates around. Deciding the plane of the angular *momentum* is less easy, particularly if the internal mass distribution is hidden from us.
>
> But it is the angular *momentum* that responds directly to the external torques, not the angular *velocity*, and this can have some unexpected consequences.

Even in the torqueless case, the tennis racket effect already shows counterintuitive aspects of Newtonian motion...

Appendix C

Background Material

C.1 Useful PGA Nuggets

When we start computing kinematics and dynamics, some routine computations recur. We collect some of those nuggets and their derivations here, so that we can refer to them and focus more fully on the physics than on the algebra. Skip them at first reading, and use them as needed; or view them as exercises to brush up your GA skills.

We give relationships for the PGA of d-dimensional Euclidean space, with pseudoscalar $\mathcal{I} = \boldsymbol{\epsilon}\,\mathbf{I}_d$.

- Inner and outer product between a vector and multivector:

$$x \cdot A = \tfrac{1}{2}\,(x\,A - \widehat{A}\,x) \tag{C.1}$$

$$x \wedge A = \tfrac{1}{2}\,(x\,A + \widehat{A}\,x), \tag{C.2}$$

  where $\widehat{A} = (-1)^{\text{grade}(A)}\,A$. This holds for general multivectors (with the sign swap considered per contained grade).
  If A is a *blade* and the vector x is 'in' the space spanned by that blade A (so that $x \wedge A = 0$), we have as commutation relationship:

$$x \in A \iff x\,A = -\widehat{A}\,x = (-1)^{\text{grade}(A)-1}\,A\,x. \tag{C.3}$$

- Duality relative to pseudoscalar.
  For multivectors A and B both residing in a subspace spanned by a pseudoscalar I, we have duality of inner and outer product with a vector:

$$(x \cdot A)\,I = x \wedge (A\,I) \tag{C.4}$$

$$(x \wedge A)\,I = x \cdot (A\,I). \tag{C.5}$$

  (The former only holds for *all* A if the inner product is taken to be the *contraction* [10].)

- Commutation of PGA pseudoscalar $\mathcal{I}_d = \boldsymbol{\epsilon}\mathbf{I}_d$ and Euclidean vector $\mathbf{v}$:

$$\begin{aligned} \mathcal{I}_d\,\mathbf{v} &= \boldsymbol{\epsilon}\,\mathbf{I}_d\,\mathbf{v} \\ &= (-1)^{d-1}\boldsymbol{\epsilon}\,\mathbf{v}\,\mathbf{I}_d \\ &= (-1)^{d}\,\mathbf{v}\,\mathcal{I}_d. \end{aligned} \tag{C.6}$$

- Points joined to vanishing points: show that $Q \vee (\mathbf{u}\mathcal{I}_d) = \mathbf{u} \cdot Q$.

  Solution: The join of a point Q (of grade d) and a direction $\mathbf{u}\mathcal{I}$ (of grade d) can be derived in two ways, using the join definition by reciprocal pseudoscalar from [13] (easy) or by Hodge duality (a bit more involved):

  $$Q \vee (\mathbf{u}\mathcal{I}_d) = \Big(\big((\mathbf{u}\mathcal{I}_d)\rfloor\mathcal{I}^r{}_d\big) \wedge (Q\rfloor\mathcal{I}^r{}_d)\Big)\rfloor\mathcal{I}_d = \big((\mathbf{u}\cdot Q)\rfloor\mathcal{I}^r{}_d\big)\rfloor\mathcal{I}_d = \mathbf{u}\cdot Q \tag{C.7}$$

  or, using the Hodge dual,

  $$Q \vee (\mathbf{u}\mathcal{I}_d) = \star^{-1}\Big((\star Q \wedge \star(\mathbf{u}\mathcal{I}_d)\Big) = \star^{-1}\big((\boldsymbol{\epsilon}+\mathbf{q}) \wedge \mathbf{u}\big) = \mathbf{u}\mathbf{I}_d + (\mathbf{u}\wedge\mathbf{q})\,\mathcal{I}_d = \mathbf{u}\cdot Q. \tag{C.8}$$

  For readers of [13] lower than version 2.0, we have now decided to redefine the join with a swapped order, namely as $A \vee B = \big((B\rfloor\mathcal{I}^r) \wedge (A\rfloor\mathcal{I}^r)\big)\rfloor\mathcal{I} = \star^{-1}(\star A \wedge \star B)$, to correspond better to the homogeneous-coordinate-aligned software. This also returns the Common Factor Axiom to the order it has in [3]: $(A\wedge B)\vee(B\wedge C) = (A\wedge B\wedge C)\vee B$.

- Commutator of direction $\mathbf{u}\mathcal{I}$ and bivector $B = \mathbf{B} + \boldsymbol{\epsilon}\mathbf{v}$

  $$\begin{aligned}(\mathbf{u}\mathcal{I}) \times B &= (\mathbf{u}\mathcal{I}) \times (\mathbf{B} + \boldsymbol{\epsilon}\mathbf{v}) \\ &= \tfrac{1}{2}\big(\mathbf{u}\,\mathbf{B} - \mathbf{B}\,\mathbf{u}\,\mathcal{I}\big) \\ &= (\mathbf{u}\cdot\mathbf{B})\,\mathcal{I}. \end{aligned} \tag{C.9}$$

  Only the Euclidean part $\mathbf{B}$ of B remains, to produce a vector direction (aka ideal point).

- Commutation with a bivector B:

  $$A * (A \times B) = 0 \tag{C.10}$$

  for a *bivector* B and arbitrary A. We show this using the cyclic property of the scalar product (see e.g. [10]), which we can use since the commutator with a bivector is grade-preserving on A: $2A * (A \times B) = \langle AAB - ABA\rangle_0 = \langle AAB\rangle_0 - \langle ABA\rangle_0 = \langle AAB\rangle_0 - \langle AAB\rangle_0 = 0$.

- Translation of a PGA multivector in a Euclidean split:

  $$\begin{aligned}\mathsf{T}_C[\mathbf{X} + \boldsymbol{\epsilon}\,\mathbf{Y}] &\equiv (1 + \boldsymbol{\epsilon}\,\mathbf{c}/2)\,(\mathbf{X} + \boldsymbol{\epsilon}\,\mathbf{Y})\,(1 - \boldsymbol{\epsilon}\,\mathbf{c}/2) \\ &= \mathbf{X} + \boldsymbol{\epsilon}\,\tfrac{1}{2}(\mathbf{c}\mathbf{X} - \widehat{\mathbf{X}}\,\mathbf{c}) + \boldsymbol{\epsilon}\,\mathbf{Y} \\ &= \mathbf{X} + \boldsymbol{\epsilon}(\mathbf{c}\cdot\mathbf{X}) + \boldsymbol{\epsilon}\,\mathbf{Y}, \end{aligned} \tag{C.11}$$

  with $\mathbf{X}$ and $\mathbf{Y}$ purely Euclidean. Thus Euclidean elements translate by adding an appropriate $\boldsymbol{\epsilon}$-part, and $\boldsymbol{\epsilon}$-parts themselves are translationally invariant.

- The *classical cross product* for vectors in 3D is embedded in 3D GA as the dual of an outer product:

  $$\mathbf{a} \times \mathbf{b} \equiv -(\mathbf{a} \wedge \mathbf{b})\,\mathbf{I}_3. \tag{C.12}$$

  In this text, we will mostly use the $\times$ symbol for the commutator product of geometric algebra. Only incidentally we revert to the cross product, to show correspondence to some classical 3D result, and then we will indicate this explicitly.

- Equations involving the non-invertible pseudoscalar $\mathcal{I}$.
  Because the pseudoscalar is not invertible, an equation like $X\,\mathcal{I} = A\,\mathcal{I}$ is not uniquely solvable as $X = A$. Any element $\boldsymbol{\epsilon}\,\mathbf{C}$ may be added to solution $X = A$, and the new X will also satisfy the equation. But if we know that both $\mathbf{X}$ and $\mathbf{A}$ are purely Euclidean, then $\mathbf{X}\,\mathcal{I} = \mathbf{A}\,\mathcal{I}$ is uniquely solved as $\mathbf{X} = \mathbf{A}$.

C.2 Canonical Decomposition of a Bivector

Lines in 3D PGA are 2-blades, factorizable by the outer product. The sum of 2-blades in the 4-dimensional representational space is no longer necessarily a 2-blade: it is a general bivector (which is not necessarily factorizable by the outer product). However, bivectors can always be split into a 2-blade L (representing a line or force) and an ideal line $L\mathcal{I}$ (where $\mathcal{I}$ is the homogeneous pseudoscalar).

We repeat the bivector split for a bivector B from [10].

> A bivector B can be decomposed as a sum of two commuting 2-blades using the formula
>
> $$B = \underbrace{B\left(1 - \tfrac{1}{2}\frac{B \wedge \widetilde{B}}{B \cdot \widetilde{B}}\right)}_{\text{Euclidean line}} + \underbrace{\tfrac{1}{2} B \frac{B \wedge \widetilde{B}}{B \cdot \widetilde{B}}}_{\text{vanishing line}}, \tag{C.13}$$
>
> unless $B \cdot \widetilde{B} = 0$; then B is already a vanishing line.

For a particular bivector of the form $B = \mathbf{B} + \boldsymbol{\epsilon}\mathbf{v}$, with $\mathbf{B}$ and $\mathbf{v}$ a Euclidean vector and bivector, respectively, this gives:

$$B \cdot \widetilde{B} = \langle (\mathbf{B} + \boldsymbol{\epsilon}\mathbf{v})\,(\widetilde{\mathbf{B}} - \boldsymbol{\epsilon}\mathbf{v}) \rangle_0 = \mathbf{B} \cdot \widetilde{\mathbf{B}}$$

and

$$B \wedge \widetilde{B} = \langle (\mathbf{B} + \boldsymbol{\epsilon}\mathbf{v})\,(\widetilde{\mathbf{B}} - \boldsymbol{\epsilon}\mathbf{v}) \rangle_4 = 2\boldsymbol{\epsilon}\,(\mathbf{v} \wedge \widetilde{\mathbf{B}})$$

so that

$$B\,\frac{B \wedge \widetilde{B}}{2\,B \cdot \widetilde{B}} = (\mathbf{B} + \boldsymbol{\epsilon}\mathbf{v})\,\boldsymbol{\epsilon}\,(\mathbf{v} \wedge \mathbf{B}^{-1}) = \boldsymbol{\epsilon}\,(\mathbf{v} \wedge \mathbf{B})/\mathbf{B}.$$

This leads to the 'Chasles split' of the bivector $\mathbf{B}$

$$B = \underbrace{\Big(\mathbf{B} - \boldsymbol{\epsilon}\,(\mathbf{v} \cdot \mathbf{B})/\mathbf{B}\Big)}_{\omega\,L} + \underbrace{\boldsymbol{\epsilon}\,(\mathbf{v} \wedge \mathbf{B})/\mathbf{B}}_{\nu\,L\mathcal{I}}.$$

showing the rotational and translational screw components ωL and $\nu L\mathcal{I}$. Compact code for this will appear in [6].

C.3 Commutation Rules

We will be mostly working in the PGA of 3D space, and its pseudoscalar $\mathcal{I} \equiv \boldsymbol{\epsilon}\mathbf{I}_3 = \boldsymbol{\epsilon}\,\mathbf{e}_1\mathbf{e}_2\mathbf{e}_3$. In many computations, we need to reshuffle geometric products to a standard form, so we need to be aware of the commutation rules. We display those for a PGA of d-dimensional space – in some applications like image processing $d = 2$ is also of interest.

Let us consider an element A of grade a, and denote it as $A = \mathbf{A}_a + \boldsymbol{\epsilon}\,\mathbf{A}_{a-1}$, the bold denoting Euclidean parts of the grade indicated. We are particularly interested in the commutation behavior relative to $\boldsymbol{\epsilon}$, $\mathbf{I}_d$ and $\mathcal{I} = \boldsymbol{\epsilon}\mathbf{I}_d$.

- Commuting with $\boldsymbol{\epsilon}$:

$$A\,\boldsymbol{\epsilon} = (\mathbf{A}_a + \boldsymbol{\epsilon}\mathbf{A}_{a-1})\boldsymbol{\epsilon} = \mathbf{A}_a\boldsymbol{\epsilon} = \boldsymbol{\epsilon}(-1)^a\mathbf{A}_a = \boldsymbol{\epsilon}\,\widehat{\mathbf{A}}_a = \boldsymbol{\epsilon}\,\widehat{A},$$

  where we introduced the usual hat notation for the grade inversion; and since $\boldsymbol{\epsilon}^2 = 0$, we can absorb the $\boldsymbol{\epsilon}$-part of A in the pattern.

- Commuting with $\mathbf{I}_d$:

$$\begin{aligned} A\,\mathbf{I}_d &= (\mathbf{A}_a + \boldsymbol{\epsilon}\mathbf{A}_{a-1})\mathbf{I}_d \\ &= \mathbf{I}_d\big((-1)^{a(d-1)}\mathbf{A}_a + (-1)^{(a-1)(d-1)+d}\boldsymbol{\epsilon}\mathbf{A}_{a-1}\big) \\ &= (-1)^{a(d-1)}\,\mathbf{I}_d(\mathbf{A}_a - \boldsymbol{\epsilon}\mathbf{A}_{a-1}\big) \\ &= (-1)^{ad}\,\mathbf{I}_d(\widehat{\mathbf{A}}_a + \boldsymbol{\epsilon}\widehat{\mathbf{A}}_{a-1}). \end{aligned}$$

- Commuting with $\mathcal{I} = \boldsymbol{\epsilon}\,\mathbf{I}_d$:

$$A\,\mathcal{I} = \mathbf{A}_a\,\mathcal{I} = (-1)^a(-1)^{a(d-1)}\mathcal{I}\mathbf{A}_a = (-1)^{ad}\,\mathcal{I}A.$$

In 3D (and general odd-D), the commutation rules are thus

$$A\,\boldsymbol{\epsilon} = \boldsymbol{\epsilon}\,\widehat{A}, \quad \mathbf{A}\,\mathbf{I}_3 = \mathbf{I}_3\,\mathbf{A}, \quad (\boldsymbol{\epsilon}\mathbf{A})\,\mathbf{I}_3 = -\mathbf{I}_3\,(\boldsymbol{\epsilon}\mathbf{A}), \quad A\,\mathcal{I} = \widehat{A}\,\mathcal{I}.$$

In 2D (and general even-D), the commutation rules are thus

$$A\,\boldsymbol{\epsilon} = \boldsymbol{\epsilon}\,\widehat{A}, \quad \mathbf{A}\,\mathbf{I}_2 = \mathbf{I}_2\,\widehat{\mathbf{A}}, \quad (\boldsymbol{\epsilon}\mathbf{A})\,\mathbf{I}_2 = \mathbf{I}_2\,(\boldsymbol{\epsilon}\widehat{\mathbf{A}}), \quad A\,\mathcal{I} = A\,\mathcal{I},$$

where now $\mathcal{I} = \boldsymbol{\epsilon}\mathbf{I}_2$.

C.4 Solving $\mathbf{p}\cdot C + \mathbf{P}\,\mathcal{I}$ Equations

In the main text, we unpeeled the momentum/forque equation into the classical force and torque equations for linear and angular momentum. Let us show that this is permitted.

Theorem 1 *For a point C (represented as a d-blade) and d-dimensional null pseudoscalar $\mathcal{I} \equiv \boldsymbol{\epsilon}\,\mathbf{I}_d$, the following linear bivector equation involving Euclidean vectors $\mathbf{p}$ and $\mathbf{q}$ and Euclidean bivectors $\mathbf{P}$ and $\mathbf{Q}$ can be solved simply by equating the corresponding parts:*

$$\mathbf{p}\cdot C + \mathbf{P}\,\mathcal{I} = \mathbf{q}\cdot C + \mathbf{Q}\,\mathcal{I} \iff \mathbf{p} = \mathbf{q} \;\; \textit{and} \;\; \mathbf{P} = \mathbf{Q}, \tag{C.14}$$

despite the non-invertibility of $\mathcal{I}$.

Proof: The term $\mathbf{p} \cdot C$ also contains $\mathcal{I}$-parts, so this it not immediate. Let us rewrite in terms of the origin point O which is represented by the Euclidean pseudoscalar $\mathbf{I}_d$. We rewrite:

$$\mathbf{p} \cdot C + \mathbf{P}\,\mathcal{I} = \mathbf{p} \cdot O + (\mathbf{c} \wedge \mathbf{p} + \mathbf{P})\,\mathcal{I},$$

and similar for the right hand side.

Then the Euclidean parts should be identical, giving $\mathbf{p} \cdot O = \mathbf{q} \cdot O$, so that $\mathbf{p}\,\mathbf{I}_d = \mathbf{q}\,\mathbf{I}_d$, and therefore $\mathbf{p} = \mathbf{q}$.

Also, the $\mathcal{I}$-parts should be identical, which by subtracting $\mathcal{I}(\mathbf{c}-\mathbf{p}) = \mathcal{I}(\mathbf{c}-\mathbf{q})$ from both sides reduces to $\mathbf{P}\,\mathcal{I} = \mathcal{I}\mathbf{Q}$. Rewriting that to $\boldsymbol{\epsilon}\,\mathbf{P}\,\mathbf{I}_d = \boldsymbol{\epsilon}\,\mathbf{Q}\,\mathbf{I}_d$, divide both sides by $\mathbf{I}_d$ to find $\boldsymbol{\epsilon}\,\mathbf{P} = \boldsymbol{\epsilon}\,\mathbf{Q}$. Even though $\boldsymbol{\epsilon}$ is not invertible, the assumed Euclidean nature of $\mathbf{P}$ and $\mathbf{Q}$ yields that they must be equal for this to hold. □

C.5 Motor Chain: Derivative and Integration

In the symmetric top example of Section B.3, we had three constants of motion: $\mathbf{A}$ was the rotational $\mathbf{E}_3$ component in the body frame, $\mathbf{L}$ was the angular momentum in the rotating frame, and $\mathbf{p}$ was the linear momentum vector in the world frame, leading to a PGA line momentum of $\mathbf{p} \cdot C$ through the centroid.

Let us call such invariant bivectors B_1, B_2, B_3 respectively, and denote their motors $M_i(t) = \exp(-\frac{1}{2}B_i t)$ (where we may drop the time-dependence as shorthand). Then the total motor M can be constructed from these, and from the constant motor that indicates the frame transformations between body and world frame. Body frame elements are denoted by a prime (and are hence to the right of M_0).

For three motors, this gives several alternative forms:

$$\begin{aligned} M &= M_3 M_0 M_2' M_1' \\ &= M_3 M_2 M_0 M_1' \\ &= M_3 M_2 M_1 M_0. \end{aligned}$$

Differentiating these then also gives different patterns to match for integration of M:

$$\begin{aligned} -2\dot{M} &= B_3\,M + M\,(\widetilde{M_1'} B_2' M_1') + M\,B_1' \\ &= B_3\,M + (M_3 B_2 \widetilde{M_3})\,M + M\,B_1' \\ &= B_3\,M + (M_3 B_2 \widetilde{M_3})\,M + M\,(\widetilde{M_0} B_1 M_0). \end{aligned}$$

Therefore once we have brought the differential equations in one of these forms, we can read off the motor M.

Note the subtlety in the placement of the reverses: body frame entities are reversed relative to the 'natural' order of world frame entities. The initial state motor M_0 could be placed anywhere, one should put it in the most natural place to parametrize the initial state.

Similar patterns can be developed for motors for more than three invariants, though then the combinatorics on the alternative forms that may arise starts to explode.

Appendix D

Exercises

In order to keep the main text readable, we have deferred some of the technical algebraic details to these Exercises. You can try your hand at them, or just read them as the derivations of results required in the main text.

1. Show that in 2D PGA, a bivector rate ωC, where C is the centroid of the object, leads to an inertia $\mathsf{I}[\omega C] = \omega \mathsf{i} \boldsymbol{\epsilon}$, where $\mathsf{i} = \Sigma_i m_i \mathbf{r}_i^2$ with $\mathbf{r}_i$ the location vector of mass point $m_i X_i$ relative to C. Numerically, this is what one would expect from a rotation around the centroid; the vector factor $\boldsymbol{\epsilon}$ denotes an ideal line, which we should apparently learn to interpret as that of a pure rotation around the centroid. (With Section 2.2.8.)

   Solution: $X_i \vee (X_i \times \omega C) = \omega\, X_i \vee (\mathbf{r}_i \mathcal{I}_2 \times \mathbf{I}_2) = \omega\, X_i \vee (\boldsymbol{\epsilon} \mathbf{r}_i) = \omega\, X_i \vee ((\mathbf{r}_i \mathbf{I}_2) \mathcal{I}_2) = \omega\, (\mathbf{r}_i \mathbf{I}_2) \cdot X_i$. Then $P = \Sigma_i \omega\, (\mathbf{r}_i \mathbf{I}_2) \cdot X_i = \omega \Sigma_i (\mathbf{r}_i \mathbf{I}_2) \cdot (\mathbf{r}_i \mathcal{I}_2) + 0 = \omega\, (\Sigma_i m_i \mathbf{r}_i^2)\, \boldsymbol{\epsilon}$.

2. Show that in 2D PGA, a bivector rate $\boldsymbol{\epsilon}\mathbf{v}$ leads to an inertia line $\mathsf{I}[\boldsymbol{\epsilon}\mathbf{v}] = m\, \mathbf{v} \cdot C$, where $m = \Sigma_i m_i$ and $C = \Sigma_i m_i X_i / m$ is the centroid of the object. This the motion line that one would expect from a translation. (With Section 2.2.8.)

   Solution: $\mathsf{I}[\boldsymbol{\epsilon}\mathbf{v}] = \Sigma_i m_i X_i \vee (X_i \times (\boldsymbol{\epsilon}\mathbf{v})) = \Sigma_i m_i X_i \vee (\mathbf{I}_2 \times (\boldsymbol{\epsilon}\mathbf{v})) = mC \vee (\mathbf{v}\mathcal{I}_2) = m\mathbf{v} \cdot C$.

3. The classical vector form of the inertia map eq.(2.16) is $\mathbf{a} \mapsto \Sigma_i m_i \mathbf{r}_i \times (\mathbf{a} \times \mathbf{r}_i)$. Show that this map is related to the GA-based bivector inertia as $\mathbf{a} \mapsto \mathsf{I}_C[\mathbf{a}\mathbf{I}_3]/\mathbf{I}_3$.

4. Show that $\mathsf{I}_C[\mathbf{B}]$ is a symmetric map, i.e., that $\mathbf{A} \cdot \mathsf{I}_C[\mathbf{B}] = \mathbf{B}_w \cdot \mathsf{I}_C[\mathbf{A}]$.

Solution:

$$\begin{aligned}
\mathbf{A} \cdot \mathsf{I}_C[\mathbf{B}] &= \mathbf{A} \cdot \Sigma_i\, m_i\, \mathbf{r}_i \wedge (\mathbf{r}_i \cdot \mathbf{B}) \\
&= -\mathbf{A} \cdot \Big(\Sigma_i\, m_i\, \mathbf{r}_i \cdot \big(\mathbf{r}_i \wedge (\mathbf{B}\mathbf{I}_3)\big)\Big)\, \mathbf{I}_3 \\
&= -\Big(\mathbf{A} \wedge \Sigma_i\, m_i\, \mathbf{r}_i \cdot \big(\mathbf{r}_i \wedge (\mathbf{B}\mathbf{I}_3)\big)\Big)\, \mathbf{I}_3 \\
&= -(\mathbf{A}\mathbf{I}_3) \cdot \Big(\Sigma_i\, m_i\, \mathbf{r}_i \cdot \big(\mathbf{r}_i \wedge (\mathbf{B}\mathbf{I}_3)\big)\Big) \\
&= -\Sigma_i\, m_i\, \big((\mathbf{A}\mathbf{I}_3) \wedge \mathbf{r}_i\big) \cdot \big(\mathbf{r}_i \wedge (\mathbf{B}\mathbf{I}_3)\big) \\
&= -\Sigma_i\, m_i\, \big((\mathbf{B}\mathbf{I}_3) \wedge \mathbf{r}_i\big) \cdot \big(\mathbf{r}_i \wedge (\mathbf{A}\mathbf{I}_3)\big) \\
&= \mathbf{B} \cdot \mathsf{I}_C[\mathbf{A}].
\end{aligned}$$

5. Continuing the eigenstructure of the classical inertia of Section A.5, Within the context of 3D PGA, we have three more dimensions $\mathcal{I}\mathbf{E}_j$. Establish that they are eigenblades as well, and compute their eigenvalues.

Solution:

$$\begin{aligned}
\mathsf{I}_C[\mathcal{I}_3\mathbf{E}_j] &= \mathsf{I}_C[\boldsymbol{\epsilon}\, \mathbf{E}_j\, \mathbf{I}_3] \\
&= \boldsymbol{\epsilon}\, \Sigma_i\, m_i\, \mathbf{r}_i \wedge (\mathbf{r}_i \cdot (\mathbf{E}_j\mathbf{I}_3)) \\
&= \boldsymbol{\epsilon}\, \Sigma_i\, m_i\, \big(\mathbf{r}_i \cdot (\mathbf{r}_i \wedge \mathbf{E}_j)\big)\mathbf{I}_3 \\
&= \Sigma_i\, m_i\, \big(\mathbf{r}_i^2\, \mathbf{E}_j - \mathbf{r}_i \wedge (\mathbf{r}_i \cdot \mathbf{E}_j)\big)\, \mathcal{I}_3 \\
&= \big(\Sigma\, m_i \mathbf{r}_i^2 - \mathsf{i}_j\big)\, \mathbf{E}_j\big)\, \mathcal{I}_3 \\
&= (mr^2 - \mathsf{i}_j)\, \mathcal{I}_3 \mathbf{E}_j.
\end{aligned}$$

Here we defined $r^2 = \Sigma_i m_i \mathbf{r}_i^2 / \Sigma m_i$ as a weighted mean squared size of the point cloud.

6. (Continuing from previous) As an additional exercise, you might show that mr^2 equals the trace $\Sigma_k \mathsf{i}_k$ of the classical inertia operator $\mathsf{I}_C[]$.

Solution: We can easily prove $mr^2 = \Sigma_j \mathsf{i}_j$ by decomposing each $\mathbf{r}_i$ in terms of the eigenvectors $\mathbf{E}_j\mathbf{I}_3$, and expanding:

$$\begin{aligned}
m\, r^2 &= \Sigma_i\, m_i \mathbf{r}_i^2 \\
&= \Sigma_{i,j}\, m_i \big(\mathbf{r}_i \cdot (\mathbf{E}_j\mathbf{I}_3)\big)^2 \\
&= \Sigma_{i,j}\, m_i \big((\mathbf{r}_i \wedge \mathbf{E}_j)\, \mathbf{I}_3)\big)^2 \\
&= \Sigma_{i,j}\, m_i (\mathbf{r}_i \wedge \mathbf{E}_j)\, (\widetilde{\mathbf{E}}_j \wedge \mathbf{r}_i) \\
&= \Sigma_{i,j}\, m_i (\widetilde{\mathbf{E}}_j \wedge \mathbf{r}_i) \cdot (\mathbf{r}_i \wedge \mathbf{E}_j) \\
&= \Sigma_j\, \widetilde{\mathbf{E}}_j \cdot \big(\Sigma_i\, m_i\, \mathbf{r}_i \cdot (\mathbf{r}_i \wedge \mathbf{E}_j) \\
&= \Sigma_j\, \widetilde{\mathbf{E}}_j \cdot (\mathsf{i}_j \mathbf{E}_j) \\
&= \Sigma_j\, \mathsf{i}_j.
\end{aligned}$$

7. (Continued) Write out the 3D classical inertia map $\mathsf{I}_C[]$ explicitly on the 3D PGA bivector basis.

   Solution:

$$\begin{aligned}&\mathsf{I}_C[\mathbf{B}' + \boldsymbol{\epsilon}\mathbf{v}'] \\ &= \mathsf{i}_1\, B'_1\mathbf{E}_1 + \mathsf{i}_2\, B'_2\mathbf{E}_2 + \mathsf{i}_3\, B'_3\mathbf{E}_3 \\ &\quad -(\mathsf{i}_2 + \mathsf{i}_3)\, v'_1\, \mathcal{I}_3\mathbf{E}_1 - (\mathsf{i}_3 + \mathsf{i}_1)\, v'_2\, \mathcal{I}_3\mathbf{E}_2 - (\mathsf{i}_1 + \mathsf{i}_2)\, v'_3\, \mathcal{I}_3\mathbf{E}_3\end{aligned}$$

8. (Continued) Show that the additional bivectors $\mathcal{I}_3\mathbf{E}_j$ of the classical inertia map $\mathsf{I}_C[]$ has no effect on kinematics.

   Solution: In the total inertia $\mathsf{I}_w[B_w]$, the classical inertia $\mathsf{I}_C[B_b]$ occurs with a factor $\mathcal{I}$, so only the Euclidean part remains:

$$\mathsf{I}_C[B_b]\,\mathcal{I} = \mathsf{I}_C[\mathbf{B}_b]\,\mathcal{I}_3.$$

9. Show that the 2D total inertia can be viewed as a matrix

$$\begin{bmatrix} 0 & 0 & \mathsf{i} \\ 0 & -m & 0 \\ m & 0 & 0 \end{bmatrix}$$

   on a properly chosen basis (which you should specify).

   Solution: $\mathsf{I}_b[\boldsymbol{\epsilon}\mathbf{e}_1] = m\star(\boldsymbol{\epsilon}\mathbf{e}_1) = m\mathbf{e}_2$, $\mathsf{I}_b[\boldsymbol{\epsilon}\mathbf{e}_2] = m\star(\boldsymbol{\epsilon}\mathbf{e}_2) = -m\mathbf{e}_1$, $\mathsf{I}_b[\mathbf{e}_1\mathbf{e}_2] = \mathsf{i}\star(\mathbf{e}_1\mathbf{e}_2) = \mathsf{i}\boldsymbol{\epsilon}$. So for the given matrix, the input basis was $\{\boldsymbol{\epsilon}\mathbf{e}_1, \boldsymbol{\epsilon}\mathbf{e}_2, \mathbf{e}_1\mathbf{e}_2\}$, and the output basis $\{\boldsymbol{\epsilon}, \mathbf{e}_1, \mathbf{e}_2\}$. Note that meet lines (which in 2D PGA are bivector points) are transformed to lines (which in 2D PGA are vectors).

10. A special case of the PGA inertia is that of a single mass point. Investigate it, and make sensible choices for the inverse to replace eq.(2.21).

    Solution: In this case, the principal moments are zero, so $\mathsf{I}_b[\,]$ becomes purely Euclidean and the Euclidean part of its argument plays no role:

$$\text{for a mass point:}\quad \mathsf{I}_b[\mathbf{B}_b + \boldsymbol{\epsilon}\mathbf{v}_b] = m\mathbf{v}_b O.$$

    (For instance, in 3D PGA the $\mathcal{I}_3\mathbf{e}_{23} = \boldsymbol{\epsilon}\mathbf{e}_1$ coefficient of the input is v_1, and that turns into $v_1\mathbf{e}_{23} = v_1\mathbf{e}_1\mathbf{I}_3 = [\mathbf{v}_b]_1 O$; and cyclic.) Thus only the translational motion of the rate plays a role in the motion of a point (as one would expect).

    For a point, the inertia is clearly non-invertible. If we take the limit of the point as a very small ball with principal moments $\mathsf{i}_i = \rho$, and take the limit $\rho \to 0$, then

$$\text{for a mass point:}\quad \mathsf{I}_b{}^{-1}[\mathbf{B}_b + \boldsymbol{\epsilon}\mathbf{v}_b] = 1/0\mathbf{v}_b \cdot O + \mathcal{I}(\star\mathbf{B})/m.$$

    We can therefore only sensibly invoke the inverse inertia with purely Euclidean arguments; this will amount to again ignoring the rotational aspects of the point mass.

    Just to verify signs: $\mathsf{I}_b{}^{-1}[\mathsf{I}_b[\boldsymbol{\epsilon}\mathbf{v}_b]] = \mathsf{I}_b{}^{-1}[m\mathbf{v}_b O] = \mathcal{I}\mathbf{v}_b\widetilde{O} = \boldsymbol{\epsilon}\mathbf{v}_b$.

11. (Continued) Determine Newton's equation for the motion of a mass point, using the above inverse in eq.(2.28).

    Solution: For a point mass, we have to reduce the rate to the translational part $\boldsymbol{\epsilon}\mathbf{v}$, or the inverse inertia map does not exist (see Exercise 10). So set $B_b = \boldsymbol{\epsilon}\mathbf{v}_b$ (see Section 2.1.2 to see why the body frame velocity rate is not zero!). Then during the computation we find that we also have to limit the force F_b to its Euclidean part $\mathbf{f}_b \cdot O$. We then obtain

$$\begin{aligned}
\dot{B}_b &= \boldsymbol{\epsilon}\,\dot{\mathbf{v}}_b \\
&= \mathsf{I}_b{}^{-1}[(\boldsymbol{\epsilon}\,\mathbf{v}_b) \times \mathsf{I}_b[\boldsymbol{\epsilon}\,\mathbf{v}_b] + F_b] \\
&= \mathsf{I}_b{}^{-1}[(\boldsymbol{\epsilon}\,\mathbf{v}_b) \times (m\mathbf{v}_b O) + \mathbf{f}_b \cdot O] \\
&= \mathsf{I}_b{}^{-1}[0 + \mathbf{f}_b \cdot O] \\
&= \boldsymbol{\epsilon}\,\mathbf{f}_b/m.
\end{aligned}$$

    Hence we retrieve Newton's law for a point mass: $\mathbf{f}_b = m\dot{\mathbf{v}}_b$ in the body frame. Transforming to the world frame, the rate equation $\boldsymbol{\epsilon}\dot{\mathbf{v}}_w = \boldsymbol{\epsilon}\mathbf{f}_w/m$ similarly leads to $\mathbf{f}_w = m\dot{\mathbf{v}}_w$. This very degenerate case of a moving infinitesimal mass point will not occur in practice, but it is comforting to see it included in the limit.

12. Compute the radius of a unit weight sphere that has the Hodge dualization of bivectors as its body inertia map, i.e., $\mathsf{I}_b[B] = \star B$.

    Solution: Since the moment of inertia of a sphere with uniform mass m and radius r equals $\frac{2}{5}mr^2$ (according to standard tables), it follows that a sphere with radius $\sqrt{5/2}$ and $m = 1$ has a body frame inertia that effectively makes an angular momentum equal to the Hodge dual of any bivector you give it.

13. Take an object with unit mass $m = 1$ and $\mathsf{I}_b[B] = \star B$. Write the equation of motion eq.(2.28) in terms of the Hodge dual.

    Solution: For the bivectors involved, the Hodge undual is the same as the Hodge dual, for all dimension-dependent signs cancel. Therefore

$$\dot{B}_b = \star\tfrac{1}{2}\big(B_b \star B_b - \star B_b\, B_b + 2F_b\big).$$

14. The body frame inertia has a special form which allows a minor additional simplification, requiring only the Euclidean part of B_b in the commutator product. Derive that $\mathsf{I}_b[B_b] \times B_b = \mathsf{I}_b[B_b] \times \mathbf{B}_b$.

    Solution: We have computed $\mathsf{I}_b[\mathbf{B}_b + \boldsymbol{\epsilon}\mathbf{v}_b] = m\mathbf{v}_b \cdot O + \mathsf{I}_C[\mathbf{B}_b]\mathcal{I}$, so the commutator

can be computed more explicitly.

$$\begin{aligned}
\mathsf{I}_b[B_b] \times B_b &= \mathsf{I}_b[\mathbf{B}_b + \boldsymbol{\epsilon}\mathbf{v}_b] \times (\mathbf{B}_b + \boldsymbol{\epsilon}\mathbf{v}_b) \\
&= (m\,\mathbf{v}_b \cdot O + \mathsf{I}_C[\mathbf{B}_b]\,\mathcal{I}) \times (\mathbf{B}_b + \boldsymbol{\epsilon}\mathbf{v}_b) \\
&= (m\,\mathbf{v}_b \cdot O + \mathsf{I}_C[\mathbf{B}_b]\,\mathcal{I}) \times \mathbf{B}_b + m\,(\mathbf{v}_b \cdot O) \times (\boldsymbol{\epsilon}\mathbf{v}_b) + 0 \\
&= (m\,\mathbf{v}_b \cdot O + \mathsf{I}_C[\mathbf{B}_b]\,\mathcal{I}) \times \mathbf{B}_b + 0 \\
&= \mathsf{I}_b[\mathbf{B}_b + \boldsymbol{\epsilon}\mathbf{v}_b] \times \mathbf{B}_b \\
&= \mathsf{I}_b[B_b] \times \mathbf{B}_b.
\end{aligned}$$

15. (continued from previous) Show that the Euclidean part of the body frame commutator can be rewritten using

$$\mathbf{B} \times (\mathbf{v} \cdot O) = -(\mathbf{v} \cdot \mathbf{B}) \cdot O$$

.

Solution:

$$\begin{aligned}
4\mathbf{B} \times (\mathbf{v} \cdot O) &= 2\mathbf{B}(\mathbf{v} \cdot O) - (\mathbf{v} \wedge O)\mathbf{B} \\
&= \mathbf{B}\mathbf{v}O - \mathbf{B}\widehat{O}\mathbf{v} - \mathbf{v}O\mathbf{B} + \widehat{O}\mathbf{v}\mathbf{B} \\
&= \mathbf{B}\mathbf{v}O + \mathbf{B}\mathbf{v}O - \mathbf{v}\mathbf{B}O - \mathbf{v}\mathbf{B}O \\
&= 2(\mathbf{B}\mathbf{v} - \mathbf{v}\mathbf{B})O \\
&= -4(\mathbf{v} \cdot \mathbf{B}) \cdot O.
\end{aligned}$$

16. Show that the derivative of the inertia in the world frame of is

$$\tfrac{d}{dt}\,\mathsf{I}_w[B_w] = \mathsf{I}_w[B_w] \times B_w + \mathsf{I}_w[(MB_w\widetilde{M}) \times B_w + \dot{B}_w]. \tag{D.1}$$

Solution:

$$\begin{aligned}
\tfrac{d}{dt}\,\mathsf{I}_w[B_w] &= \tfrac{d}{dt} M\,\mathsf{I}_b[\widetilde{M}\,B_w\,M]\,\widetilde{M} \\
&= \mathsf{I}_w[B_w] \times B_w + M\,(\tfrac{d}{dt}\,\mathsf{I}_b[\widetilde{M}\,B_w\,M])\,\widetilde{M} \\
&= \mathsf{I}_w[B_w] \times B_w + M\,\mathsf{I}_b[B_w \times (\widetilde{M}B_wM) + \widetilde{M}\,\dot{B}_w\,M]\,\widetilde{M} \\
&= \mathsf{I}_w[B_w] \times B_w + \mathsf{I}_w[(MB_w\widetilde{M}) \times B_w + \dot{B}_w].
\end{aligned}$$

The quantity $MB_w\widetilde{M} = M^2B_b\widetilde{M}^2$ is strange: it is a world quantity transformed by M, so a body quantity transformed by M^2. Yet the commutator occurring above, $(MB_w\widetilde{M}) \times B_w$, is not zero, so the term does contribute.

17. The expression $\mathbf{f} \cdot Q$ takes the correct representational form automatically; it truly is just a line, in any dimension. Show this, and especially appreciate what happens in 2D.

Solution: We wrote O to represent the origin, which in the PGA of d-dimensional Euclidean space is the Euclidean pseudoscalar $\mathbf{I}_d = \mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \cdots \wedge \mathbf{e}_d$ (since it is the

intersection of the d coordinate hyperplanes of the chosen coordinate frame). The same applies to a general point Q, with properly displaced hyperplanes.

The expression $\mathbf{f} \cdot Q$ is then the intersection with the additional hyperplane $\mathbf{f}$, indeed producing a line. In 2D PGA, $\mathbf{f} \cdot Q$ is a vector, and thus a hyperplane – which in 2D is a line.

18. Use Appendix C.2 to split off the ideal line $L\mathcal{I}_3$ from a total forque (aka wrench) in 3D PGA given by $F = \mathbf{f} \cdot O - \mathbf{T}_O\mathcal{I}_3$.
Solution:

$$\begin{aligned}
L\mathcal{I}_3 &= (\mathbf{f}\cdot\mathbf{I}_3 - \mathbf{T}_O\mathcal{I}_3)\left((\mathbf{f}\cdot\mathbf{I}_3 - \mathbf{T}\mathcal{I}_3)\wedge(-\mathbf{f}\cdot O + \mathbf{T}_O\mathcal{I}_3)\right)/(\mathbf{f}\cdot\mathbf{I}_3)^2 \\
&= (\mathbf{f}\cdot\mathbf{I}_3)\left((\mathbf{f}\cdot\mathbf{I}_3)\wedge(\mathbf{T}_O\mathcal{I}_3)\right)/(\mathbf{f}^2) \\
&= \boldsymbol{\epsilon}\,\mathbf{f}^{-1}\,\mathbf{I}_3\left(\mathbf{f}\mathbf{I}_3\wedge\mathbf{T}_O\mathbf{I}_3\right) \\
&= \boldsymbol{\epsilon}\,\mathbf{f}^{-1}\left(\mathbf{f}\mathbf{I}_3\wedge\mathbf{T}_O\mathbf{I}_3\right)\mathbf{I}_3 \\
&= -\boldsymbol{\epsilon}\,\mathbf{f}^{-1}\left(\mathbf{f}\mathbf{I}_3\cdot\mathbf{T}_O\right) \\
&= -\boldsymbol{\epsilon}\,\mathbf{f}^{-1}\left(\mathbf{T}_O\cdot\mathbf{f}\mathbf{I}_3\right) \\
&= -\boldsymbol{\epsilon}\,\mathbf{f}^{-1}\left((\mathbf{T}_O\wedge\mathbf{f})\mathbf{I}_3\right) \\
&= -\boldsymbol{\epsilon}\left(\mathbf{f}^{-1}\cdot(\mathbf{f}\wedge\mathbf{T}_O)\right)\mathbf{I}_3 \\
&= -\left(\mathbf{f}^{-1}\cdot(\mathbf{f}\wedge\mathbf{T}_O)\right)\mathcal{I}_3.
\end{aligned}$$

We can see this $L\mathcal{I}_3$ as the measure of 'screwyness': if it is zero, L is a line 2-blade rather than a screw 2-vector. Note that it is independent of the choice of origin, despite appearances: $\mathbf{f}\wedge\mathbf{T}_O = \mathbf{f}\wedge\mathbf{T}_Q$ for any Q.

You could also solve this problem somewhat more directly from the split expression $L\mathcal{I}_3 = \boldsymbol{\epsilon}(\mathbf{v}\wedge\mathbf{B})/\mathbf{B}$ using $\mathbf{B} = \mathbf{f}\mathbf{I}_3$ and $\mathbf{v} = \tau = \mathbf{T}/\mathbf{I}_3$.

19. Rewrite eq.(A.22) as a Euclidean split, and show that the resulting equations of motion are still the same as eq.(A.23) and eq.(A.24).

Solution:

$$m\,(\dot{\mathbf{v}}_w + \mathbf{c}\cdot\dot{\mathbf{B}}_w)\cdot O - \left(\mathbf{c}\wedge(\dot{\mathbf{v}}_w + \mathbf{c}\cdot\dot{\mathbf{B}}_w) + M\,\mathsf{I}_C[\widetilde{R}\dot{\mathbf{B}}_w R]\,\widetilde{M}\right)\mathcal{I}.$$

20. When processing constraints, we may come across the equation

$$B\times X = C, \quad \text{for a 2-blade } B.$$

Solve this!

Solution: This is only solvable if B and C anti-commute: $BC = -CB$ (Proof: add equation multiplied on left and right with B, and use the fact that B^2 is scalar). The solution is then $X = B^{-1}\times C$, modulo any element that commutes with B. (Proof: easily verified by substitution.)

21. **Lagrangian**
Fill in the details of the derivation of the Lagrangian-based equation 'invoking some standard differentiation results'.

Solution: First, realize that the join-based Lagrangian can be written as a scalar-product based Lagrangian:

$$\langle A \vee B \rangle = \langle B \mathcal{I}^r A \rangle. \tag{D.2}$$

(Show this yourself.) Then we have some standard results for the derivatives of a scalar product using its symmetries:

$$\partial_\psi \langle B\, A \rangle = -2\, \partial_\psi \langle \psi^{-1}\, \dot{\psi}\, A \rangle = 2\, \psi^{-1}\, \dot{\psi}\, A\, \psi^{-1}$$

$$\partial_{\dot{\psi}} \langle B\, A \rangle = -2\, \partial_{\dot{\psi}} \langle \psi^{-1}\, \dot{\psi}\, A \rangle = -2\, \partial_{\dot{\psi}} \langle \dot{\psi}\, A\, \psi^{-1} \rangle = -2\, A\, \psi^{-1}.$$

It follows that

$$\tfrac{d}{dt} \partial_{\dot{\psi}} \langle B\, A \rangle = -2\, \dot{A}\, \psi^{-1} + 2\, A\, \psi^{-1}\, \dot{\psi}\, \psi^{-1}.$$

We thus obtain from differentiating a scalar-product-lagrangian:

$$\begin{aligned} 0 &= \partial_\psi \langle B\, A \rangle - \tfrac{d}{dt} \partial_{\dot{\psi}} \langle B\, A \rangle \\ 0 &= \psi^{-1}\, \dot{\psi}\, A\, \psi^{-1} + \dot{A}\, \psi^{-1} - A\, \psi^{-1} \dot{\psi} \psi^{-1} \\ 0 &= \psi^{-1}\, \dot{\psi}\, A + \dot{A} - A\, \psi^{-1} \dot{\psi} \\ \dot{A} &= B \times A \end{aligned}$$

Back to the inertia, substitute $A = B\mathcal{I}^r$ to obtain eq.(2.38). The main text then notes that in our identity eq.(D.2) we might also have written $\mathcal{I}^r B$ rather than $B\mathcal{I}^r$. With that symmetry, the ultimate result $\mathsf{I}[\dot{B}] = B \times \mathsf{I}[B]$ follows. You can remove the left $\mathcal{I}^r$ by a dot product with $\boldsymbol{\epsilon}$, then left multiply by the inverse of $\mathbf{I}_d$.

22. **6D Coordinate Frameworks**
From the PGA point of view, Plücker coordinates (from line coordinate representation) and pure dual quaternions (from motion representation) are very closely related. Explain how.

Bibliography

[1] V.I. Arnold. *Mathematical Methods of Classical Mechanics*. Springer, 1978, 1989.

[2] Stéphane Breuils, Vincent Nozick, Akihiro Sugimoto, and Eckhard Hitzer. Quadric conformal geometric algebra in $\mathbb{R}(9,6)$. *Adv. Appl. Clifford Algebras*, 28, 2018.

[3] J. Browne. *Grassmann Algebra. Volume 1: Foundations*. Barnard Publishing, 2012.

[4] Chakravala. `Grassmann.wl`, 2021. Wolfram Research, `https://github.com/chakravala/Grassmann.wl`.

[5] Patrick Charrier and Dietmar Hildenbrand. `Gaalop` algorithm optimizer, 2021. `https://github.com/CallForSanity/Gaalop` and `www.gaalop.de`.

[6] Stephen De Keninck and Martin Roelfs. Polar decomposition, normalization, square roots and the exponential map in Clifford algebras of fewer than 6 dimensions (working title). In preparation 2022.

[7] Steven De Keninck. `GAamphetamine` symbolic manipulator, 2022. To be released.

[8] Chris Doran. Euclidean geometry and geometric algebra, June 2020. Blogpost `https://geometry.mrao.cam.ac.uk/2020/06/euclidean-geometry-and-geometric-algebra/`.

[9] Chris Doran and Anthony Lasenby. *Geometric Algebra for Physicists*. Cambridge University Press, 2003.

[10] L. Dorst, D. Fontijne, and S. Mann. *Geometric Algebra for Computer Science: An Object-oriented Approach to Geometry*. Morgan Kaufman, 2009.

[11] L. Dorst and R.J. Valkenburg. Square root and logarithm of rotors in 3D conformal geometric algebra using polar decomposition. In L. Dorst and J. Lasenby, editors, *Guide to Geometric in Practice*, pages 81–104. Springer-Verlag, 2011.

[12] Leo Dorst. 3D oriented projective geometry through versors of $\mathbb{R}^{3,3}$. *Advances in Applied Clifford Algebras*, 26:1137–1172, 2016.

[13] Leo Dorst and Steven De Kenink. Guided tour to the plane-based geometric algebra PGA (version 2.0), February 2022. Available at `bivector.net/PGA4CS.html`.

[14] R. Featherstone. The acceleration vector of a rigid body. *Int. Jnl. of Robotics Research*, 20:841–846, 2001.

[15] R. Featherstone. *Rigid Body Dynamics Algorithms*. Springer Verlag, 2008.

[16] R. Featherstone. A beginner's guide to 6-d vectors (part 1). *IEEE Robotics and Automation Magazine*, September:83–94, 2010.

[17] Ch. Gunn. *Geometry, Kinematics, and Rigid Body Mechanics in Cayley-Klein Geometries*. PhD thesis, TUBerlin, 2011.

[18] Charles Gunn. On the homogeneous model of Euclidean geometry. In L. Dorst and J. Lasenby, editors, *Guide to Geometric in Practice*, pages 297–327. Springer-Verlag, 2011.

[19] Hugo Hadfield and Joan Lasenby. Constrained dynamics in conformal and projective geometric algebra. In N. Magnenat-Thalmann, C. Stephanidis, E. Wu, D. Thalmann, B. Sheng, J. Kim, G. Papagiannakis, and M. Gavrilova, editors, *Advances in Computer Graphics*, pages 459–471, Cham, 2020. Springer International Publishing.

[20] David Hestenes. *New Foundations for Classical Mechanics*. Reidel, 2nd edition, 2000.

[21] Jeremy Ong. `GAL` library, 2021. Warner Bros, `https://github.com/jeremyong/gal`.

[22] N. Roelfs and S. De Keninck. Graded symmetry groups: Plane and simple, 2021. Available at `https://arxiv.org/abs/2107.03771`.

[23] Wikipedia. `https://en.wikipedia.org/wiki/Collision_response#Impulse-based_contact_model`. Consulted December 2021.

Index