Geometric Discrete Variational Dynamics — Part 1: Free Bodies

Brian Beckman
April 2024

Abstract

Prior Failures

The Future: Discretized Lagrangians on Lie Groups

References

1. Jack B Kuipers, *Quaternions and Rotation Sequences*, Princeton University Press, 1999.
2. Jeongseok Lee, Karen Liu, Frank C. Park, Siddartha S. Srinivasa, *A Linear-Time Variational Integrator for Multibody Systems*, 2018, https://arxiv.org/abs/1609.02898
3. Ari Stern, *Discrete Geometric Mechanics and Variational Integrators*, 2006, http://ddg.cs.columbia.edu/SIGGRAPH06/stern-siggraph-talk.pdf.
4. Ethan Eade, *Lie Groups for 2D and 3D Transformations*, 2017, http://ethaneade.com/lie.pdf.
5. NIST: *Digital Library of Mathematical Functions*, https://dlmf.nist.gov.
6. Brian C. Hall, *Lie Groups, Lie Algebras, and Representations*, Second Edition, 2015, Springer.
7. Guangyu Liu, *Modeling, stabilizing control and trajectory tracking of a spherical inverted pendulum*, PhD thesis, 2007, University of Melbourne, https://minerva-access.unimelb.edu.au/handle/11343/37225.
8. Wikipedia, *Tennis-Racket Theorem*, https://en.wikipedia.org/wiki/Tennis_racket_theorem.
9. Marin Kobilarov, Keenan Crane, Mathieu Desbrun, *Lie Group Integrators for Animation and Control of Vehicles*, 2009 (?).
10. Ari Stern, Mathieu Desbrun, *Discrete Geometric Mechanics for Variational Time Integrators*, 2006 (http://www.geometry.caltech.edu/pubs/SD06.pdf).

11. Kenth Engø, *On The BCH Formula in $\mathfrak{so}(3)$*, https://www.researchgate.net/profile/Kenth_Engo-Monsen2/publication/233591614_On _the _BCH-formula_in _so3/links/004635199177f69467000000/On-the-BCH-formula-in-so3.pdf.
12. Alexander Van-Brunt, Max Visser, *Explicit Baker-Campbell-Hausdorff formulae for some specific Lie algebras*, https://arxiv.org/pdf/1505.04505.pdf.
13. Wikipedia, *Baker-Campbell-Hausdorff Formula*, https://en.wikipedia.org/wiki/Baker% E2 %80 %93 Campbell % E2 %80 %93 Hausdorff_formula.
14. Jerrold E. Marsden, Tudor S. Ratiu, *Introduction to Mechanics and Symmetry*, Second Edition, 2002
15. Moylan, Andrew, *Stabilized Inverted Pendulum*, https://blog.wolfram.com/2011/01/19/stabilized-inverted-pendulum/, and *Stabilized n-Link Pendulum*, https://blog.wolfram.com/2011/03/01/stabilized-n-link-pendulum/
16. Paul R. Halmos, *Naive Set Theory*, 2015.
17. David Lovelock and Hanno Rund, *Tensors, Differential Forms, and Variational Principles*, 1975.
18. Ralph Abraham and Jerrold E. Marsden, *Foundations of Mechanics: 2nd Edition*, 1980.
19. Taeyoung Lee, Melvin Leok, N. Harris McClamroch, *Discrete Control Systems*, https://arxiv.org/abs/0705.3868.
20. Taeyoung Lee, Melvin Leok, N. Harris McClamroch, *Global Formulations of Lagrangian and Hamiltonian Mechanics on Manifolds*, Springer, https://a.co/d/0lTbL9t.
21. Tristan Needham, *Visual Differential Geometry and Forms*, https://a.co/d/9hoKekz.
22. *xAct, Efficient tensor computer algebra for the Wolfram Language*, http://xact.es/index.html.
23. Blanco, Jose-Luis, *A tutorial on SE(3) transformation parameterizations and on-manifold optimization*, Technical Report #012010, Universidad de Malaga (https://jinyongjeong.github.io/Download/SE3/jlblanco2010geometry3d_techrep.pdf).
24. Marsden and West has exhaustive treatment of Noether’s Theorem (http://www.cds.caltech.edu/~marsden/bib/2001/09-MaWe2001/MaWe2001.pdf).
25. Maxime Tournier, *Notes on Lie Groups*, https://maxime-tournier.github.io/notes/lie-groups.html.
26. John Baez, Javier P. Muniain, *Gauge Fields, Knots and Gravity*, (https://a.co/d/jdRnOU0).
27. F. C. Park, J. E. Bobrow, S. R. Ploen, *A Lie Group Formulation of Robot Dynamics.*
28. George W. Hart, *Multidimensional Analysis, https://a.co/d/8c5RqOX*

Quaternions and RK4

- Section Abstract: This method promises to avoid gimbal lock inherent to Euler angles. It produces plausible results on several benchmarks, running quickly enough for interactive animation; and spontaneously exhibits the intermediate-axis effect, precession, and nutation. However, it fails dramatically on 4DISP with 10-msec time step, requiring microsecond granularity to conserve energy and momentum. Such failure is not unexpected, reading the references, but the drama is surprising — despite doing a good job on other problems, it’s not even close for 4DISP. We are forced to find other integrators.

Tiny Quaternionic Dynamics Library (TQDL)

Forced Motion for TQDL & RK4

Demos of TQDL & RK4

Spherical Pendulum

ISSUE: Accumulating Energy and Angular Momentum

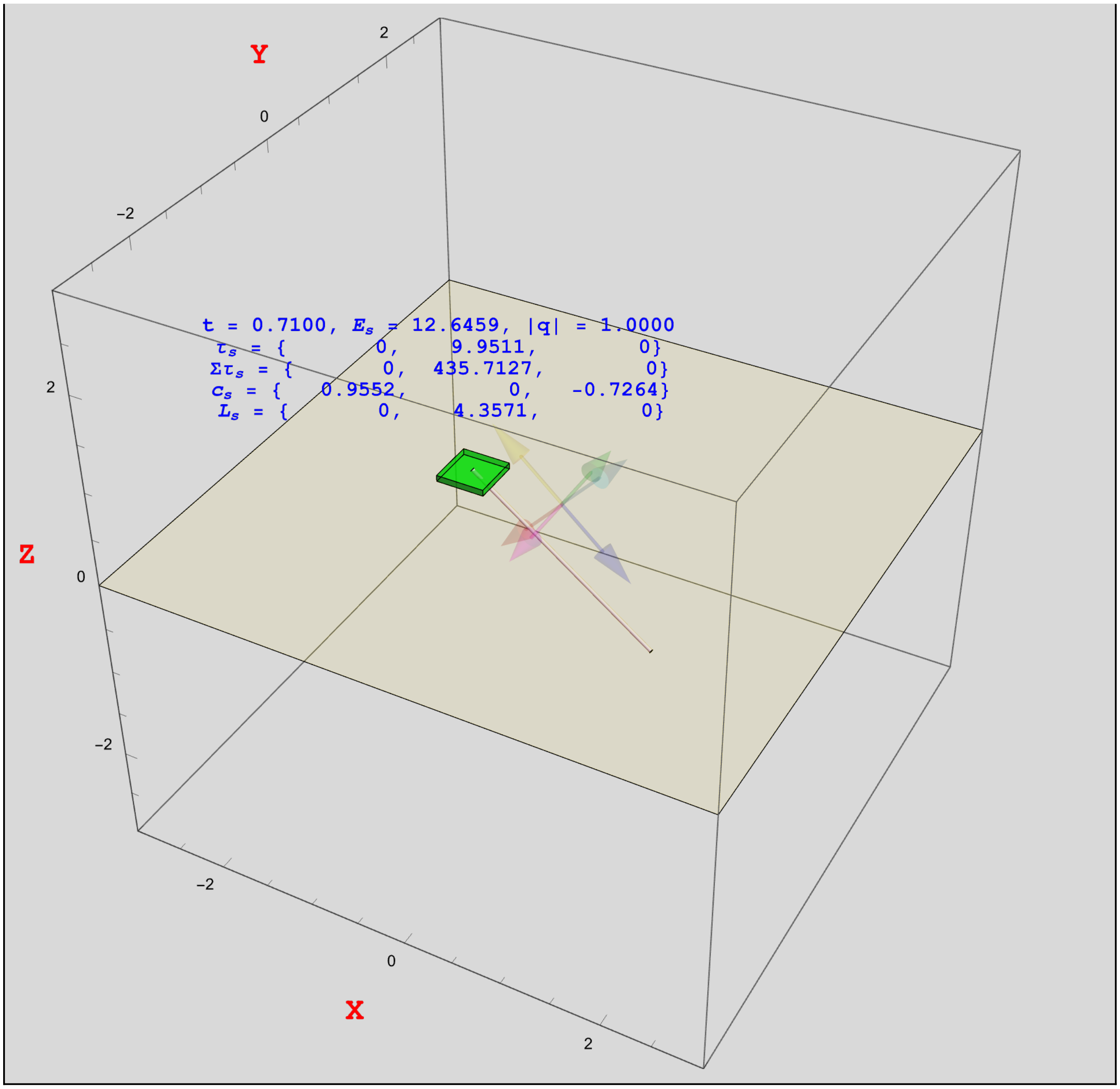
In the body frame, gravitation points upward with a point of application at the bottom of the baton. The vector from the center of gravity to the point of application is $-c_s$, the torque is $(-c_s \times m\,|g|\,e_3) = (m\,|g|\,e_3 \times c_s)$, with × the 3D vector cross product. The gravitational force vector is applied at the same point with the same signs. The integration step size is 10 milliseconds, as before.

This rapidly accumulates kinetic energy and average angular momentum. In fact, the dropped pendulum immediately goes all the way around with a time step of 10 msec, an unphysical result. The time step must be reduced by a factor of 100 to prevent such swinging around. At that time step, the animation is intolerably slow (try it by changing `dt`, if you have 10 minutes or so to devote to watching it). In any event, a real pendulum with a frictionless mount would not accumulate energy or angular momentum.

In[302]:=

```
With[{epsilon = 0.0001, dt = 0.01, h = 1 / 15., w = 1 / 4., vu = 3.},
 With[{ztxt = 3, xtxt = 0, ytxt = -2, znl = 0.2},
  With[{kart = Cuboid[{-w, -w, epsilon}, {w, w, epsilon + h}],
    floor = Polygon[{{-vu, -vu, 0}, {vu, -vu, 0}, {vu, vu, 0}, {-vu, vu, 0}}],
    axisLabelStyle =
     text ↦ Style[text, Red, Bold, 18, FontFamily → "Courier New"],
    Ib = rig["moment of inertia"]⟦1⟧, Ibi = rig["moment of inertia"]⟦2⟧,
    cb = rig["cb"], rq0 = Q[0, 10 °, 0 * -0.5 °]},
   DynamicModule[
    {rq = rq0, ωb = o, ωs = o, Lb = o, Ls = o, pos = o, ps = o, force = o, τs = o, t = 0,
     cs = rv[rq0, cb], θv = θvrq[rq0],
     renderPos = o, rotarg1 = 0, rotarg2 = o, Στs = o},
    Dynamic[
     t += dt;
     τs = ((rig["mass"] Abs[g] e3 + force) × (cs));
     Στs += τs;
     cs = rv[rq, cb];
     renderPos = -cs⟦1⟧ e1 - cs⟦2⟧ e2;
     (* TODO: pos? Risk z drift. *)
     θv = θvrq[rq];
     rotarg1 = θv⟦1⟧;
     rotarg2 = If[o === Chop[θv⟦2⟧], e1, θv⟦2⟧];
     {ωb, rq, ωs, Lb, Ls} =
      oneStepForcedRotationalMotion[ωb, rq, Ib, Ibi, dt, τs];
     Graphics3D[{
       Text[font["t = " <> nfm@t <> ", Es = " <> nfm@(ωsᵀ.Ls / 2 - g cs⟦3⟧) <>
          ", |q| = " <> nfm@Abs[rq]], {xtxt, ytxt, ztxt}],
       Text[font["τs = " <> vfm@τs], {xtxt, ytxt, ztxt - znl}],
       Text[font["Στs = " <> vfm@Στs], {xtxt, ytxt, ztxt - 2 znl}],
       Text[font["cs = " <> vfm@cs], {xtxt, ytxt, ztxt - 3 znl}],
       Text[font["Ls = " <> vfm@Ls], {xtxt, ytxt, ztxt - 4 znl}],
       jack[rq], {Yellow, Opacity[.3 / 4], floor},
       {Green, Opacity[.6], Translate[kart, renderPos]},
       {White, Opacity[.75], Translate[Translate[
          Rotate[rig["graphics primitives"], rotarg1, rotarg2],
          cs], renderPos + (epsilon + h) e3]}},
      ImageSize → Large, Axes → True,
      AxesLabel → axisLabelStyle /@ {"X", "Y", "Z"},
      PlotRange → {{-vu, vu}, {-vu, vu}, {-vu, vu}}]  ]  ]  ]]]
```

Out[302]=



Addressing the Mystery

Stern-Desbrun Symplectic Integrator

- Section Abstract: This integrator produces plausible results for a small time step, but, in general, is too slow and unstable. We leave further investigation to the future, preferring to invest time in

CGDVIE3. In particular, we have not profiled the code and we have not explored its poor energy conservation, which is supposed to be automatic via the discrete Noether's Theorem. In the offing, for ground truth, we produce a good numerical integrator via *Mathematica*'s `NDSolve`. This integrator is not a primary objective of this work, but serves only to check others.

Recast the problem in Lagrangian form. What are good generalized coordinates and velocities of the pendulum? The obvious choice, spherical coordinates, has a singularity at the North pole, right where we want the baton to linger when we get to controlling it. At the North pole, the longitude can be any number. A better choice of coordinates are pitch and an aerobatic, coordinated-cone roll (not axis roll!). These angles are co-latitude δ and an angular displacement η at a clockwise right angle to the great-circle arc of co-latitude. For any non-zero δ, spinning η makes a non-great circle on the surface around a pair of poles on the Equator. The demonstration below makes this visually clear. These poles are not as catastrophic as the North and South poles. *Mathematica*'s integrator produce interpolation functions that fly right through them. Both δ and η are zero at the North pole. η has two singularities on the Equator, when δ is 90° or 270°. But that situation is better than a singularity at the North pole because the baton will fly through these poles at the Equator, not linger at them.

The actual physical system is represented by the red ball at the center of mass of the baton, swinging around the origin. The sliding cart is a visual fiction, useful later when we control the baton. The Lagrangian, here, does not account for translational energy imparted by the cart, only rotational energy. We fix that later.

Demonstration of the Coordinate System

Dynamical Set-Up

Center of mass

z-Height

Potential Energy

We'll need symbolical and numerical variations on the theme of energy. The symbolical variations support *Mathematica*'s magic calculus with parametric functions like δ[t], η[t], δ'[t], and η'[t]. That magic yields the equations of motion almost effortlessly. I learned this from Andrew Moylan's Reference [15]. The numerical forms support graphics and discrete calculations later. For the numerical forms, we simply replace parametric functions with unbound symbols via numRules.

numRules

Velocity Vector

Kinetic Energy

Lagrangian

Leqns

Ground-Truth Solution with Initial Conditions

Discrete Lagrangian

DREPE, DRNEA, and CGDVIE3

Go back to quaternions, and from there to the group SE(3).[4] We’ll integrate directly on the manifold of SE(3) via discrete methods. We’ll bring up *Dzhanybekhov* now, and 4DISP in a later notebook after figuring out the potential-energy term in Equations 16b and 16c below. The paper does not define the symbol T^{k*}. I have contacted the authors to find out what it might be, but not received a satisfactory answer. Perhaps it is some kind of adjoint of the configuration T^k or perhaps it is a pullback of some sort.

There is a lot of machinery, here. It’s mostly about implementing Equations 16b and 16c in Reference [2]. Notice that Equation 16a of Reference [2] is exactly the same as we have above in DEL, but in a 6-dimensional coordinate system on SE(3) instead of in our 2-dimensional Lagrangian coordinate system.

Out[ ]=

By the least action principle with Equation (9), (10), and (14), we can derive the DEL equation for a single rigid body in SE(3), which is the well known *discrete reduced Euler-Poincaré* equations [11,16]:

$$D_2 L_d(T^{k-1}, T^k) + D_1 L_d(T^k, T^{k+1}) = 0 \quad \in \mathbb{R}^6, \tag{16a}$$

where

$$D_2 L_d(T^{k-1}, T^k) = -[\mathrm{Ad}_{\exp(\Delta t[V^{k-1}])}]^T \, [d\log_{\Delta t V^{k-1}}]^T \, G V^{k-1} + \frac{\Delta t}{2} \, T^{k\,*} \, P(T^k) \tag{16b}$$

$$D_1 L_d(T^k, T^{k+1}) = [d\log_{\Delta t V^k}]^T \, G V^k + \frac{\Delta t}{2} \, T^{k\,*} \, P(T^k). \tag{16c}$$

Naming Convention (*Hungarian Types*)

Vectors as Lists, Column Vectors, Row Vectors

Utilities

Append Column

Append Matrix

rotMatFromQ: Rotation Matrix From Quaternion

R: 3-2-1 Object Rotation From ψ, θ, ϕ

yprFromRotMat: ψ, θ, ϕ From Rotation Matrix

qFromRotMat: Quaternion From Rotation Matrix

Proofs and Tests

DREPE, DRNEA, and CGDVIE3 Theory

DRNEA (Reference [2])

SO(3), $\mathfrak{so}(3)$

Definitions (UNDONE)

Example

hatSo3: $\mathbb{R}^3 \to \mathfrak{so}(3)$, unHatR3: $\mathfrak{so}(3) \to \mathbb{R}^3$

expSO3: $\mathfrak{so}(3) \to$ SO(3), AKA Rodrigues' Formula

logSo3: SO(3) $\to \mathfrak{so}(3)$

Remarks

TODO: $\text{Adj}_R \in \mathbb{R}^3 \to \mathbb{R}^3$: $R \in SO(3)$, Adjoint in SO(3)

SE(3)

Definitions

SE3Form

pickSO3: SE(3) → SO(3)

pickR3: SE(3) → $\mathbb{R}^3$

$\mathfrak{se}(3)$

se3Form

hatSe3: $\mathbb{R}^6 \to \mathfrak{se}(3)$, unHatR6: $\mathfrak{se}(3) \to \mathbb{R}^6$, unHat2R3: $\mathfrak{se}(3) \to \mathbb{R}^{2\times 3}$

expSE3: $\mathfrak{se}(3)$ → SE(3)

expSE3symbolic: $\mathfrak{se}(3)$→SE(3)

logSe3: SE(3) → $\mathfrak{se}(3)$, logR6: SE(3) → $\mathbb{R}^6$

ad6R6: $\mathfrak{se}(3) \to \mathfrak{se}(3) \to \mathbb{R}^{6\times 6}$, Lie Bracket

dlog6R6: $\mathfrak{se}(3) \to \mathfrak{se}(3) \to \mathbb{R}^6$, Inverse Right Trivialized Tangent Operator

Adj6R6: Adjoint Operator in SE(3)

Lagrangian

TSE3: Configuration

VR6, VhatSe3: Velocity in $\mathfrak{se}(3)$

PR, PrigR ∈ SE(3)→$\mathbb{R}$: Potential Energy

G6R6: Grig6R6: Inertial Matrix

LR, LrigR ∈ SE(3)→$\mathfrak{se}(3)$→$\mathbb{R}$: Lagrangian

LdRigR ∈ SE(3)→SE(3)→$\mathbb{R}$: Trapezoidal Quadrature

$T^{k^*} P(T^k) = T^{k^*} \partial_T P(T^k)$

From https://math.stackexchange.com/questions/4926505

it seems the term $T^{k^*} P(T^k)$ should actually be something like $T^{k^*} \partial_T P(T^k)$ (see for example equation (11a) in reference [11]), meaning it is the element of $\mathfrak{se}(3)^*$ defined by

$$V \in \mathfrak{se}(3) \mapsto (d_{T^k} P)(T^k V) := \left.\frac{d}{ds}\right|_{s=0} P(T^k \gamma_V(s))$$

where $\gamma_V(s)$ is any curve in SE(3) with $\gamma_V(0) = e, \gamma_V'(0) = V$ (for example, $\gamma_V(s) = \exp_{\mathrm{SE}(3)}(sV)$). In other words, the asterisk implies "pull-back" ($g^* f = f \circ g$). The second confusing thing

In[527]:=

```
PrigR[TSE3[ψ, ϑ, ϕ, x, y, z]]
```

Out[527]=

```
-11.772 Cos[ϑ] Cos[ϕ]
```

In[528]:=

```
{D[#, ψ], D[#, ϑ], D[#, ϕ], D[#, x], D[#, y], D[#, z]} &@
 PrigR[Echo@TSE3[ψ, ϑ, ϕ, x, y, z]]
```

```
» {{Cos[ϑ] Cos[ψ], -Cos[ϑ] Sin[ψ], Sin[ϑ], x},
  {Cos[ψ] Sin[ϑ] Sin[ϕ] + Cos[ϕ] Sin[ψ], Cos[ϕ] Cos[ψ] - Sin[ϑ] Sin[ϕ] Sin[ψ],
   -Cos[ϑ] Sin[ϕ], y}, {-Cos[ϕ] Cos[ψ] Sin[ϑ] + Sin[ϕ] Sin[ψ],
   Cos[ψ] Sin[ϕ] + Cos[ϕ] Sin[ϑ] Sin[ψ], Cos[ϑ] Cos[ϕ], z}, {0, 0, 0, 1}}
```

Out[528]=

```
{0, 11.772 Cos[ϕ] Sin[ϑ], 11.772 Cos[ϑ] Sin[ϕ], 0, 0, 0}
```

In[529]:=

```
{ψ, ϑ, ϕ, x, y, z}.{D[#, ψ], D[#, ϑ], D[#, ϕ], D[#, x], D[#, y], D[#, z]} &@
 PrigR[TSE3[ψ, ϑ, ϕ, x, y, z]]
```

Out[529]=

```
11.772 ϑ Cos[ϕ] Sin[ϑ] + 11.772 ϕ Cos[ϑ] Sin[ϕ]
```

In[530]:=

```
SE3Form[TSE3[ψ, ϑ, ϕ, x, y, z]] /. shorteningRules
```

Out[530]//DisplayForm=

$$\left(\begin{array}{ccc|c} C_\vartheta C_\psi & -C_\vartheta S_\psi & S_\vartheta & x \\ C_\psi S_\vartheta S_\phi + C_\phi S_\psi & C_\phi C_\psi - S_\vartheta S_\phi S_\psi & -C_\vartheta S_\phi & y \\ -C_\phi C_\psi S_\vartheta + S_\phi S_\psi & C_\psi S_\phi + C_\phi S_\vartheta S_\psi & C_\vartheta C_\phi & z \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

In[531]:=

```
With[{se3 = {ω1, ω2, ω3, vx, vy, vz}, print = Identity, qrint = Identity},
 VR6[Sequence @@ se3] // MatrixForm // print;
 VhatSe3[Sequence @@ se3] // se3Form // print;
 expSE3symbolic[s VhatSe3[Sequence @@ se3]] // print;
 D[expSE3symbolic[s VhatSe3[Sequence @@ se3]], {se3}] // print;
 With[{PTs = PrigR[expSE3symbolic[s VhatSe3[Sequence @@ se3]]]},
    Limit[D[PTs, {se3}], s → 0]] // MatrixForm // print;]
```

about the notation is that there is an implicit identification of $\mathfrak{se}(3)^*$ with $\mathfrak{se}(3) \simeq \mathbb{R}^6$, and so you would like to actually interpret the above element of $\mathfrak{se}(3)^*$ as an element of $\mathbb{R}^6$, in other words, to find an element $W \in \mathbb{R}^6$ such that

$$W \cdot V = \left.\frac{d}{ds}\right|_{s=0} P(T^k \gamma_V(s)) \qquad \forall V \in \mathbb{R}^6,$$

where $\cdot$ denotes the usual dot product in $\mathbb{R}^6$. If you have a parametrization of SE(3) already in mind, you could calculate wrt that. If not, probably the most natural parametrization is via the exponential map

$$\exp_{\mathrm{SE}(3)} : \mathfrak{se}(3) \mapsto \mathrm{SE}(3)$$

Then taking $T^k = \exp_{\mathrm{SE}(3)}(U)$, say, it's straightforward (but tedious) to calculate

$$\left.\frac{d}{ds}\right|_{s=0} \exp_{\mathrm{SE}(3)}(U) \exp_{\mathrm{SE}(3)}(sV)$$

and from this

$$\left.\frac{d}{ds}\right|_{s=0} P(\exp_{\mathrm{SE}(3)}(U) \exp_{\mathrm{SE}(3)}(sV)) =$$

$$\sum_{ij} \frac{\partial P}{\partial x_{ij}}(\exp_{\mathrm{SE}(3)}(U)) \left.\frac{d}{ds}\right|_{s=0} \left(\exp_{\mathrm{SE}(3)}(U) \exp_{\mathrm{SE}(3)}(sV)\right)_{ij}$$

The above is actually just

$$d_e[\exp_{\mathrm{SE}(3)}(U)](V),$$

Generalized Forces and Velocities (TODO)

Variational Integrators in SE (3) (TODO)

Here are the discrete, reduced, Euler-Poincaré equations (**DREPE**; equations 16 and 17 of reference [2]). These are transcribed up to uncertainty about the meaning of T^{k*}, undefined in the paper, but possibly an adjoint representation or a member of the co-tangent space of T^k. We took a guess via `Transpose`, which converts vectors to co-vectors, and verified it on the *Dzhanybekhov* benchmark. (**TODO**: this is almost certainly wrong and works only by accident on *Dzhanybekhov*.)

Working on T^{k*}

Taking a guess that T^{k*}, given moment and force covectors $\boldsymbol{F} = \left(\underset{\rightharpoondown}{\mu}, \underset{\rightharpoondown}{f}\right)$, is $\mathrm{Ad}_{\boldsymbol{T}}{}^{\star}(\boldsymbol{F}) = \left(R^{\mathsf{T}}.\underset{\rightharpoondown}{\mu} - \left(R^{\mathsf{T}}.\vec{p}\right) \times \underset{\rightharpoondown}{f}, R^{\mathsf{T}}.\underset{\rightharpoondown}{f}\right)$ in $\mathbb{R}^6$,

$F^k \in \mathfrak{se}^*(3)$ is the integral of the virtual work of external forces over time interval h, loosely represented by a flat list in $\mathbb{R}^6$. Properly, it should be a row vector in $\mathbb{R}^{1\times 6}$ (**TODO**: fix this mess).

Mathematica computes the Hadamard (pointwise) product for adjacent column vectors.

In[532]:=

```
{{a}, {b}} {{c}, {d}}
```

Out[532]=

```
{{a c}, {b d}}
```

The `Flatten` in `dPR` brings the derivative from a proper column vector to a flat list like the rest of the terms.

In[585]:=

```
ClearAll[tkStarR6, tkR6, dPR];

tkStarR6[TkSE3_] := unHatR6[TkSE3];
tkR6[TkSE3_] := unHatR6[TkSE3];
dPR[PR_, TkSE3_] := Flatten@Module[{ψ, θ, ϕ, x, y, z}, ({
          D[#, ψ], D[#, θ], D[#, ϕ],
          D[#, x], D[#, y], D[#, z]} &@PR[TSE3[ψ, θ, ϕ, x, y, z]]) /.
      {Inner[Rule, {ψ, θ, ϕ, x, y, z}, tkR6[TkSE3], List]}];

ClearAll[ΔTSE3];

ΔTSE3[TkmSE3_, TkSE3_] := Inverse[TkmSE3].TkSE3;

ClearAll[D2Ld, D1Ld, DREPE];
With[{print = Null &}, (* change to "Print" or "Echo" if you want to *)
   (* "km" abbreviates "k-1" *)
   D2Ld[TkmSE3_, TkSE3_, G6R6_, PR_, h_ : 0.01] :=
```

```
With[{ΔTkmSE3 = ΔTSE3[TkmSE3, TkSE3]},
 With[{hVkmSe3 = logSe3[ΔTkmSE3]}, (* canonicalize ΔT *)
  With[{exphVkmSE3 = expSE3[hVkmSe3]}, (* back into SE(3) *)
   With[{AdjTexphVkm6R6 = Transpose[Adj6R6[exphVkmSE3]]},
    With[{hVkmR6 = unHatR6[hVkmSe3]},
     With[{dlogThVkm6R6 = Transpose[dlog6R6[hVkmR6]["dlog[V]^6×6"]]},
      With[{term1 = 1/h (-AdjTexphVkm6R6.dlogThVkm6R6.G6R6.hVkmR6)},
       With[{},
        With[{TkStarR6 = tkStarR6[TkSE3]},
         With[{term2 = h/2 TkStarR6 dPR[PR, TkSE3]}, (* Hadamard *)
          With[{result = term1 + term2},
           print[<|
             "TkSE3" → SE3Form[TkSE3],
             "TkmSE3" → SE3Form[TkmSE3],
             "ΔTkmSE3" → SE3Form[ΔTkmSE3],
             "exphVkmSE3" → SE3Form[exphVkmSE3],
             "AdjTexphVkm6R6" → MatrixForm[AdjTexphVkm6R6],
             "hVkmR6" → MatrixForm[hVkmR6],
             "dlogThVkm6R6" → MatrixForm[dlogThVkm6R6],
             "G6R6" → MatrixForm[G6R6],
             "TkStarR6" → MatrixForm[TkStarR6],
             "D2Ld.term1" → MatrixForm[term1],
             "D2Ld.term2" → MatrixForm[term2],
             "D2Ld.result" → result|>];
           result]]]]]]]]]]];
(* "km" abbreviates "k-1" *)
D1Ld[TkSE3_, TkpSE3_, G6R6_, PR_, h_ : 0.01] :=
 With[{ΔTkSE3 = ΔTSE3[TkSE3, TkpSE3]},
  With[{hVkSe3 = logSe3[ΔTkSE3]},
   With[{hVkR6 = unHatR6[hVkSe3]},
    With[{dlogThVk6R6 = Transpose[dlog6R6[hVkR6]["dlog[V]^6×6"]]},
     With[{term1 = dlogThVk6R6.G6R6.hVkR6 / h},
      With[{},
       With[{TkStarR6 = tkStarR6[TkSE3]},
```

```
            With[{term2 = h/2 TkStarR6 dPR[PR, TkSE3]}, (* Hadamard *)
             With[{result = term1 + term2},
              print[<|
                "TkSE3" → SE3Form[TkSE3],
                "TkpSE3" → SE3Form[TkpSE3],
                "ΔTkSE3" → SE3Form[ΔTkSE3],
                "hVkSe3" → se3Form[hVkSe3],
                "hVkR6" → MatrixForm[hVkR6],
                "dlogThVk6R6" → MatrixForm[dlogThVk6R6],
                "G6R6" → MatrixForm[G6R6],
                "TkStarR6" → MatrixForm[TkStarR6],
                "D1Ld.term1" → MatrixForm[term1],
                "D1Ld.term2" → MatrixForm[term2],
                "D1Ld.result" → result|>];
              result]]]]]]]]];
  DREPE[TkmSE3_, TkSE3_, TkpSE3_, G6R6_, PR_, FkR6_, h_ : 0.01] :=
   D2Ld[TkmSE3, TkSE3, G6R6, PR, h] + D1Ld[TkSE3, TkpSE3, G6R6, PR, h] + FkR6;];
```

■ **Unit Tests**

In[580]:=

```
With[{big = 10, o6 = ConstantArray[0, 6]},
 With[{bigRange = {-big, big}},
  With[{tk = TSE3 @@ RandomReal[bigRange, 6],
    tkp = TSE3 @@ RandomReal[bigRange, 6],
    tkm = TSE3 @@ RandomReal[bigRange, 6]},
   <|"D2Ld" → D2Ld[tkm, tk, Grig6R6, PrigR],
    "D1Ld" → D1Ld[tk, tkp, Grig6R6, PrigR],
    "DREPE" → DREPE[tkm, tk, tkp, Grig6R6, PrigR, o6] |>]]]

With[{big = 10, o6 = ConstantArray[0, 6]},
 With[{bigRange = {-big, big}},
  With[{tk = TSE3 @@ RandomReal[bigRange, 6],
    tkp = TSE3 @@ RandomReal[bigRange, 6],
    tkm = TSE3 @@ RandomReal[bigRange, 6]},
   <|"D2Ld" → D2Ld[tkm, tk, Gdzhany6R6, 0 &],
    "D1Ld" → D1Ld[tk, tkp, Gdzhany6R6, 0 &],
    "DREPE" → DREPE[tkm, tk, tkp, Gdzhany6R6, 0 &, o6] |>]]]
```

```
» {1896.72, -1314.4, -5567.21, -2081.88, -292.288, -799.597}
» {0., 0.0223395, 0.0103339, 0., 0., 0.}
» {6694.6, -3286.12, 14104.4, -3418.43, -1486.6, 386.716}
» {0., 0.0223395, 0.0103339, 0., 0., 0.}
» {1896.72, -1314.4, -5567.21, -2081.88, -292.288, -799.597}
» {0., 0.0223395, 0.0103339, 0., 0., 0.}
» {6694.6, -3286.12, 14104.4, -3418.43, -1486.6, 386.716}
» {0., 0.0223395, 0.0103339, 0., 0., 0.}
```

Out[580]=

```
<|D2Ld → {1896.72, -1314.38, -5567.2, -2081.88, -292.288, -799.597},
 D1Ld → {6694.6, -3286.1, 14104.4, -3418.43, -1486.6, 386.716},
 DREPE → {8591.32, -4600.48, 8537.21, -5500.31, -1778.89, -412.881}|>
```

```
» {-586.672, -2330.69, 790.633, -712.171, 658.597, 1653.09}
» {0., 0., 0., 0., 0., 0.}
» {362.752, -5540.62, -3503.94, -922.368, -837.999, 2042.49}
» {0., 0., 0., 0., 0., 0.}
» {-586.672, -2330.69, 790.633, -712.171, 658.597, 1653.09}
» {0., 0., 0., 0., 0., 0.}
» {362.752, -5540.62, -3503.94, -922.368, -837.999, 2042.49}
» {0., 0., 0., 0., 0., 0.}
```

Out[581]=

```
<|D2Ld → {-586.672, -2330.69, 790.633, -712.171, 658.597, 1653.09},
 D1Ld → {362.752, -5540.62, -3503.94, -922.368, -837.999, 2042.49},
 DREPE → {-223.92, -7871.3, -2713.3, -1634.54, -179.403, 3695.58}|>
```

One way to go about finding a root for T^{k+1} is to integrate forward in SO(3) using RK4. Reference [2] does not find roots this way.

In[543]:=

```
ClearAll[showSO3Apparatus];
showSO3Apparatus[t_, ωb_, Lb_, ωs_, Ls_, Ib_, rotMatSO3_, apparatus_] :=
  (Module[{placeZ = 1, y, p, r},
    With[{arrowDiameter = 0.02, displaceZ = -0.15},
     {y, p, r} = yprFromRotMat[rotMatSO3];
     Show[{
       Graphics3D[{
         Text[myFont[Blue][
           "t = " <> ToString[NumberForm[t, {10, 2}]]], {-1, -1, placeZ}],
```

```
placeZ += displaceZ;
Text[myFont[Blue][
  "yaw = " <> ToString[NumberForm[y / °, {10, 4}]] <> "°"],
 {-.999, -1, placeZ}],

placeZ += displaceZ;
Text[myFont[Blue][
  "pitch = " <> ToString[NumberForm[p / °, {10, 4}]] <> "°"],
 {-.999, -1, placeZ}],

placeZ += displaceZ;
Text[myFont[Blue][
  "roll = " <> ToString[NumberForm[r / °, {10, 4}]] <> "°"],
 {-.999, -1, placeZ}],

placeZ += displaceZ;
Text[myFont[Blue][
  "|Ls| = " <> ToString[NumberForm[Sqrt[Ls.Ls], {10, 4}]]]],
 {-.975, -1, placeZ}],

placeZ += displaceZ;
Text[myFont[Blue][
  "|Lb| = " <> ToString[NumberForm[Sqrt[Lb.Lb], {10, 4}]]]],
 {-.975, -1, placeZ}],

placeZ += displaceZ;
Text[myFont[Blue][
  "ωbᵀIbωb/2 = " <> ToString[NumberForm[Sqrt[1/2 ωb.Ib.ωb], {10, 4}]]]],
 {-.95, -1, placeZ}],

placeZ += displaceZ;
Text[myFont[Blue][
  "ωbᵀLb/2 = " <> ToString[NumberForm[Sqrt[1/2 ωb.Lb], {10, 4}]]]],
 {-.95, -1, placeZ}],

placeZ += displaceZ;
Text[myFont[Blue][
  "ωsᵀLs/2 = " <> ToString[NumberForm[Sqrt[1/2 ωs.Ls], {10, 4}]]]],
```

```
            {-.95, -1, placeZ}],

          Magenta, Arrow[Tube[{o, Ls}, arrowDiameter]],
          Text[myFont[Black, 12]["Inertial-Frame Ang Mom"],
            {-1.9, +0.2, 1}, Background → Magenta],

          Red, Arrow[Tube[{o, Lb}, arrowDiameter]],
          Text[myFont[Black, 12]["Body-Frame Ang Mom"],
            {-1.7, -0.1, 1}, Background → Red],

          Cyan, Arrow[Tube[{o, ωs / 10}, arrowDiameter * 1.10]],
          Text[myFont[Black, 12]["Inertial-Frame Ang Vel"],
            {-1.4, -0.4, 1}, Background → Cyan],

          Blue, Arrow[Tube[{o, ωb / 10}, arrowDiameter * 1.10]],
          Text[myFont[White, 12]["Body-Frame Ang Vel"],
            {-1.3, -0.7, 1}, Background → Blue],

          White, GeometricTransformation[
            apparatus["graphics primitives"],
            rotMatSO3]}
        ]},
      Axes → True,
      PlotRange → ConstantArray[plotRg, 3],
      AxesLabel → myFont[Red] /@ {"X", "Y", "Z"},
      ImageSize → Large]]]);

ClearAll[initialConditionsSO3ForcedRotationalMotion];
initialConditionsSO3ForcedRotationalMotion[ωb_, rotMatSO3_, Ib_, Ibi_] :=
  With[{rq = qFromRotMat[rotMatSO3], Lb = Ib.ωb},
   {ωb,
    rotMatSO3,
    (* ωs *)rv[rq, ωb],
    Lb,
    (* Ls *)rv[rq, Lb]}];

ClearAll[oneStepSO3ForcedRotationalMotion];
oneStepSO3ForcedRotationalMotion[ωb_, rotMatSO3_, Ib_, Ibi_, h_, τs_] :=
  With[{rq = qFromRotMat[rotMatSO3]},
   With[{
     ωbNew = ωbn[ωb, Ib, Ibi, h, τs, rq, rk4],
     rqNew = rqb2sn[rq, ωb, h, rk4]},
```

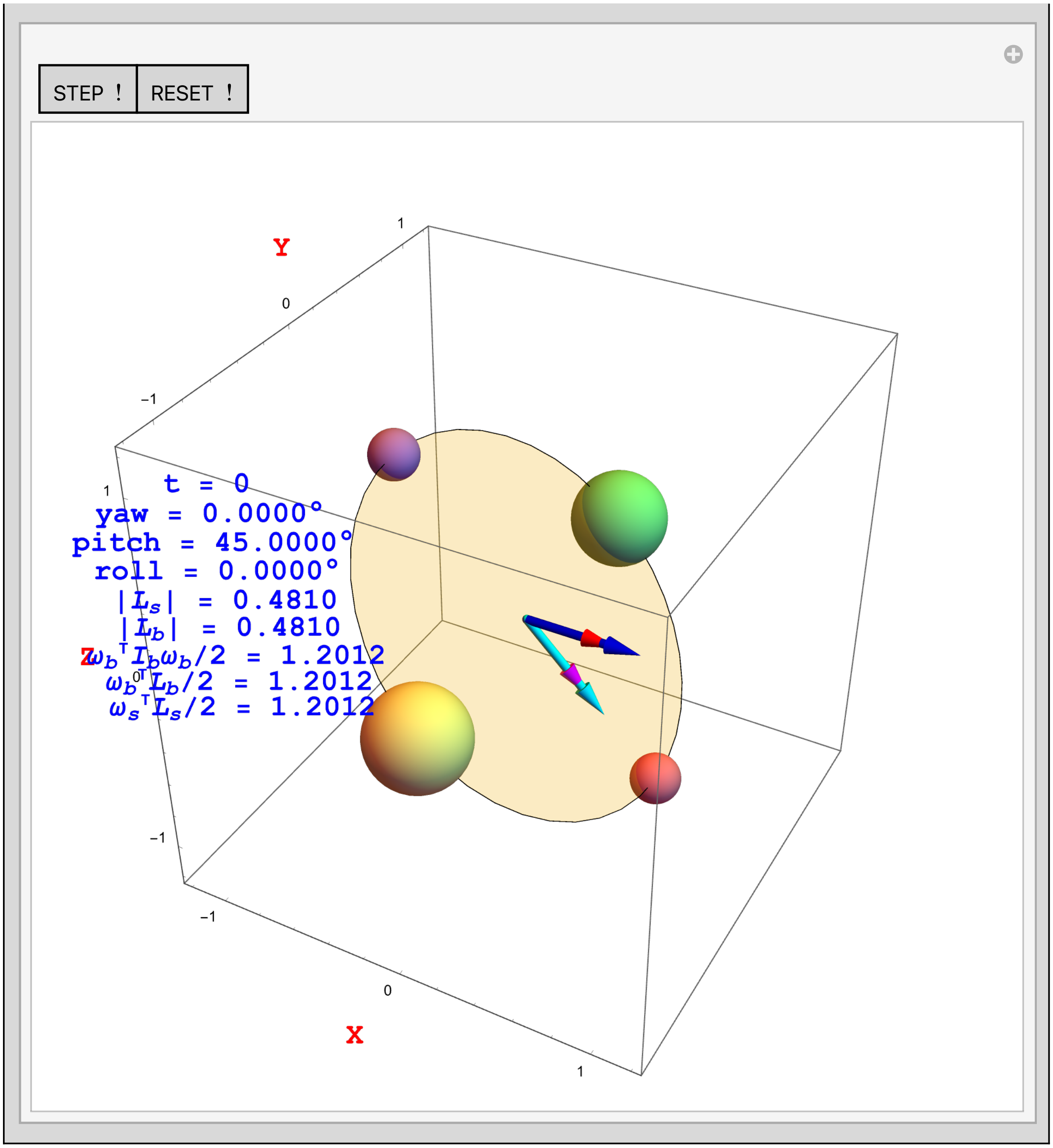
```
    With[{LbNew = Ib.ωbNew},
     {ωbNew,
      rotMatFromQ[rqNew],
      (* ωs *)rv[rqNew, ωbNew],
      LbNew,
      (* Ls *)rv[rqNew, LbNew]}]]];

ClearAll[runSimSO3ForcedRotationalMotion];
runSimSO3ForcedRotationalMotion[
   apparatus_,
   ωbIn_ : {6., .01, 0},
   rqIn_ : rq[π / 4.0, {0, 1., 0}],
   fs_ : {0, 0, 0},
   τs_ : {0, 0, 0}] :=
  With[{ibibi = apparatus["moment of inertia"]},
   With[{Ib = ibibi⟦1⟧, Ibi = ibibi⟦2⟧},
    DynamicModule[
     {t = 0, h = 0.01, ωb = ωbIn, ωs, Lb, Ls, rotMat = rotMatFromQ[rqIn]},
     Dynamic[t += dt;
      {ωb, rotMat, ωs, Lb, Ls} =
       (* calls rk4 *)
       oneStepSO3ForcedRotationalMotion[ωb, rotMat, Ib, Ibi, h, τs];
      showSO3Apparatus[t, ωb, Lb, ωs, Ls, Ib, rotMat, apparatus]]]]];
```

Demonstration of step-by-step RK4 integration on SO(3).

In[551]:=

```
With[{apparatus = dzhanybekhov, h = 0.01,
  ωbIn = {6., .01, 0.0}, rqIn = rq[π / 4.0, {0, 1, 0}], τs = {0, 0, 0}},
 With[{ibibi = apparatus["moment of inertia"]},
  With[{Ib = ibibi⟦1⟧, Ibi = ibibi⟦2⟧},
   DynamicModule[{ωb = ωbIn, rotMatSO3 = rotMatFromQ[rqIn], ωs, Lb, Ls, t = 0},
    With[{
      init = Function[(* of no arguments *)
        {ωb, rotMatSO3, ωs, Lb, Ls} =
         initialConditionsSO3ForcedRotationalMotion[
          ωbIn, rotMatFromQ[rqIn], Ib, Ibi];
        t = 0;
        showSO3Apparatus[t, ωb, Lb, ωs, Ls, Ib, rotMatSO3, apparatus]],
      step = Function[(* of no arguments *)
        {ωb, rotMatSO3, ωs, Lb, Ls} =
         (* calls rk4 *)
         oneStepSO3ForcedRotationalMotion[ωb, rotMatSO3, Ib, Ibi, h, τs];
        t += h;
        showSO3Apparatus[t, ωb, Lb, ωs, Ls, Ib, rotMatSO3, apparatus]]},
     DynamicModule[{dpy = init[]},
      Manipulate[dpy,
       Row[{Button[" STEP ! ", dpy = step[]],
         Button[" RESET ! ", dpy = init[]]}]]]]]]]]
```

Out[551]=



Root Finding

To step from T^{k-1} and T^k to T^{k+1}, find the root T^{k+1} of $D_1 L_d(T^k, T^{k+1}) + D_2 L_d(T^{k-1}, T^k) + F^k = 0$, where $F^k \in \mathfrak{se}^*(3)$ is the integral of the virtual work of the external force over time interval h.

Start with a certain orientation and angular velocity:

In[552]:=

```
o3 = {0, 0, 0};
```

In[553]:=

```
(T0$ = TSE3[initQuat$, o3]) // SE3Form
```

Out[553]//DisplayForm=

$$\left(\begin{array}{ccc|c} 0.707107 & 0. & 0.707107 & 0 \\ 0. & 1. & 0. & 0 \\ -0.707107 & 0. & 0.707107 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

In[554]:=

```
ClearAll[stepDzhanyRK4];
With[{apparatus = dzhanybekhov},
  With[{ibibi = apparatus["moment of inertia"]},
   With[{Ib = ibibi〚1〛, Ibi = ibibi〚2〛},
    stepDzhanyRK4[Tk_, ωbIn_, h_, τs_ : o3] :=
     Module[{ωb = ωbIn, rotMatSO3 = pickSO3[Tk], ωs, Lb, Ls},
      {ωb, rotMatSO3, ωs, Lb, Ls} =
       oneStepSO3ForcedRotationalMotion[ωb, rotMatSO3, Ib, Ibi, h, τs];
      <|"TSE3" → TSE3[rotMatSO3, o3], "ωb" → ωb, "Lb" → Lb, "ωs" → ωs, "Ls" → Ls|>
     ]]]];
```

In[556]:=

```
ClearAll[adHocStep];
adHocStep[T_, w_, h_] :=
  (step$ = stepDzhanyRK4[T, w, h];
   wb$ = step$["ωb"];
   Tkp$ = step$["TSE3"]);
adHocStep[T0$, initωb$, 0.01];
(T1$ = Tkp$) // SE3Form
```

Out[559]//DisplayForm=

$$\left(\begin{array}{ccc|c} 0.707109 & 0.0423303 & 0.705836 & 0 \\ 0.00009994 & 0.998201 & -0.059964 & 0 \\ -0.707105 & 0.0424716 & 0.705832 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

In[560]:=

```
adHocStep[T1$, wb$, 0.01];
(T2$ = Tkp$) // SE3Form
```

Out[561]//DisplayForm=

$$\left(\begin{array}{ccc|c} 0.707119 & 0.084508 & 0.702026 & 0 \\ 0.000199122 & 0.992809 & -0.119712 & 0 \\ -0.707094 & 0.0847906 & 0.702017 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

DREPE should have magnitude close to zero.

In[562]:=

```
DREPE[T0$, T1$, T2$, Gdzhany6R6, 0 &, 0, 0.01] // Norm
```

Out[562]=

```
7.99789 × 10^-9
```

`T2$` is very nearly a root of DREPE. Is it close enough? Hard to say, yet, but we study the numerics below and we will refine the root with ***bandit search*** when needed (https://en.wikipedia.org/wiki/Multi-armed_bandit).

Adapt the step-by-step demonstration above to DREPE. The first step produces a macroscopic value of DREPE, that is, a failure to find a root. This failure is due only to the fact that DREPE requires two configuration estimates to bootstrap — we've seen this before in DELE. From the second step onward, the roots proposed by RK4 produce only microscopic values of DREPE, showing that the proposed roots are good for *Dzhanybekhov*. **We suspect that they will not be good for 4DISP.**

In[563]:=

```
ClearAll[showSO3ApparatusWithDREPE];
showSO3ApparatusWithDREPE[
   t_, ωb_, Lb_, ωs_, Ls_, Ib_, rotMatSO3_, DREPENorm_, apparatus_] :=
  (Module[{placeZ = 1, y, p, r},
    With[{arrowDiameter = 0.02, displaceZ = -0.15},
     {y, p, r} = yprFromRotMat[rotMatSO3];
     Show[{
       Graphics3D[{
         Text[myFont[Blue][
           "t = " <> ToString[NumberForm[t, {10, 2}]]], {-1, -1, placeZ}],

         placeZ += displaceZ;
         Text[myFont[Blue][
           "|\!\(\*StyleBox[\"DREPE\",Background->RGBColor[1,
              1, 0]]\)| = " <>
            ToString[NumberForm[DREPENorm, {10, 6}, NumberFormat →
               (Row[{#1, "e", If[#3 === "", "0 ", #3]}] &)]]],
          {-.999, -1, placeZ}],

         placeZ += displaceZ;
         Text[myFont[Blue][
           "yaw = " <> ToString[NumberForm[y / °, {10, 4}]] <> "°"],
          {-.999, -1, placeZ}],
```

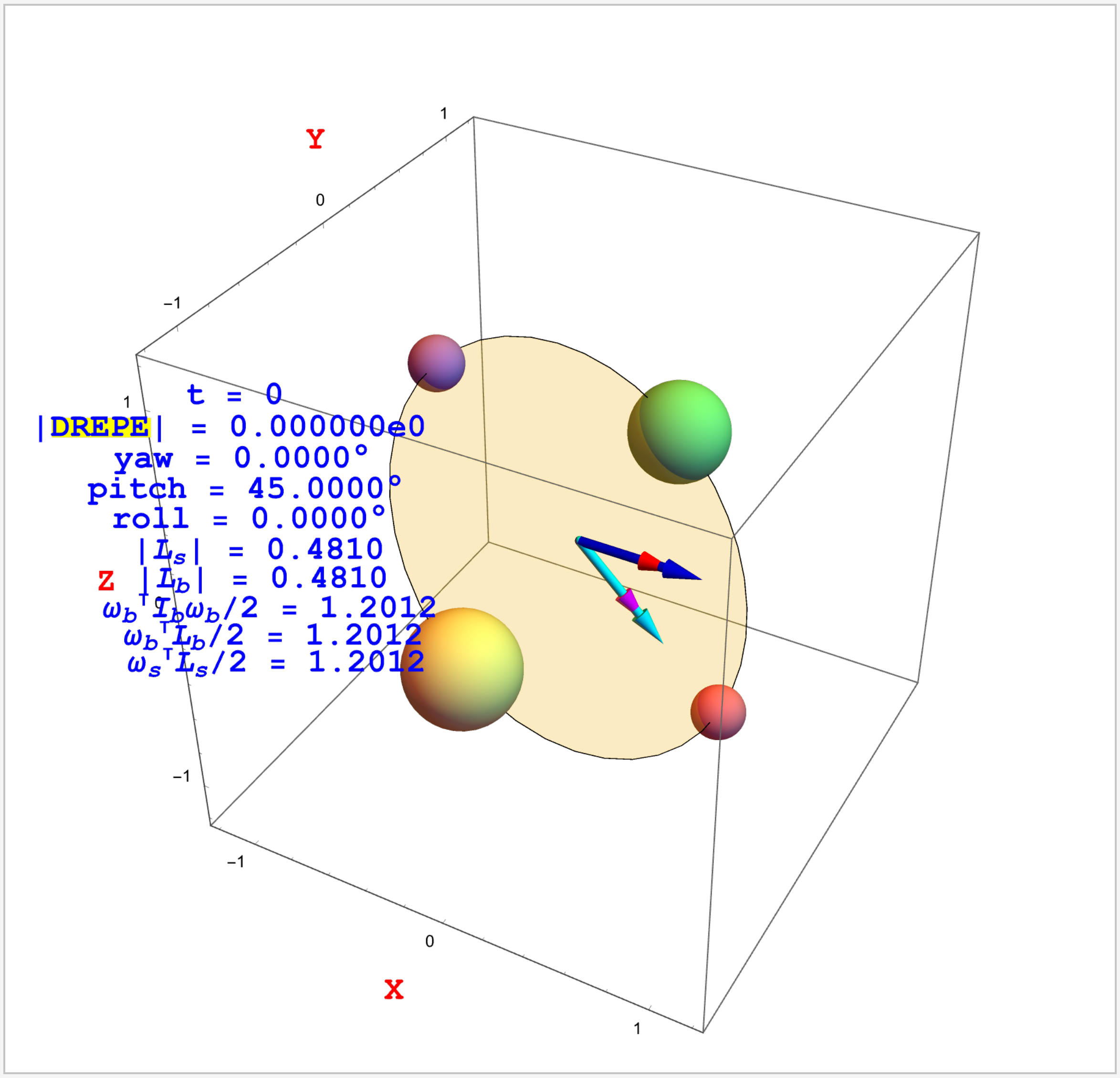
```
        placeZ += displaceZ;
        Text[myFont[Blue][
          "pitch = " <> ToString[NumberForm[p / °, {10, 4}]] <> "°"],
         {-.999, -1, placeZ}],

        placeZ += displaceZ;
        Text[myFont[Blue][
          "roll = " <> ToString[NumberForm[r / °, {10, 4}]] <> "°"],
         {-.999, -1, placeZ}],

        placeZ += displaceZ;
        Text[myFont[Blue][
          "|Ls| = " <> ToString[NumberForm[Sqrt[Ls.Ls], {10, 4}]]],
         {-.975, -1, placeZ}],

        placeZ += displaceZ;
        Text[myFont[Blue][
          "|Lb| = " <> ToString[NumberForm[Sqrt[Lb.Lb], {10, 4}]]],
         {-.975, -1, placeZ}],

        placeZ += displaceZ;
        Text[myFont[Blue][
          "ωbᵀIbωb/2 = " <> ToString[NumberForm[Sqrt[1/2 ωb.Ib.ωb], {10, 4}]]],
         {-.95, -1, placeZ}],

        placeZ += displaceZ;
        Text[myFont[Blue][
          "ωbᵀLb/2 = " <> ToString[NumberForm[Sqrt[1/2 ωb.Lb], {10, 4}]]],
         {-.95, -1, placeZ}],

        placeZ += displaceZ;
        Text[myFont[Blue][
          "ωsᵀLs/2 = " <> ToString[NumberForm[Sqrt[1/2 ωs.Ls], {10, 4}]]],
         {-.95, -1, placeZ}],

        Magenta, Arrow[Tube[{o, Ls}, arrowDiameter]],
        Text[myFont[Black, 12]["Inertial-Frame Ang Mom"],
         {-1.9, +0.2, 1}, Background → Magenta],
```

```
          Red, Arrow[Tube[{o, Lb}, arrowDiameter]],
          Text[myFont[Black, 12]["Body-Frame Ang Mom"],
           {-1.7, -0.1, 1}, Background → Red],

          Cyan, Arrow[Tube[{o, ωs / 10}, arrowDiameter * 1.10]],
          Text[myFont[Black, 12]["Inertial-Frame Ang Vel"],
           {-1.4, -0.4, 1}, Background → Cyan],

          Blue, Arrow[Tube[{o, ωb / 10}, arrowDiameter * 1.10]],
          Text[myFont[White, 12]["Body-Frame Ang Vel"],
           {-1.3, -0.7, 1}, Background → Blue],

          White, GeometricTransformation[
           apparatus["graphics primitives"],
           rotMatSO3]}
        ]},
       Axes → True,
       PlotRange → ConstantArray[plotRg, 3],
       AxesLabel → myFont[Red] /@ {"X", "Y", "Z"},
       ImageSize → Large]]]);

With[{apparatus = dzhanybekhov, h = 0.01,
  ωbIn = {6., .01, 0.0}, rqIn = rq[π / 4.0, {0, 1, 0}], τs = {0, 0, 0}},
 With[{ibibi = apparatus["moment of inertia"]},
  With[{Ib = ibibi[[1]], Ibi = ibibi[[2]]},
   DynamicModule[{ωb = ωbIn, rotMatSO3 = rotMatFromQ[rqIn],
     ωs, Lb, Ls, t = 0, Tkm, Tk, Tkp, DREPENorm},
    With[{
      init = Function[(* of no arguments *)
        Tkm = Tk = TSE3[rqIn, o3];
        {ωb, rotMatSO3, ωs, Lb, Ls} =
         initialConditionsSO3ForcedRotationalMotion[
          ωbIn, rotMatFromQ[rqIn], Ib, Ibi]; t = 0;
        Tkp = TSE3[rotMatSO3, o3];
        DREPENorm = Norm[DREPE[Tkm, Tk, Tkp, Gdzhany6R6, 0 &, 0, h]];
        showSO3ApparatusWithDREPE[
         t, ωb, Lb, ωs, Ls, Ib, rotMatSO3, DREPENorm, apparatus]],
      step = Function[(* of no arguments *)
        Tkm = Tk; Tk = Tkp;
        {ωb, rotMatSO3, ωs, Lb, Ls} =
         oneStepSO3ForcedRotationalMotion[ωb, rotMatSO3, Ib, Ibi, h, τs];
        Tkp = TSE3[rotMatSO3, o3]; (* still a quat in the sim *)
```

```
        DREPENorm = Norm[DREPE[Tkm, Tk, Tkp, Gdzhany6R6, 0 &, 0, h]];
        t += h;
        showSO3ApparatusWithDREPE[
         t, ωb, Lb, ωs, Ls, Ib, rotMatSO3, DREPENorm, apparatus]]},
     DynamicModule[{dpy = init[]},
      Manipulate[dpy,
       Row[{Button[" STEP ! ", dpy = step[]],
         Button[" RESET ! ", dpy = init[]]}]]]]]]]
```

Out[565]=

STEP ! | RESET !



In[593]:=

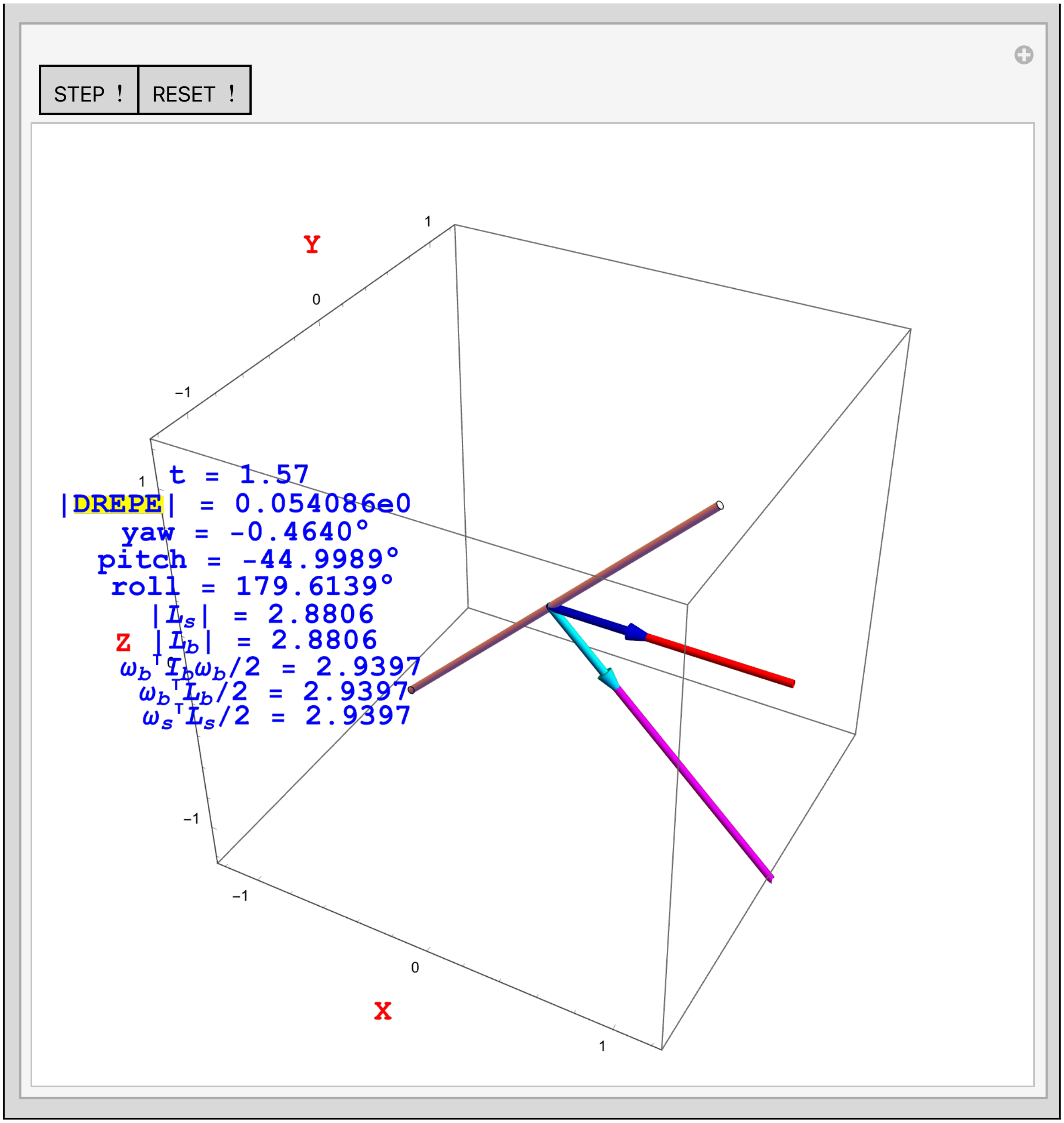
```
ClearAll[showSO3ApparatusWithDREPE];
```

```
showSO3ApparatusWithDREPE[
   t_, ωb_, Lb_, ωs_, Ls_, Ib_, rotMatSO3_, DREPENorm_, apparatus_] :=
  (Module[{placeZ = 1, y, p, r},
    With[{arrowDiameter = 0.02, displaceZ = -0.15},
     {y, p, r} = yprFromRotMat[rotMatSO3];
     Show[{
       Graphics3D[{
         Text[myFont[Blue][
           "t = " <> ToString[NumberForm[t, {10, 2}]]], {-1, -1, placeZ}],

         placeZ += displaceZ;
         Text[myFont[Blue][
           "|\!\(\*StyleBox[\"DREPE\",Background->RGBColor[1,
              1, 0]]\)| = " <>
            ToString[NumberForm[DREPENorm, {10, 6}, NumberFormat →
               (Row[{#1, "e", If[#3 === "", "0 ", #3]}] &)]]],
          {-.999, -1, placeZ}],

         placeZ += displaceZ;
         Text[myFont[Blue][
           "yaw = " <> ToString[NumberForm[y / °, {10, 4}]] <> "°"],
          {-.999, -1, placeZ}],

         placeZ += displaceZ;
         Text[myFont[Blue][
           "pitch = " <> ToString[NumberForm[p / °, {10, 4}]] <> "°"],
          {-.999, -1, placeZ}],

         placeZ += displaceZ;
         Text[myFont[Blue][
           "roll = " <> ToString[NumberForm[r / °, {10, 4}]] <> "°"],
          {-.999, -1, placeZ}],

         placeZ += displaceZ;
         Text[myFont[Blue][
           "|Ls| = " <> ToString[NumberForm[Sqrt[Ls.Ls], {10, 4}]]],
          {-.975, -1, placeZ}],

         placeZ += displaceZ;
         Text[myFont[Blue][
           "|Lb| = " <> ToString[NumberForm[Sqrt[Lb.Lb], {10, 4}]]],
          {-.975, -1, placeZ}],
```

```
    placeZ += displaceZ;
    Text[myFont[Blue][
      "ωb^T Ib ωb/2 = " <> ToString[NumberForm[Sqrt[1/2 ωb.Ib.ωb], {10, 4}]]],
     {-.95, -1, placeZ}],

    placeZ += displaceZ;
    Text[myFont[Blue][
      "ωb^T Lb/2 = " <> ToString[NumberForm[Sqrt[1/2 ωb.Lb], {10, 4}]]],
     {-.95, -1, placeZ}],

    placeZ += displaceZ;
    Text[myFont[Blue][
      "ωs^T Ls/2 = " <> ToString[NumberForm[Sqrt[1/2 ωs.Ls], {10, 4}]]],
     {-.95, -1, placeZ}],

    Magenta, Arrow[Tube[{o, Ls}, arrowDiameter]],
    Text[myFont[Black, 12]["Inertial-Frame Ang Mom"],
     {-1.9, +0.2, 1}, Background → Magenta],

    Red, Arrow[Tube[{o, Lb}, arrowDiameter]],
    Text[myFont[Black, 12]["Body-Frame Ang Mom"],
     {-1.7, -0.1, 1}, Background → Red],

    Cyan, Arrow[Tube[{o, ωs / 10}, arrowDiameter * 1.10]],
    Text[myFont[Black, 12]["Inertial-Frame Ang Vel"],
     {-1.4, -0.4, 1}, Background → Cyan],

    Blue, Arrow[Tube[{o, ωb / 10}, arrowDiameter * 1.10]],
    Text[myFont[White, 12]["Body-Frame Ang Vel"],
     {-1.3, -0.7, 1}, Background → Blue],

    White, GeometricTransformation[
     apparatus["graphics primitives"],
     rotMatSO3]}
  ]},
 Axes → True,
 PlotRange → ConstantArray[plotRg, 3],
```

```
      AxesLabel → myFont[Red] /@ {"X", "Y", "Z"},
      ImageSize → Large]]]);

With[{apparatus = rig, h = 0.01,
  ωbIn = {6., .01, 0.0}, rqIn = rq[π / 4.0, {0, 1, 0}], τs = {0, 0, 0}},
 With[{ibibi = apparatus["moment of inertia"]},
  With[{Ib = ibibi⟦1⟧, Ibi = ibibi⟦2⟧},
   DynamicModule[{ωb = ωbIn, rotMatSO3 = rotMatFromQ[rqIn],
     ωs, Lb, Ls, t = 0, Tkm, Tk, Tkp, DREPENorm},
    With[{
      init = Function[(* of no arguments *)
        Tkm = Tk = TSE3[rqIn, o3];
        {ωb, rotMatSO3, ωs, Lb, Ls} =
         initialConditionsSO3ForcedRotationalMotion[
          ωbIn, rotMatFromQ[rqIn], Ib, Ibi]; t = 0;
        Tkp = TSE3[rotMatSO3, o3];
        DREPENorm = Norm[DREPE[Tkm, Tk, Tkp, Grig6R6, PrigR, 0, h]];
        showSO3ApparatusWithDREPE[
         t, ωb, Lb, ωs, Ls, Ib, rotMatSO3, DREPENorm, apparatus]],
      step = Function[(* of no arguments *)
        Tkm = Tk; Tk = Tkp;
        {ωb, rotMatSO3, ωs, Lb, Ls} =
         oneStepSO3ForcedRotationalMotion[ωb, rotMatSO3, Ib, Ibi, h, τs];
        Tkp = TSE3[rotMatSO3, o3]; (* still a quat in the sim *)
        DREPENorm = Norm[DREPE[Tkm, Tk, Tkp, Grig6R6, PrigR, 0, h]];
        t += h;
        showSO3ApparatusWithDREPE[
         t, ωb, Lb, ωs, Ls, Ib, rotMatSO3, DREPENorm, apparatus]]},
     DynamicModule[{dpy = init[]},
      Manipulate[dpy,
       Row[{Button[" STEP ! ", dpy = step[]],
         Button[" RESET ! ", dpy = init[]]}]]]]]]]]
```

Out[595]=



Adapt the continuous demonstration of Dzhanybekhov to CGDVIE3 and DREPE. This only tracks the microscopic values of DREPE; it does not update the roots, proposed by RK4. **Note the time increment, $h = 0.03$ is three times larger (better) than we had with RK4.** 0.03 is much too large to produce decent roots of DREPE for 4DISP; even $h = 0.01$ is too large for 4DISP.

In[596]:=

```
ClearAll[runSimForcedRotationalMotionWithDREPE];
runSimForcedRotationalMotionWithDREPE[
   apparatus_,
   h_,
   ωbIn_ : {6., .01, 0},
   rqIn_ : rq[π / 4.0, {0, 1., 0}],
   fs_ : {0, 0, 0},
   τs_ : {0, 0, 0}] :=
  With[{ibibi = apparatus["moment of inertia"]},
   With[{Ib = ibibi⟦1⟧, Ibi = ibibi⟦2⟧},
    DynamicModule[
     {t = 0, ωb = ωbIn, ωs, Lb, Ls,
      rotMatSO3 = rotMatFromQ[rqIn], Tkm, Tk, Tkp, DREPENorm},
     With[{
        init = Function[(* of no arguments *)
          Tkm = Tk = TSE3[rqIn, o3];
          {ωb, rotMatSO3, ωs, Lb, Ls} =
           initialConditionsSO3ForcedRotationalMotion[
            ωbIn, rotMatFromQ[rqIn], Ib, Ibi]; t = 0;
          Tkp = TSE3[rotMatSO3, o3];
          DREPENorm = Norm[DREPE[Tkm, Tk, Tkp, Gdzhany6R6, 0 &, 0, h]];
          showSO3ApparatusWithDREPE[
           t, ωb, Lb, ωs, Ls, Ib, rotMatSO3, DREPENorm, apparatus]],
        step = Function[(* of no arguments *)
          Tkm = Tk; Tk = Tkp;
          {ωb, rotMatSO3, ωs, Lb, Ls} =
           oneStepSO3ForcedRotationalMotion[ωb, rotMatSO3, Ib, Ibi, h, τs];
          Tkp = TSE3[rotMatSO3, o3];
          (* still a quat in the sim *)
          DREPENorm = Norm[DREPE[Tkm, Tk, Tkp, Gdzhany6R6, 0 &, 0, h]];
          t += h;
          showSO3ApparatusWithDREPE[
           t, ωb, Lb, ωs, Ls, Ib, rotMatSO3, DREPENorm, apparatus]]},
      init[];
      Dynamic[t += h;
       step[]]]]]];
```

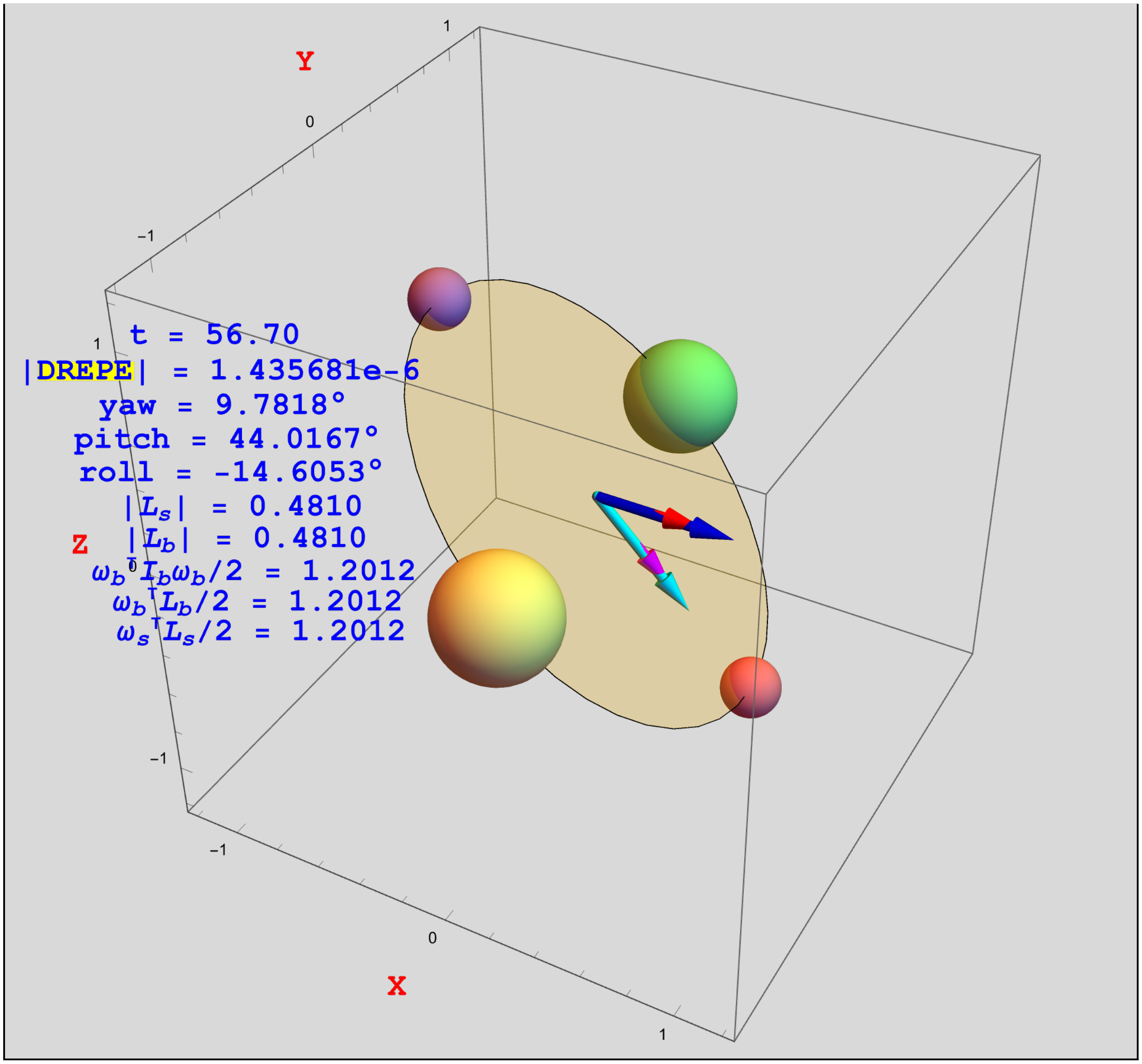
This also loses energy and angular momentum, but much more slowly than does TQDL & RK4 and at a time step 3 times more coarse — 0.3 instead of the .01 we needed with TDQL and RK4. **This is a significant improvement.** Note that the quantity |DREPE| is ideally zero. We see that it fluctuates around some small values, suggesting overall good performance. Also note that there is apparently no memory leak: I've allowed this to run for days at a time. In two days, the computed energy reduced from

1.2012 to 1.1940, a reduction of 0.6 percent.

In[598]:=

```
runSimForcedRotationalMotionWithDREPE[dzhanybekhov, 0.03]
```

Out[598]=

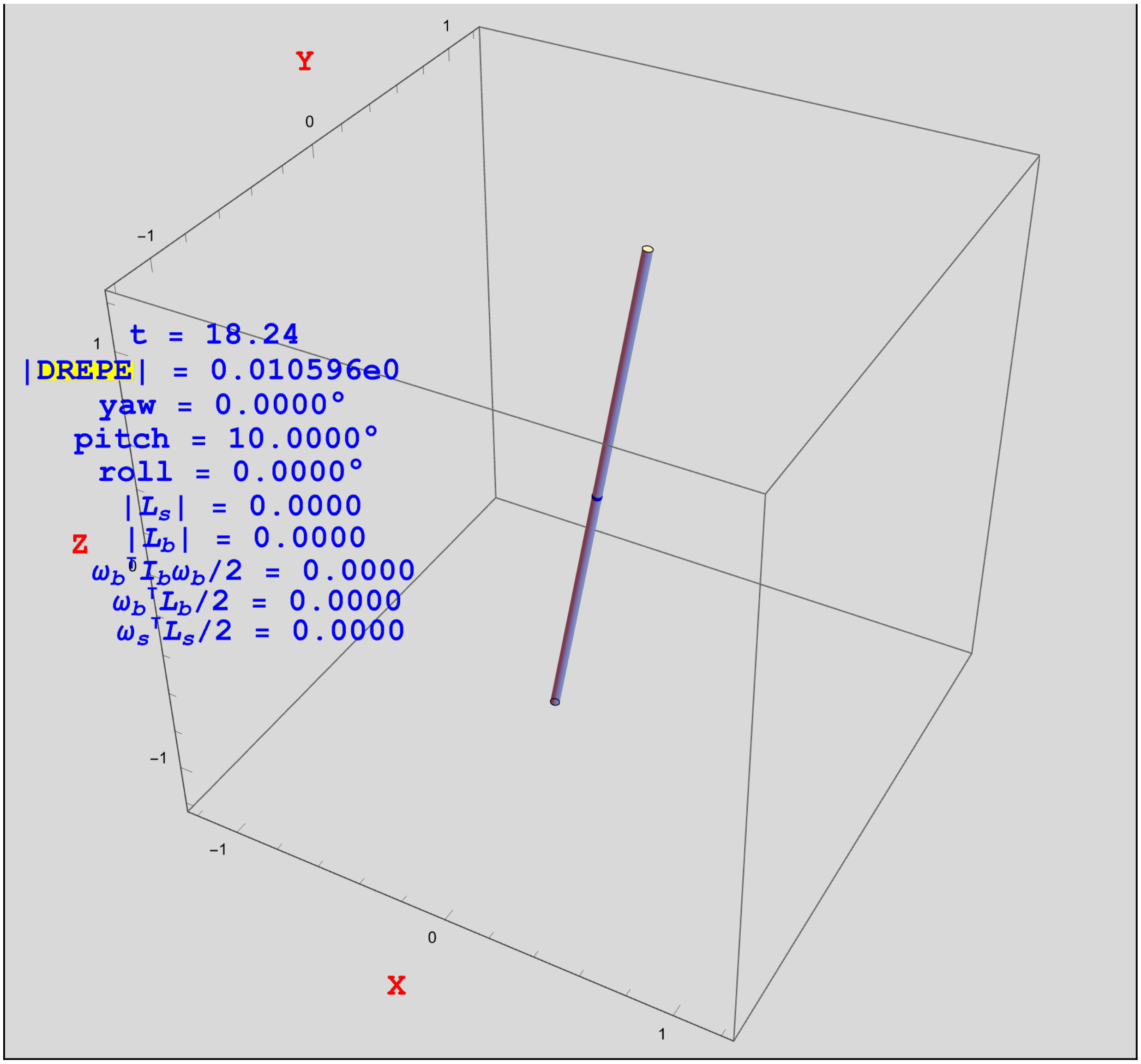


In[599]:=

```
ClearAll[runSimForcedRotationalMotionWithDREPErig];
runSimForcedRotationalMotionWithDREPErig[
   apparatus_,
   h_,
   ωbIn_ : {6., .01, 0},
   rqIn_ : rq[π / 4.0, {0, 1., 0}],
   fs_ : {0, 0, 0},
   τs_ : {0, 0, 0}] :=
```

```
  With[{ibibi = apparatus["moment of inertia"]},
   With[{Ib = ibibi⟦1⟧, Ibi = ibibi⟦2⟧},
    DynamicModule[
     {t = 0, ωb = ωbIn, ωs, Lb, Ls, rotMatSO3 = rotMatFromQ[rqIn],
      Tkm, Tk, Tkp, DREPENorm, qk, csk, τsk, Fk},
     With[{
       init = Function[(* of no arguments *)
         Tkm = Tk = TSE3[rqIn, o3];
         {ωb, rotMatSO3, ωs, Lb, Ls} =
          initialConditionsSO3ForcedRotationalMotion[
           ωbIn, rotMatFromQ[rqIn], Ib, Ibi]; t = 0;
         Tkp = TSE3[rotMatSO3, o3];
         qk = qFromRotMat[rotMatSO3];
         csk = rv[qk, rig["cb"]];
         τsk = (rig["mass"] Abs[g] e3) × csk;
         Fk = {0, 0, 0, 0, 0, 0};
         DREPENorm = Norm[DREPE[Tkm, Tk, Tkp, Grig6R6, PrigR, Fk, h]];
         showSO3ApparatusWithDREPE[
          t, ωb, Lb, ωs, Ls, Ib, rotMatSO3, DREPENorm, apparatus]],
       step = Function[(* of no arguments *)
         Tkm = Tk; Tk = Tkp;
         {ωb, rotMatSO3, ωs, Lb, Ls} =
          oneStepSO3ForcedRotationalMotion[ωb, rotMatSO3, Ib, Ibi, h, τs];
         Tkp = TSE3[rotMatSO3, o3];
         (* still a quat in the sim *)
         qk = qFromRotMat[rotMatSO3];
         csk = rv[qk, rig["cb"]];
         τsk = (rig["mass"] Abs[g] e3) × csk;
         Fk = {0, 0, 0, 0, 0, 0};
         DREPENorm = Norm[DREPE[Tkm, Tk, Tkp, Grig6R6, PrigR, Fk, h]];
         t += h;
         showSO3ApparatusWithDREPE[
          t, ωb, Lb, ωs, Ls, Ib, rotMatSO3, DREPENorm, apparatus]]},
      init[];
      Dynamic[t += h;
       step[]]]]]]];
runSimForcedRotationalMotionWithDREPErig[rig, 0.03, o, Q[0, 10 °, 0 * -0.5 °], o]
```

Out[601]=



Y
X
Z
1
0
−1
1
0
−1
−1
0
1
t = 18.24
|DREPE| = 0.010596e0
yaw = 0.0000°
pitch = 10.0000°
roll = 0.0000°
|Ls| = 0.0000
|Lb| = 0.0000
ωbᵀIbωb/2 = 0.0000
ωbᵀLb/2 = 0.0000
ωsᵀLs/2 = 0.0000