Kinodynamic Compendium

Wolfgang Hönig

2025-09-01

Table of contents

1 Introduction

This is a summary of important kinodynamic systems of varying complexity to simplify research and benchmarking in the area of motion planning and controls, for single- and multi-robot systems.

Particular emphasis is on including results such as differential flatness, interesting instances (with respect to bounds), and relationship to real robots.

Part I

First Order Systems

2 Single Integrator

3 Unicycle

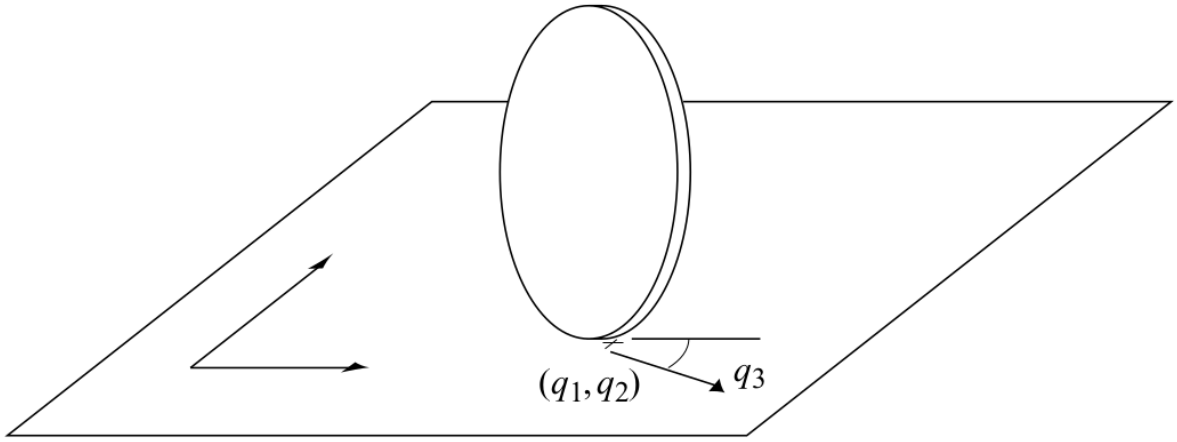

Figure 3.1: Unicycle Robot

3.1 Dynamics

- Parameters
  - state space $\mathcal{X}$
  - action space $\mathcal{U}$
- State: $\mathbf{x} = \big(x, y, \theta\big)^\top \in \mathcal{X}$ [m, m, rad] (position and orientation of the robot in the world frame)
- Action: $\mathbf{u} = \big(v, \omega\big)^\top \in \mathcal{U}$ [m/s, rad/s] (speed and angular speed, respectively)
- Dynamics:

$$\dot{x} = v \cos\theta$$
$$\dot{y} = v \sin\theta$$
$$\dot{\theta} = \omega$$

3.2 Differential Flatness

Pick *flat outputs* $\mathbf{z}(t) = (x(t), y(t))^\top$, i.e., the position of the unicycle. Then we can compute all necessary variables if $\mathbf{z}(t)$ is at least C2-continuous.

$$\mathbf{x}(t) = g_x(\mathbf{z}, \dot{\mathbf{z}}) = \left(x, y, \arctan\left(\frac{\dot{y}}{\dot{x}}\right)\right)$$
$$\mathbf{u}(t) = g_u(\dot{\mathbf{z}}, \ddot{\mathbf{z}}) = \left(\pm\sqrt{\dot{y}^2 + \dot{x}^2}, \frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{\dot{x}^2 + \dot{y}^2}\right)$$

Derivation

Divide y and x-part of the dynamics, yields θ:

$$\frac{\dot{y}}{\dot{x}} = \frac{v \sin\theta}{v \cos\theta}$$
$$\frac{\dot{y}}{\dot{x}} = \tan\theta$$
$$\theta = \arctan\left(\frac{\dot{y}}{\dot{x}}\right)$$

Using the x-part, rearranging for v and substituting the expression for θ, yields v:

$$\begin{aligned} v &= \frac{\dot{x}}{\cos\theta} \\ &= \frac{\dot{x}}{\cos\left(\arctan\left(\frac{\dot{y}}{\dot{x}}\right)\right)} \\ &= \dot{x}\sqrt{\frac{\dot{y}^2}{\dot{x}^2} + 1} = \dot{x}\sqrt{\frac{\dot{y}^2}{\dot{x}^2} + \frac{\dot{x}^2}{\dot{x}^2}} \\ &= \pm\sqrt{\dot{y}^2 + \dot{x}^2} \end{aligned}$$

Taking the time-derivative of the expression for θ yields ω

$$\begin{aligned} \omega = \dot{\theta} &= \frac{d}{dt}\arctan\left(\frac{\dot{y}}{\dot{x}}\right) \\ &= \frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{\dot{x}^2 + \dot{y}^2} \end{aligned}$$

3.3 Invariance

The dynamics are translation-invariant.

3.4 Controllers

3.4.1 Geometric Controller (Kanayama et al. (1990))

- Given reference state $\mathbf{x}_r = \left(x_r, y_r, \theta_r\right)^\top \in \mathcal{X}$ and reference action $\mathbf{u}_r = \left(v_r, \omega_r\right)^\top \in \mathcal{U}$
- $K_x, K_y, K_\theta \in \mathbb{R}^+$ are tuning gains
- Control law:

$$
\begin{aligned}
x_e &= (x_r - x)\cos\theta + (y_r - y)\sin\theta \\
y_e &= -(x_r - x)\sin\theta + (y_r - y)\cos\theta \\
\theta_e &= \theta_d \ominus \theta \\
v &= v_r \cos\theta_e + K_x x_e \\
\omega &= \omega_r + v_r(K_y y_e + K_\theta \sin\theta_e)
\end{aligned}
$$

3.4.2 Action Mixing

Geometric controllers might output actions that are outside the nominal action space $\mathcal{U}$ (saturation limits). To remedy this, a QP can be used that prefers ω over v using a tuning parameter λ:

$$
\begin{aligned}
&\min_{v^*, \omega^*} (\omega^* - \omega)^2 + \lambda(v^* - v)^2 \\
&\text{s.t. } \left(v^*, \omega^*\right)^\top \in \mathcal{U}
\end{aligned}
$$

3.5 Useful Parameters

3.5.1 unicycle1_v0 (Ortiz-Haro et al. (2024))

A basic version

$$\mathcal{U} = [-0.5, 0.5]m/s \times [-0.5, 0.5]rad/s$$

3.5.2 unicycle1_v1 (Ortiz-Haro et al. (2024))

A plane-like version with minimum speed

$$\mathcal{U} = [0.25, 0.5]m/s \times [-0.5, 0.5]rad/s$$

3.5.3 unicycle1_v2 (Ortiz-Haro et al. (2024))

A plane-like version with a rudder damage

$$\mathcal{U} = [0.25, 0.5]m/s \times [-0.25, 0.5]rad/s$$

4 Differential Drive

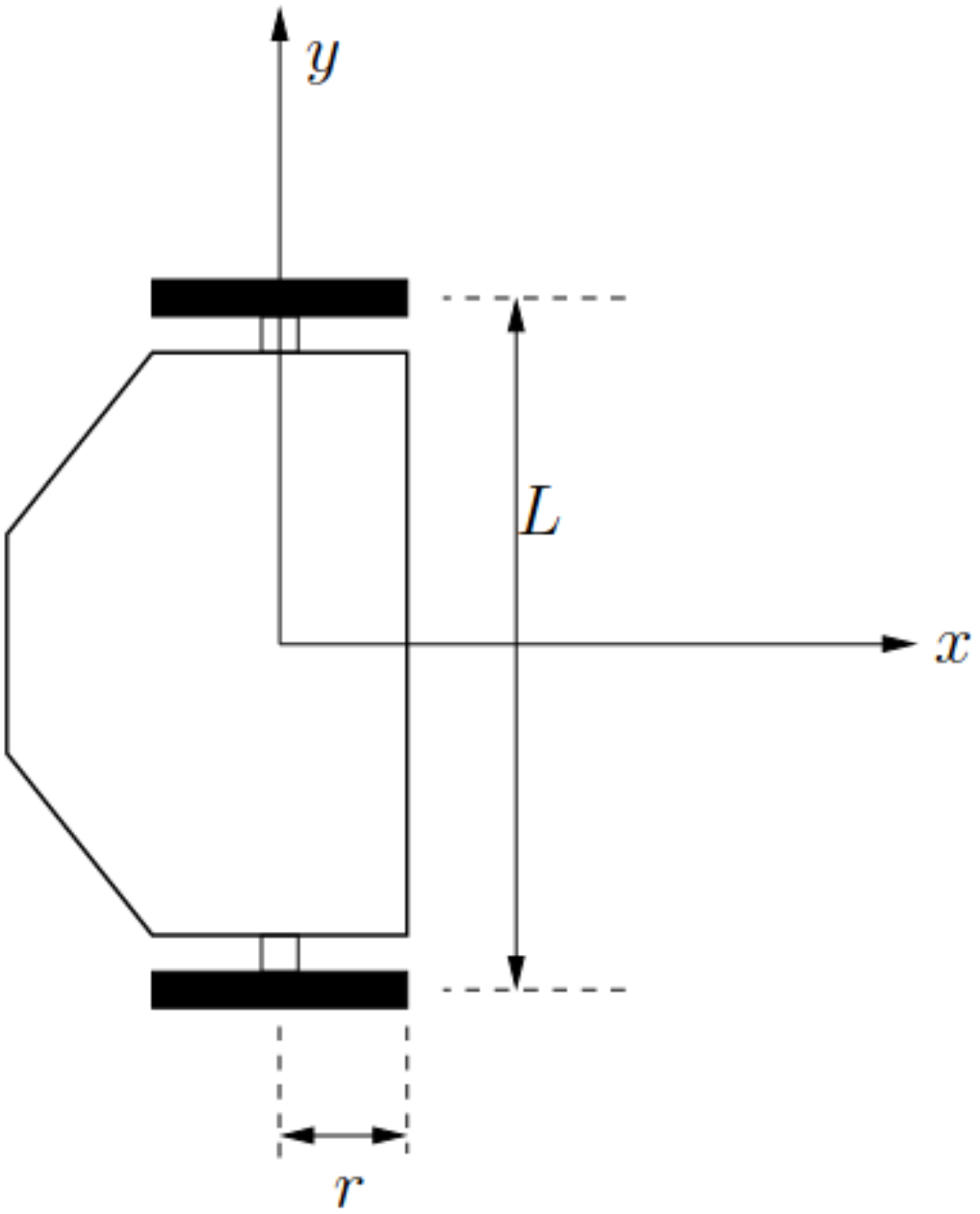

Figure 4.1: Differential Drive Robot (Picture from LaValle (2006))

4.1 Dynamics

- Parameters
  - r: radius of the wheels [m]
  - L: distance between the wheels [m]
- State: $\mathbf{x} = (x, y, \theta)^\top$ [m, m, rad] (position and orientation of the robot in the world frame)
- Action: $\mathbf{u} = (u_l, u_r)^\top$ [rad/s, rad/s] (angular velocity of the left and right wheels, respectively)
- Dynamics:

$$\begin{aligned}\dot{x} &= \frac{r}{2}(u_l + u_r)\cos\theta \\ \dot{y} &= \frac{r}{2}(u_l + u_r)\sin\theta \\ \dot{\theta} &= \frac{r}{L}(u_r - u_l)\end{aligned}$$

4.1.1 Relationship to Unicycle

There is a linear relationship between the differential drive and the unicycle. One can substitute actions to $v = \frac{r}{2}(u_l + u_r)$ and $\omega = \frac{r}{L}(u_r - u_l)$ to get

$$\begin{aligned}\dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \omega\end{aligned}$$

Then, the "original" actions can be computed as

$$\begin{aligned}u_l &= \frac{2v - L\omega}{2r} \\ u_r &= \frac{2v + L\omega}{2r}\end{aligned}$$

> **Derivation**
>
> $$2v = r(u_l + u_r) \tag{4.1}$$
>
> $$L\omega = r(u_r - u_l) \tag{4.2}$$
>
> Adding Equation 4.1 and Equation 4.2:

$$
\begin{aligned}
2v + L\omega = r(u_l + u_r + u_r - u_l) = 2ru_r \\
u_r = \frac{2v + L\omega}{2r}
\end{aligned}
\tag{4.3}
$$

Substituting Equation 4.3 in Equation 4.1:

$$
\begin{aligned}
2v = r(u_l + u_r) = r(u_l + \frac{2v + L\omega}{2r}) = ru_l + v + \frac{L\omega}{2} \\
u_l = \frac{2v - v - \frac{L\omega}{2}}{r} = \frac{2v - L\omega}{2r}
\end{aligned}
\tag{4.4}
$$

4.2 Differential Flatness

Pick *flat outputs* $\mathbf{z}(t) = (x(t), y(t))^\top$, i.e., the position of the robot. Then we can compute all necessary variables if $\mathbf{z}(t)$ is at least C2-continuous.

$$
\begin{aligned}
\mathbf{x}(t) &= g_x(\mathbf{z}, \dot{\mathbf{z}}) = \left(x, y, \arctan\left(\frac{\dot{y}}{\dot{x}}\right)\right) \\
\mathbf{u}(t) &= g_u(\dot{\mathbf{z}}, \ddot{\mathbf{z}}) = \left(\frac{2v - L\omega}{2r}, \frac{2v + L\omega}{2r}\right), \text{ where} \\
v &= \pm\sqrt{\dot{y}^2 + \dot{x}^2} \\
\omega &= \frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{\dot{x}^2 + \dot{y}^2}
\end{aligned}
$$

i Derivation

One can apply the same differential flatness result from the unicycle, followed by the linear mapping derived earlier to recover the actual actions g_u.

4.3 Invariance

The dynamics are translation-invariant.

4.4 Controllers

4.4.1 Geometric Controller (Kanayama et al. (1990))

This is the same controller as for the unicycle, with the linear mapping applied.

- Given reference state $\mathbf{x}_r = \left(x_r, y_r, \theta_r\right)^\top \in \mathcal{X}$ and reference action $\mathbf{u}_r = \left(u_{l,r}, u_{r,r}\right)^\top \in \mathcal{U}$
- $K_x, K_y, K_\theta \in \mathbb{R}^+$ are tuning gains
- Control law:

$$
\begin{aligned}
v_r &= \frac{r}{2}(u_{l,r} + u_{r,r}) \\
\omega_r &= \frac{r}{L}(u_{r,r} - u_{l,r}) \\
x_e &= (x_r - x)\cos\theta + (y_r - y)\sin\theta \\
y_e &= -(x_r - x)\sin\theta + (y_r - y)\cos\theta \\
\theta_e &= \theta_d \ominus \theta \\
v &= v_r \cos\theta_e + K_x x_e \\
\omega &= \omega_r + v_r(K_y y_e + K_\theta \sin\theta_e) \\
u_l &= \frac{2v - L\omega}{2r} \\
u_r &= \frac{2v + L\omega}{2r}
\end{aligned}
$$

4.4.2 Action Mixing

Geometric controllers might output actions that are outside the nominal action space $\mathcal{U}$ (saturation limits). To remedy this, a QP can be used that prefers ω over v using a tuning parameter λ:

$$
\begin{aligned}
&\min_{v^*, \omega^*} (\omega^* - \omega)^2 + \lambda(v^* - v)^2 \\
&\text{s.t. } \left(\tfrac{2v^* - L\omega^*}{2r}, \tfrac{2v^* + L\omega^*}{2r}\right)^\top \in \mathcal{U}
\end{aligned}
$$

followed by converting the result back to the desired control values:

$$
\begin{aligned}
u_l &= \frac{2v^* - L\omega^*}{2r} \\
u_r &= \frac{2v^* + L\omega^*}{2r}
\end{aligned}
$$

4.5 Useful Parameters

4.5.1 pololu-3piplus-hyper

Commercially off-the-shelves (Cots) robot: Pololu 3pi+ Hyper Edition

$$L = 0.089$$
$$r = 0.016$$
$$\mathcal{U} = [-157.08, 157.08]rad/s \times [-157.08, 157.08]rad/s$$

5 Car

6 Car With Trailer

Part II

Second Order Systems

7 Double Integrator

8 Multirotor (1D)

9 Multirotor (2D)

10 Lunar Lander

11 Multirotor

12 Summary

To be filled.

References

Huynh, Du Q. 2009. "Metrics for 3D Rotations: Comparison and Analysis." *Journal of Mathematical Imaging and Vision* 35 (2): 155–64. https://doi.org/10.1007/s10851-009-0161-2.

Jia, Prof. Yan-Bin. 2024. "Quaternions (Com s 477/577 Notes)." 2024. https://faculty.sites.iastate.edu/jia/files/inline-files/quaternion.pdf.

Kanayama, Y., Y. Kimura, F. Miyazaki, and T. Noguchi. 1990. "A Stable Tracking Control Method for an Autonomous Mobile Robot." In *IEEE International Conference on Robotics and Automation (ICRA)*, 384–89. https://doi.org/10.1109/ROBOT.1990.126006.

LaValle, Steven M. 2006. *Planning Algorithms*. Cambridge University Press.

Narayan, Dr. Ashwin. 2017. "How to Integrate Quaternions." 2017. https://www.ashwinnarayan.com/post/how-to-integrate-quaternions/.

Ortiz-Haro, Joaquim, Wolfgang Hönig, Valentin N. Hartmann, and Marc Toussaint. 2024. "iDb-a*: Iterative Search and Optimization for Optimal Kinodynamic Motion Planning." *IEEE Transactions on Robotics*, 1–19. https://doi.org/10.1109/TRO.2024.3502505.

Särkkä, Prof. Simo. 2007. "Notes on Quaternions." 2007. https://users.aalto.fi/~ssarkka/pub/quat.pdf.

Shoemake, Ken. 1992. "Uniform Random Rotations." In *Graphics Gems III*, 124–32. USA: Academic Press Professional, Inc.

Toussaint, Prof. Marc. 2024. "Lecture Note: Quaternions, Exponential Map, and Quaternion Jacobians." 2024. https://www.user.tu-berlin.de/mtoussai/notes/quaternions.html.

A ODEs

B SO(2) - Rotations in 2D

C SO(3) - Rotations in 3D

C.1 Introduction

For rotations without singularities, rotation matrices and quaternions are common. The special orthogonal group SO(3) is often defined using a constrained set of 3×3 rotation matrices:

$$SO(3) = \{R \in \mathbb{R}^{3\times3} | R^\top R = RR^\top = I \wedge \det R = 1\} \tag{C.1}$$

Quaternions have the advantage of a lower memory requirement (4 floats vs. 9 in a rotation matrix), often lower computational effort, and that their constraints are easier to fulfill when integrating (one norm constraint rather than the two constraints above). This document summarizes the mathematical operations for quaternions and provides useful references for derivations.

Notable implementations for quaternions are:

Language	Link
Python	Rowan
C++	Eigen, Sophus
C	cmath3d

Notable other references are: Särkkä (2007), Toussaint (2024), Jia (2024).

C.2 Definition

A quaternion for SO(3) is a 4D vector with the constraint to have norm 1:

$$\mathbf{q} = \begin{pmatrix} q_w \\ q_x \\ q_y \\ q_z \end{pmatrix} = \begin{pmatrix} q_w \\ \vec{\mathbf{q}} \end{pmatrix} \in \mathbb{R}^4 \; s.t. \|\mathbf{q}\|_2 = 1 \tag{C.2}$$

One can *augment* or *promote* a vector $\mathbf{v} \in \mathbb{R}^3$ to be a quaternion:

$$\bar{\mathbf{v}} = \begin{pmatrix} 0 \\ \mathbf{v} \end{pmatrix} \tag{C.3}$$

One can *extract* the vector or imaginary part of a quaternion:

$$\vec{\mathbf{q}} = \mathrm{Im}(\mathbf{q}) \tag{C.4}$$

Note that the order of the components is not uniquely defined. Some software packages put q_w first, some last in the vector.

C.3 Conversion

C.3.1 To Rotation Matrix

$$R(\mathbf{q}) = \begin{pmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_w q_z) & 2(q_w q_y + q_x q_z) \\ 2(q_x q_y + q_w q_z) & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z - q_w q_x) \\ 2(q_x q_z - q_w q_y) & 2(q_w q_x + q_y q_z) & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{pmatrix}$$

C.3.2 From Rotation Matrix

C.4 Low-Level Operations

C.4.1 Negation

(same as any vector):

$$-\mathbf{q} = \begin{pmatrix} -q_w \\ -q_x \\ -q_y \\ -q_z \end{pmatrix}$$

Note that $\mathbf{q}$ and $-\mathbf{q}$ represent the same rotation.

C.4.2 Addition, Substraction, Multiplication/Devision with Scalar

same as any vector, e.g.,:

$$\mathbf{q} + \mathbf{p} = \begin{pmatrix} q_w + p_w \\ q_x + p_x \\ q_y + p_y \\ q_z + p_z \end{pmatrix}$$

C.4.3 Conjugate

$$\mathbf{q}^* = \begin{pmatrix} q_w \\ -q_x \\ -q_y \\ -q_z \end{pmatrix}$$

Note that this is the inverse for normalized quaternions.

C.4.4 Norm

(regular L_2 norm on the vector)

$$\|\mathbf{q}\| = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2}$$

Note that this should be always 1 for normalized quaternions.

C.4.5 Exponential

$$\exp(\mathbf{q}) = \exp(q_w) \begin{pmatrix} \cos(\|\vec{\mathbf{q}}\|) \\ \frac{\vec{\mathbf{q}}}{\|\vec{\mathbf{q}}\|} \sin(\|\vec{\mathbf{q}}\|) \end{pmatrix}$$

C.4.6 Logarithm

$$\ln(\mathbf{q}) = \begin{pmatrix} \ln(\|\mathbf{q}\|) \\ \frac{\vec{\mathbf{q}}}{\|\vec{\mathbf{q}}\|} \arccos\left(\frac{q_w}{\|\mathbf{q}\|}\right) \end{pmatrix}$$

C.4.7 Power with Real Number

$$\mathbf{q}^p = \exp(\ln(\mathbf{q})p)$$

C.4.8 Power with Quaternion

$$\mathbf{q}^{\mathbf{p}} = \exp\left(\ln(\mathbf{q}) \otimes \mathbf{p}\right)$$

C.5 Useful Operations

C.5.1 Multiplication

Similar to rotation matrices, this can be used to concatenate rotations.

$$\mathbf{q} \otimes \mathbf{p} = \begin{pmatrix} q_w p_w - q_x p_x - q_y p_y - q_z p_z \\ q_x p_w + q_w p_x - q_z p_y + q_y p_z \\ q_y p_w + q_z p_x + q_w p_y - q_x p_z \\ q_z p_w - q_y p_x + q_x p_y + q_w p_z \end{pmatrix}$$

Note that generally $\mathbf{q} \otimes \mathbf{p} \neq \mathbf{p} \otimes \mathbf{q}$.

Be careful when looking at papers - sometimes it's defined inverse!

C.5.2 Vector Rotation

With a rotation matrix $\mathbf{R}(\mathbf{q})$, we can rotate a vector $\mathbf{v}$ as $\mathbf{v}_{rotated} = \mathbf{R}(\mathbf{q})\mathbf{v}$. Similarily, for quaternions we have:

$$\mathbf{v}_{rotated} = \mathbf{q} \odot \mathbf{v} = \mathrm{Im}(\mathbf{q} \otimes \overline{\mathbf{v}} \otimes \mathbf{q}^*)$$

(I.e., augment $\mathbf{v}$ to a quaternion, use two quaternion multiplications, and one conjugate, and then extract the vector part of the result.)

TODO: THERE IS A FASTER VERSION, see https://en.wikipedia.org/wiki/Conversion_between_quaternion

C.6 Derivatives and Integration

Assuming $\omega \in \mathbb{R}^3$ is the angular velocity in *body frame* (e.g., gyroscope):

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q} \otimes \overline{\omega}$$

Assuming ω in *world frame*:

$$\dot{\mathbf{q}} = \frac{1}{2}\overline{\omega} \otimes \mathbf{q}$$

This can also be used to numerically estimate ω. One can approximate $\dot{\mathbf{q}} \approx (\mathbf{q}(t + \Delta t) - \mathbf{q}(t))/\Delta t$ and then proceed (remembering to use $\mathbf{q}^*$ as inverse of $\mathbf{q}$). A nicer solution can be derived to reach the following (*body frame*)[1]:

$$\omega \approx 2\,\mathrm{Im}\left(\frac{\mathbf{q}^*(t) \otimes \mathbf{q}(t + \Delta t)}{\Delta t}\right)$$

Or similarly in *world frame*:

$$\omega \approx 2\,\mathrm{Im}\left(\frac{\mathbf{q}(t + \Delta t) \otimes \mathbf{q}^*(t)}{\Delta t}\right)$$

For integration, one can use Euler or RK4 to integrate $\mathbf{q}$, followed by normalizing the quaternion. Alternatively, the exponential map can be used as a more accurate, explicit integrator (*body frame*):

$$\mathbf{q}_{t+\Delta t} = \mathbf{q}_t \otimes \begin{pmatrix} \cos\left(\|\omega\|\Delta t/2\right) \\ \frac{\omega}{\|\omega\|}\sin\left(\|\omega\|\Delta t/2\right) \end{pmatrix}$$

A derivation is in Narayan (2017). Note that some libraries do not specify which frame they operate in, e.g., the Python library rowan assumes *world frame* for its methods.

C.7 Interpolation

If we want to linearly interpolate from a rotation $\mathbf{q}(0)$ to another rotation $\mathbf{q}(1)$, we can use the **slerp** operation:

$$\mathbf{q}(t) = \left(\mathbf{q}(1) \otimes \mathbf{q}(0)^*\right)^t \otimes \mathbf{q}(0) \quad \forall t \in [0, 1]$$

[1] See this math.stackexchange post.

C.8 Metrics

A metric measures the difference between two rotations as a scalar value. A comparison of different metrics is in Huynh (2009). Below are common metrics.

The following is metric 2 in Huynh (2009) and **sym_distance** in rowan:

$$d(\mathbf{q}, \mathbf{p}) = \min\left(\|\mathbf{q} - \mathbf{p}\|, \|\mathbf{q} + \mathbf{p}\|\right) \in [0, \sqrt{2}]$$

The following is metric 3 in Huynh (2009), is used by OMPL, and called **sym_intrinsic_distance** in rowan:

$$d(\mathbf{q}, \mathbf{p}) = \arccos|\mathbf{q} \cdot \mathbf{p}| \in [0, \pi/2]$$

The following is metric 4 in Huynh (2009):

$$d(\mathbf{q}, \mathbf{p}) = 1 - |\mathbf{q} \cdot \mathbf{p}| \in [0, 1]$$

C.9 Random Generation

Sampling rotations uniformly is not trivial. Here is one approach Shoemake (1992):

$$r_1, r_2, r_3 \sim \mathcal{U}(0, 1)$$
$$\mathbf{q} = \begin{pmatrix} \sqrt{1 - r_1}\sin 2\pi r_2 \\ \sqrt{1 - r_1}\cos 2\pi r_2 \\ \sqrt{r_1}\sin 2\pi r_3 \\ \sqrt{r_1}\cos 2\pi r_3 \end{pmatrix}$$

Note, this is also used inside OMPL.