



Introduction to Embodied AI

Lecture 3 - Robotics II

Prof. He Wang



$\mathbb{SO}(3)$

Rotation Matrix

$\mathbb{SO}(3)$: The Space of Rotations

- $\mathbb{SO}(n) = \{R \in \mathbb{R}^{n \times n} : \det(R) = 1, RR^T = I\}$
- $\mathbb{SO}(n)$: “Special Orthogonal Group”
- “Group”: roughly, closed under matrix multiplication
- “Orthogonal”: $RR^T = I$
- “Special”: $\det(R) = 1$
- $\mathbb{SO}(2)$: 2D rotations, 1 DoF
- $\mathbb{SO}(3)$: 3D rotations, 3 DoF

The Issues of Rotation Matrix as Rotation Representation

1. Efficiency

Rotation has 3 DoF, however a rotation matrix needs 9 values

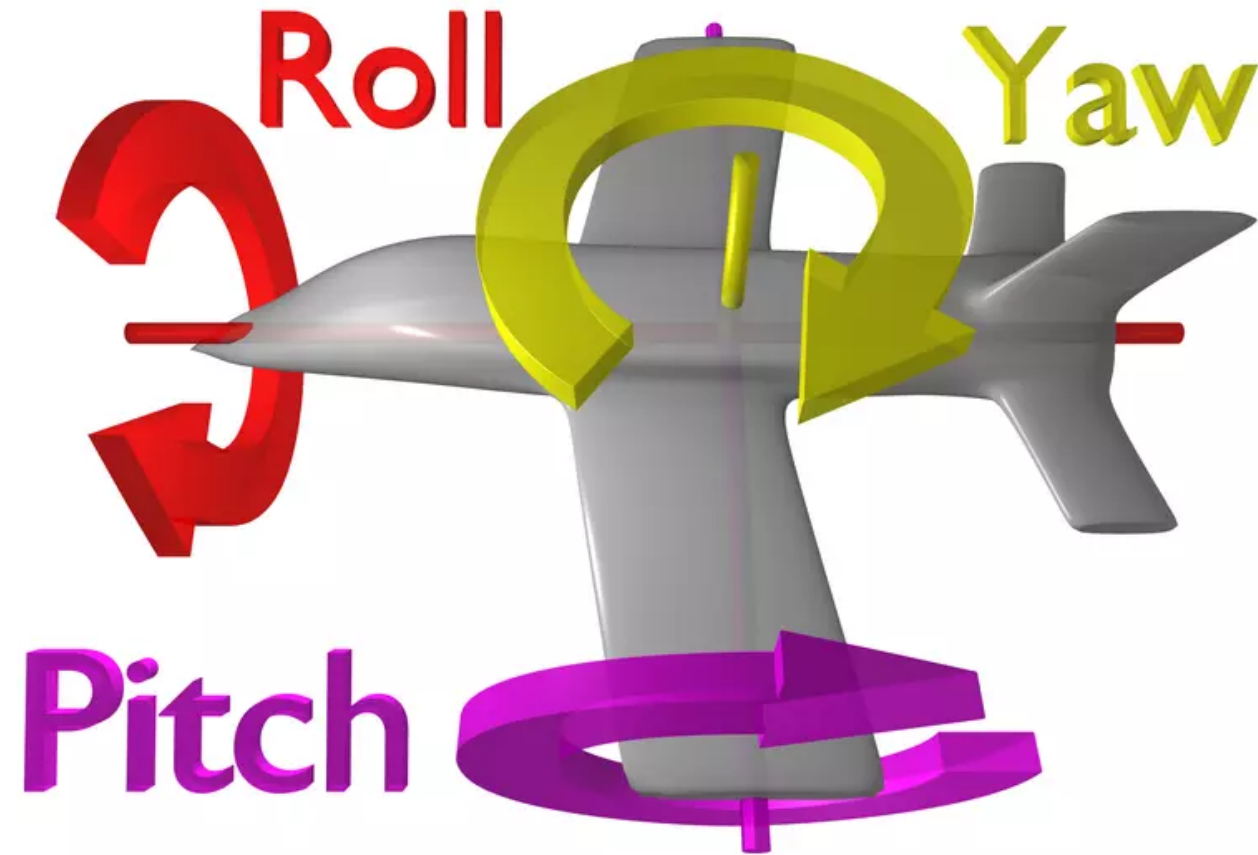
2. Numerical stability

Maintain orthogonality

Euler Angle is Very Intuitive



Euler Angle



<https://blog.csdn.net/sandaiphon4869>

Gimbal Lock

- For example: When $\beta = \pi/2$,

$$R = R_z(\alpha)R_y(\pi/2)R_x(\gamma)$$

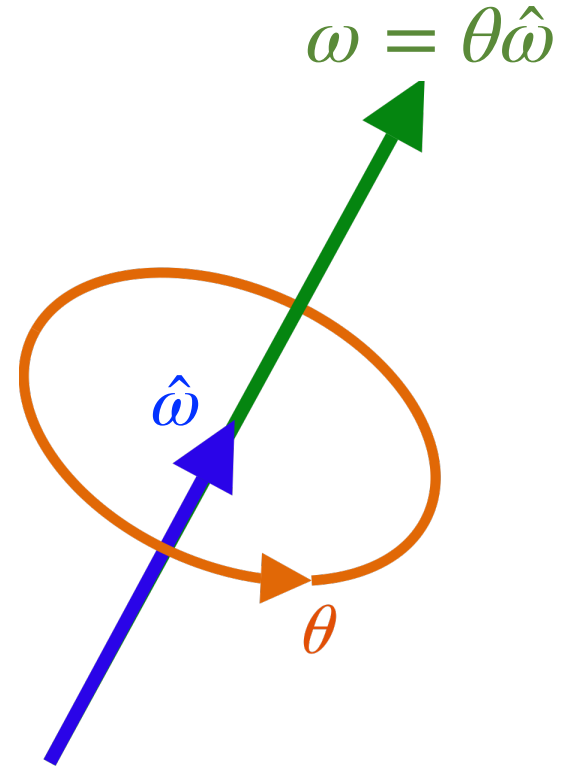
$$= \begin{bmatrix} 0 & 0 & 1 \\ \sin(\alpha + \gamma) & \cos(\alpha + \gamma) & 0 \\ -\cos(\alpha + \gamma) & \sin(\alpha + \gamma) & 0 \end{bmatrix}$$

since changing α and γ has the same effects, a degree of freedom disappears!

Angle-Axis Parameterization of Rotations

Euler's Theorem

- Any rotation is equivalent to a rotation about a fixed axis $\hat{\omega} \in \mathbb{R}^3$ ($\|\hat{\omega}\| = 1$) through a positive angle θ
- $\hat{\omega}$: unit vector of rotation axis
- θ : angle of rotation
- $R \in \mathbb{SO}(3) := \text{Rot}(\hat{\omega}, \theta)$



Given $\hat{\omega}$ and θ , what is $R \in \mathbb{SO}(3)$?

Given $\hat{\omega}$ and θ , what is $R \in \text{SO}(3)$?

- By $\text{Rot}(\hat{\omega}, \theta)x = e^{[\hat{\omega}]\theta}x, \forall x \in \mathbb{R}^3$,

$$\text{Rot}(\hat{\omega}, \theta) \equiv e^{[\hat{\omega}]\theta}$$

- This is under such a **Definition of Matrix Exponential**:

$$e^{[\hat{\omega}]\theta} = I + \theta[\hat{\omega}] + \frac{\theta^2}{2!}[\hat{\omega}]^2 + \frac{\theta^3}{3!}[\hat{\omega}]^3 + \dots$$

- By (1) in the last slide, we obtain **Rodrigues** formula:

$$e^{[\hat{\omega}]\theta} = I + [\hat{\omega}]\sin \theta + [\hat{\omega}]^2(1 - \cos \theta)$$

Given $\hat{\omega}$ and θ , what is $R \in \text{SO}(3)$?

- First question: Is there a **unique** parametrization?

- No:

1. $(\hat{\omega}, \theta)$ and $(-\hat{\omega}, -\theta)$ give the same rotation

2. when $R = I$, $\theta = 0$ and $\hat{\omega}$ can be arbitrary

3. $(\hat{\omega}, \pi)$ and $(-\hat{\omega}, \pi)$ give the same rotation
($\text{tr}(R) = -1$)

- If we restrict $\theta \in (0, \pi)$, a unique parameterization exists:

- $\theta = \arccos \frac{1}{2}[\text{tr}(R) - 1], \quad [\hat{\omega}] = \frac{1}{2 \sin \theta}(R - R^T)$

Quaternion

Note: In this section, $\vec{x} \in \mathbb{R}^3$ and $q \in \mathbb{R}^4$

Quaternion is a “Number”

- Recall the complex number $a + b\mathbf{i}$
- Quaternion is a more generalized complex number:

$$q = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$$

- w is the real part and $\vec{v} = (x, y, z)$ is the imaginary part
- Imaginary: $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$
- anti-commutative : $\mathbf{ij} = \mathbf{k} = -\mathbf{ji}$, $\mathbf{jk} = \mathbf{i} = -\mathbf{kj}$, $\mathbf{ki} = \mathbf{j} = -\mathbf{ik}$

Properties of General Quaternions

- Vector form: $q = (w, \vec{v})$
- Product:
 - For $q_1 = (w_1, \vec{v}_1)$ and $q_2 = (w_2, \vec{v}_2)$,
 $q_1 q_2 = (w_1 w_2 - \vec{v}_1^T \vec{v}_2, w_1 \vec{v}_2 + w_2 \vec{v}_1 + \vec{v}_1 \times \vec{v}_2)$
 - Not commutable (note that $\vec{v}_1 \times \vec{v}_2 \neq \vec{v}_2 \times \vec{v}_1$)
- Conjugate: $q^* = (w, -\vec{v})$
- Norm: $\|q\|^2 = w^2 + \vec{v}^T \vec{v} = qq^* = q^*q$
- Inverse: $q^{-1} := \frac{q^*}{\|q\|^2}$

Unit Quaternion as Rotation

- A **unit** quaternion $\|q\| = 1$ can represent a rotation
 - Four numbers plus one constraint $\rightarrow$ 3 DoF
- Geometrically, the shell of a 4D sphere

Build Rotation Quaternion

- Exponential coordinate $\rightarrow$ Quaternion:

$$q = [\cos(\theta/2), \sin(\theta/2)\hat{w}]$$

- Quaternion is very close to angle-axis representation!
- Exponential coordinate $\leftarrow$ Quaternion: $\theta = 2 \arccos(w)$,

$$\hat{w} = \begin{cases} \frac{1}{\sin(\theta/2)} \vec{v} & \theta \neq 0 \\ 0 & \theta = 0 \end{cases}$$

Unit Quaternion as Rotation

- Rotate a vector $\vec{x}$ by a quaternion q :
 1. Augment $\vec{x}$ to $x = (0, \vec{x})$
 2. $x' = qxq^{-1}$
- Compose rotations by quaternion:
 - $(q_2(q_1xq_1^*)q_2^*)$: first rotate by q_1 and then by q_2
 - Since $(q_2(q_1xq_1^*)q_2^*) = (q_2q_1)x(q_1^*q_2^*)$, **composing rotations is as simple as multiplying quaternions!**

Proof

Theorem 1 *For any unit quaternion*

$$q = q_0 + \mathbf{q} = \cos \frac{\theta}{2} + \mathbf{u} \sin \frac{\theta}{2}, \quad (4)$$

and for any vector $\mathbf{v} \in \mathbb{R}^3$ the action of the operator

$$L_q(\mathbf{v}) = q\mathbf{v}q^*$$

on $\mathbf{v}$ is equivalent to a rotation of the vector through an angle θ about $\mathbf{u}$ as the axis of rotation.

Proof Given a vector $\mathbf{v} \in \mathbb{R}^3$, we decompose it as $\mathbf{v} = \mathbf{a} + \mathbf{n}$, where $\mathbf{a}$ is the component along the vector $\mathbf{q}$ and $\mathbf{n}$ is the component normal to $\mathbf{q}$. Then we show that under the operator L_q , $\mathbf{a}$ is invariant, while $\mathbf{n}$ is rotated about $\mathbf{q}$ through an angle θ . Since the operator is linear, this shows that the image $q\mathbf{v}q^*$ is indeed interpreted as a rotation of $\mathbf{v}$ about $\mathbf{q}$ through an angle θ .

We know from an early reasoning that $\mathbf{a}$ is invariant under L_q . So let us focus on the effect of L_q on the orthogonal component $\mathbf{n}$. We have

$$\begin{aligned} L_q(\mathbf{n}) &= (q_0^2 - \|\mathbf{q}\|^2)\mathbf{n} + 2(\mathbf{q} \cdot \mathbf{n})\mathbf{q} + 2q_0(\mathbf{q} \times \mathbf{n}) \\ &= (q_0^2 - \|\mathbf{q}\|^2)\mathbf{n} + 2q_0(\mathbf{q} \times \mathbf{n}) \\ &= (q_0^2 - \|\mathbf{q}\|^2)\mathbf{n} + 2q_0\|\mathbf{q}\|(\mathbf{u} \times \mathbf{n}), \end{aligned}$$

where in the last step above we introduced $\mathbf{u} = \mathbf{q}/\|\mathbf{q}\|$. Denote $\mathbf{n}_\perp = \mathbf{u} \times \mathbf{n}$. So the last equation becomes

$$L_q(\mathbf{n}) = (q_0^2 - \|\mathbf{q}\|^2)\mathbf{n} + 2q_0\|\mathbf{q}\|\mathbf{n}_\perp. \quad (5)$$

Note that $\mathbf{n}_\perp$ and $\mathbf{n}$ have the same length:

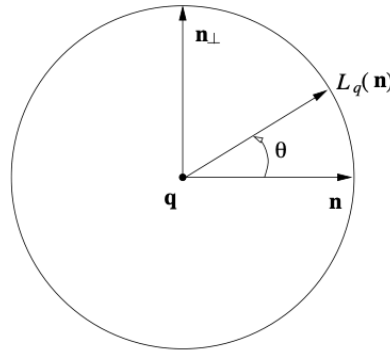
$$\|\mathbf{n}_\perp\| = \|\mathbf{n} \times \mathbf{u}\| = \|\mathbf{n}\| \cdot \|\mathbf{u}\| \sin \frac{\pi}{2} = \|\mathbf{n}\|.$$

Proof (Cont'd)

Finally, we rewrite (5) into the form

$$\begin{aligned} L_q(\mathbf{n}) &= \left(\cos^2 \frac{\theta}{2} - \sin^2 \frac{\theta}{2} \right) \mathbf{n} + \left(2 \cos \frac{\theta}{2} \sin \frac{\theta}{2} \right) \mathbf{n}_\perp \\ &= \cos \theta \mathbf{n} + \sin \theta \mathbf{n}_\perp. \end{aligned}$$

Namely, the resulting vector is a rotation of $\mathbf{n}$ through an angle θ in the plane defined by $\mathbf{n}$ and $\mathbf{n}_\perp$. See the figure below. This vector is clearly orthogonal to the rotation axis.



We substitute the unit quaternion form (4) into (3) to obtain the resulting vector from rotating a vector $\mathbf{v}$ about the axis $\mathbf{u}$ through θ :

$$\begin{aligned} L_p(\mathbf{v}) &= \left(\cos^2 \frac{\theta}{2} - \sin^2 \frac{\theta}{2} \right) \mathbf{v} + 2 \left(\mathbf{u} \sin \frac{\theta}{2} \cdot \mathbf{v} \right) \mathbf{u} \sin \frac{\theta}{2} + 2 \cos \frac{\theta}{2} \left(\mathbf{u} \sin \frac{\theta}{2} \times \mathbf{v} \right) \\ &= \cos \theta \cdot \mathbf{v} + (1 - \cos \theta)(\mathbf{u} \cdot \mathbf{v})\mathbf{u} + \sin \theta \cdot (\mathbf{u} \times \mathbf{v}). \end{aligned} \tag{6}$$

$$\text{Rot}(\mathbf{u}, \theta)\mathbf{v} = \mathbf{v} + (1 - \cos \theta)\mathbf{u} \times (\mathbf{u} \times \mathbf{v}) + (\sin \theta)\mathbf{u} \times \mathbf{v}$$

$$\because \mathbf{u} \times (\mathbf{u} \times \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})\mathbf{u} - (\mathbf{u} \cdot \mathbf{u})\mathbf{v} = (\mathbf{u} \cdot \mathbf{v})\mathbf{u} - \mathbf{v}$$

$$\therefore \text{Rot}(\mathbf{u}, \theta)\mathbf{v} = \mathbf{v} \cos \theta + (1 - \cos \theta)(\mathbf{u} \cdot \mathbf{v})\mathbf{u} + (\sin \theta)\mathbf{u} \times \mathbf{v} = L_p(\mathbf{v})$$

Unit Quaternion as Rotation

- Rotate a vector $\vec{x}$ by a quaternion q :
 1. Augment $\vec{x}$ to $x = (0, \vec{x})$
 2. $x' = qxq^{-1}$
- Compose rotations by quaternion:
 - $(q_2(q_1xq_1^*)q_2^*)$: first rotate by q_1 and then by q_2
 - Since $(q_2(q_1xq_1^*)q_2^*) = (q_2q_1)x(q_1^*q_2^*)$, **composing rotations is as simple as multiplying quaternions!**
- Each rotation corresponds to two quaternions (“double-covering”)

Quaternion vs. Rotation Matrix

Storage:

- A quaternion: 4 floating-point values (x, y, z, w),
- A rotation matrix: 9 values (3×3 matrix).

Multiplication:

- Quaternion : 16 multiplications and 12 additions per operation
- Rotation matrix: 27 multiplications and 18 additions

Numerical stability:

- Rotation matrices can accumulate numerical errors over successive operations (e.g., due to floating-point precision limits), potentially violating orthogonality.
- Quaternions, when normalized, maintain unit magnitude.

More about Quaternion

- Quaternion is computationally cheap:
 - Internal representation of Physical Engine and Robot
- Pay attention to convention (w, x, y, z) or (x, y, z, w)
 - (w, x, y, z) : SAPIEN, transforms3d, Eigen, blender, MuJoCo, V-Rep
 - (x, y, z, w) : ROS, PhysX, PyBullet

Conversion between Quaternion and Axis-Angle

- Axis-angle $\rightarrow$ Quaternion:

$$q = [\cos(\theta/2), \sin(\theta/2)\hat{w}]$$

- Quaternion is very close to angle-axis representation!
- Axis-angle $\leftarrow$ Quaternion: $\theta = 2 \arccos(w)$,

$$\hat{w} = \begin{cases} \frac{1}{\sin(\theta/2)} \vec{v} & \theta \neq 0 \\ 0 & \theta = 0 \end{cases}$$

Conversion between Quaternion and Rotation Matrix

- Rotation Matrix <- Quaternion

- Since we have $x' = qxq^{-1}$, we can calculate the rotation matrix as:

- $$R(q) = \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2zw & 2xz + 2yw \\ 2xy + 2zw & 1 - 2x^2 - 2z^2 & 2yz - 2xw \\ 2xz - 2yw & 2yz + 2xw & 1 - 2x^2 - 2y^2 \end{bmatrix}$$

- Quaternion <- Rotation Matrix

- As $tr(R) = 3 - 4(x^2 + y^2 + z^2) = 4w^2 - 1$ ($tr(R)$ is the trace of R)

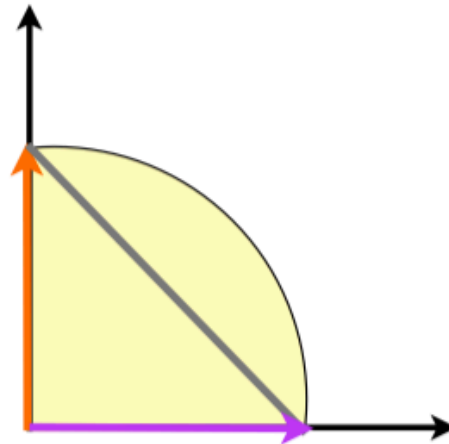
- We can calculate the quaternion as $w = \frac{\sqrt{tr(R)+1}}{2}$, $x = \frac{R_{32}-R_{23}}{4w}$, $y = \frac{R_{13}-R_{31}}{4w}$, $z = \frac{R_{21}-R_{12}}{4w}$

Distance between Quaternions

- The distance between quaternions (q_1, q_2) ?
 - The distance in $\mathbb{S}^3$ or the angle between (q_1, q_2) :
 $\angle p, q = \arccos(p \cdot q)$
 - The distance between the corresponding rotations
 - $\text{dist}(p, q) = 2 \arccos(|p \cdot q|)$ (prove by yourself)
 $= 2 \min \{ \angle p, q, \angle p, -q \}$
- The distance between rotations (R_1, R_2) is linear to the on-sphere distance between $q(R_1)$ and $q(R_2)$.

Interpolating between quaternions

- Why not linear interpolation?
 - Need to be normalized
 - Does not have a constant rate of rotation

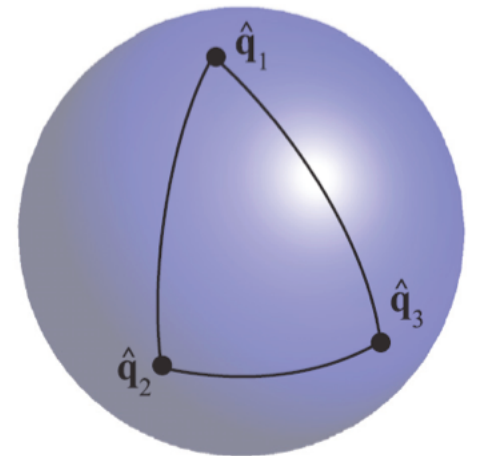


$$\frac{(1 - \alpha)x + \alpha y}{\|(1 - \alpha)x + \alpha y\|}$$

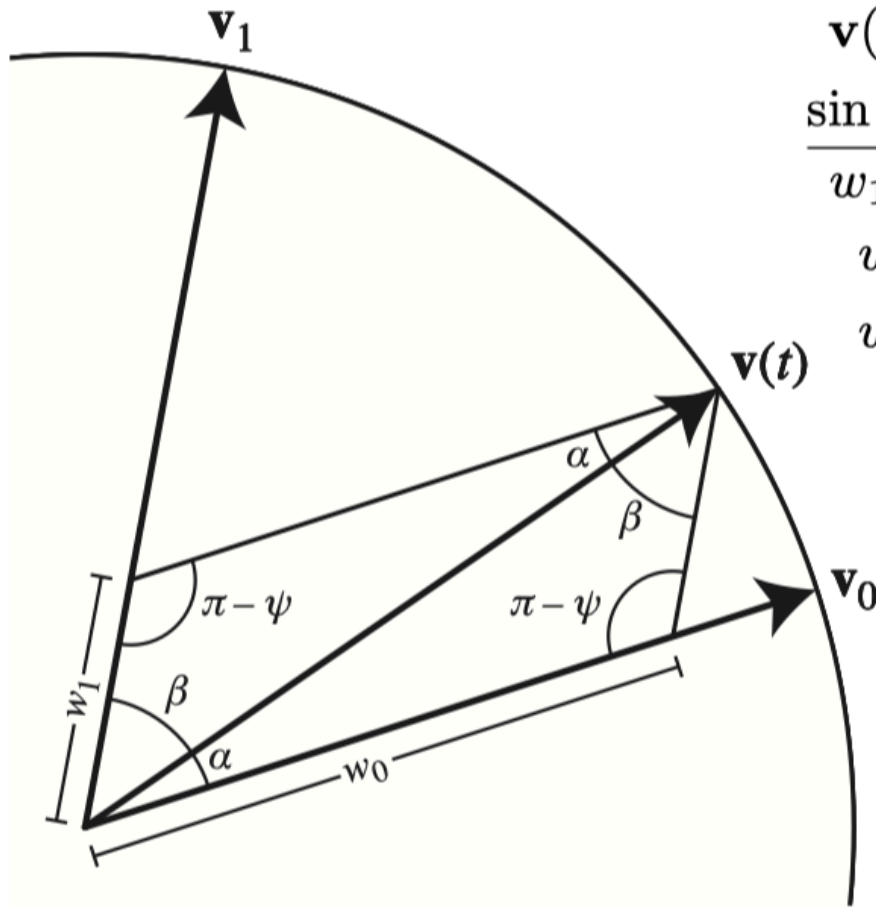
Spherical Linear Interpolation

Intuitive interpolation between different orientations

- Nicely represented through quaternions
- **Useful for animation and motion planning**
- Given two quaternions, interpolate between them
- Shortest path between two points on sphere
 - Geodesic, on Great Circle



Spherical linear interpolation (“slerp”)



$$\alpha + \beta = \psi$$

$$\mathbf{v}(t) = w_0 \mathbf{v}_0 + w_1 \mathbf{v}_1$$

$$\frac{\sin \alpha}{w_1} = \frac{\sin \beta}{w_0} = \frac{\sin(\pi - \psi)}{1} = \sin \psi$$

$$w_0 = \sin \beta / \sin \psi$$

$$w_1 = \sin \alpha / \sin \psi$$

$$\psi = \cos^{-1}(\mathbf{v}_0 \cdot \mathbf{v}_1)$$

Quaternion Interpolation

- Spherical linear interpolation (SLERP) naturally works in any dimension
- Traverses a great arc on the sphere of unit quaternions
 - Uniform angular rotation velocity about a fixed axis

$$\psi = \cos^{-1}(q_0 \cdot q_1)$$
$$q(t) = \frac{q_0 \sin(1-t)\psi + q_1 \sin t\psi}{\sin \psi}$$

Practical issues

- When the angle between q_1 and q_2 gets close to zero, use small angle approximation
 - degenerate to linear interpolation
- When angle close to 180, there is no shortest geodesic, but can pick one
- q is same rotation as $-q$
 - if the angle between q_1 and $q_2 < \pi/2$, slerp between them;
 - else, slerp between q_1 and q_2 .

Uniform Sampling in $SO(3)$

- To uniformly sample a rotation from $SO(3)$ is the same to uniformly sample a unit quaternion from $S(3)$
 - Why? Because the distance between two rotations is linear to the on-sphere distance between the two corresponding quaternions.
- How to uniformly sample $S(3)$?
 - The simplest way is to random sample a variable from $\mathcal{N}(0, I_{4 \times 4})$ and normalize it to unit length.
 - Why? Because standard normal distribution is isotropic.
 - Then convert this quaternion to a rotation if you want.

Also refer to <https://niepp.github.io/2021/12/09/uniform-sampling-on-sphere.html> for analytical solution

Resources

- In Python, you could use the transforms3d library
- For differentiable transformations, you can play with is “Kornia”, but use with cautious to its numerical properties
- “ceres” is a C++ library that is quite useful

Why Quaternions?

- Very useful and popular in practice
 - Fast, few operations, not redundant (4D compact parameterization, and efficient to compute)
 - Numerically stable for incremental changes
 - Composes rotations nicely
- Convert to matrices at the end

Summary of Rotation Representations

	Inverse?	Composing?	Any local movement in $SO(3)$ can be achieved by local movement in the domain?
Rotation Matrix	✓	✓	N/A
Euler Angle	Complicated	Complicated	No
Angle-axis	✓	Complicated	?
Quaternion	✓	✓	✓

? means no singularity with single exceptions

Summary of Rotation Representations

- It is quite often that, we use
 - rotation matrices to define concepts
 - Euler angles to visualize rotations
 - angle-axis representation to visualize rotations and calculate derivatives
 - quaternion to write fast codes

Refer to CVPR'19 paper, [On the Continuity of Rotation Representations](#), and a recent TPAMI paper from my lab, [Towards Robust Probabilistic Modeling on \$SO\(3\)\$ via Rotation Laplace Distribution](#), to know more about rotation representations for learning purposes.

Summary of Rotation Representations

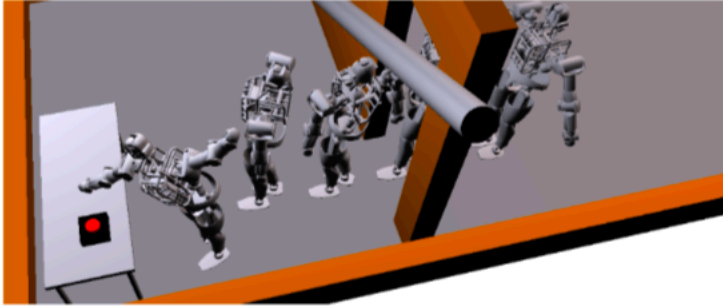
- It is quite often that, we use
 - rotation matrices to define concepts
 - Euler angles to visualize rotations
 - angle-axis representation to visualize rotations and calculate derivatives
 - quaternion to write fast codes

Motion Planning

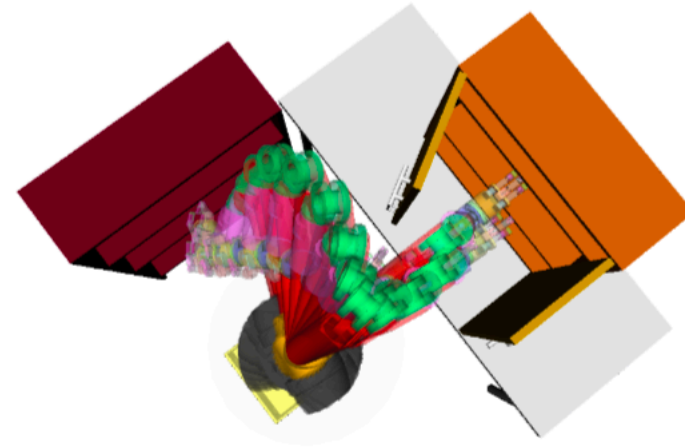
Configuration Space

- Configuration space ($\mathcal{C}$ -space) is a subset of $\mathbb{R}^n$ containing all possible states of the system (state space in RL).
- $\mathcal{C}_{free} \subseteq \mathcal{C}$ contains all valid states.
- $\mathcal{C}_{obs} \subseteq \mathcal{C}$ represents obstacles.
- Examples:
 - All valid poses of a robot.
 - All valid joint values of a robot.
 - ...

Examples



- Ratliff N, Zucker M, Bagnell J A, et al. *CHOMP: Gradient optimization techniques for efficient motion planning*, ICRA 2009
- Schulman, John, et al. *Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization*, RSS 2013



Motion Planning

- Problem:
 - Given a configuration space $\mathcal{C}_{free}$
 - Given start state q_{start} and goal state q_{goal} in $\mathcal{C}_{free}$
 - Calculate a sequence of actions that leads from start to goal
- Challenge:
 - Need to avoid obstacles
 - Long planning horizon
 - High-dimensional planning space

Motion Planning

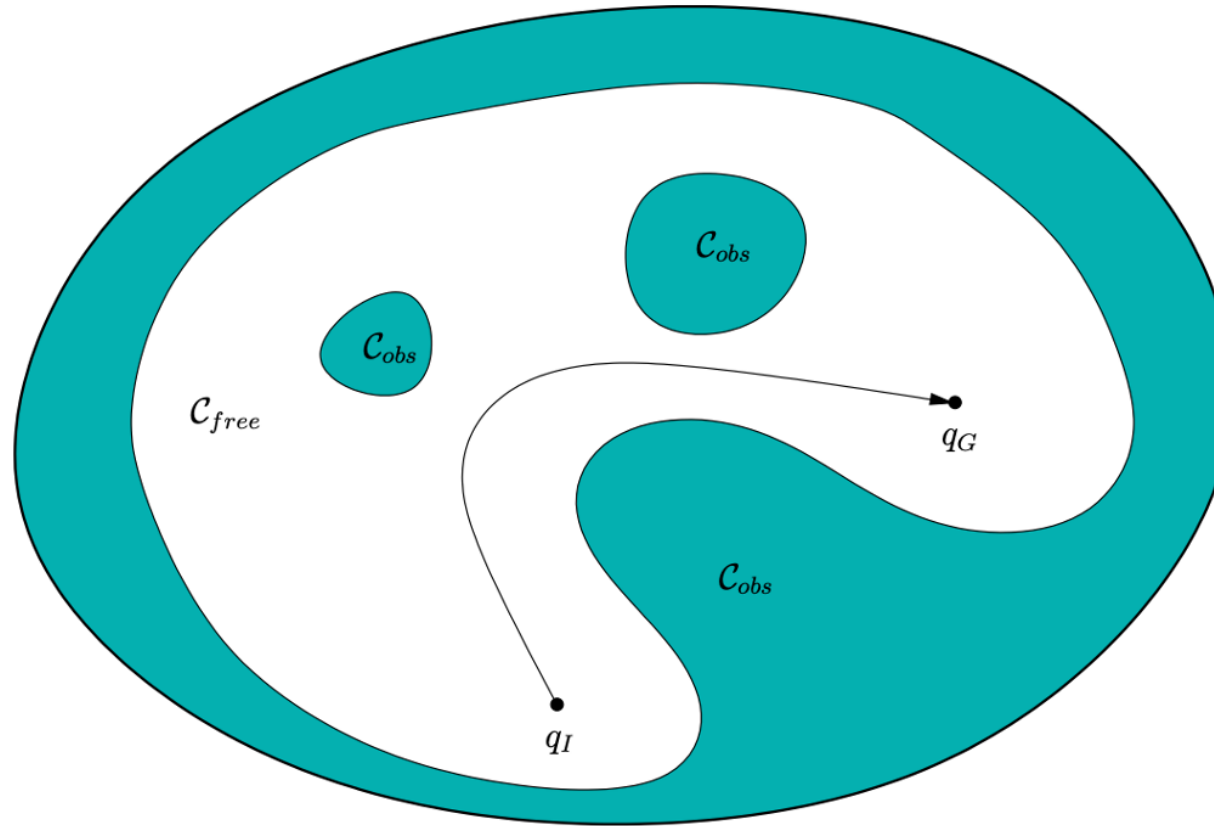
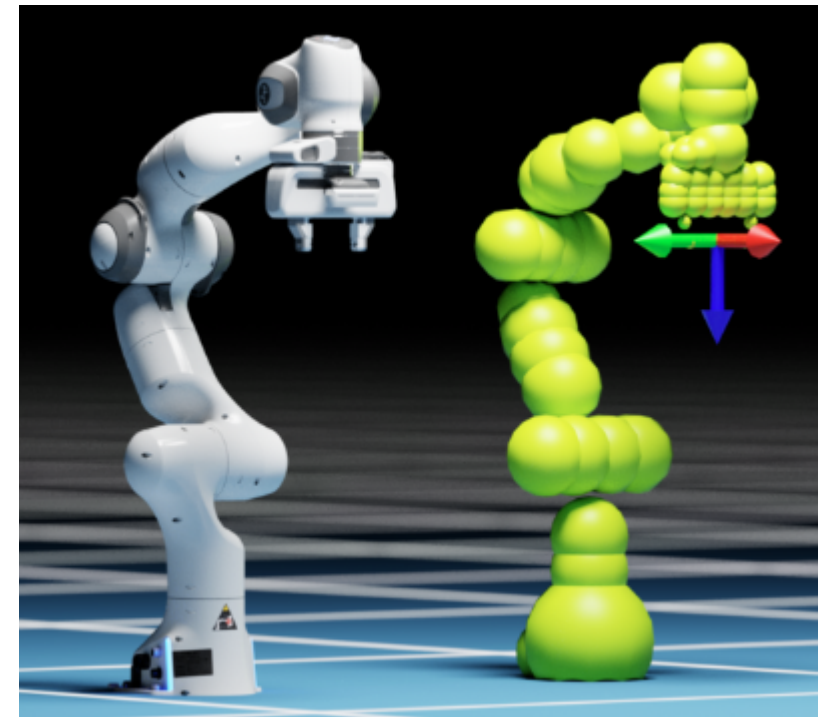


Figure 4.11: The basic motion planning problem is conceptually very simple using C-space ideas. The task is to find a path from q_I to q_G in $\mathcal{C}_{free}$. The entire blob represents $\mathcal{C} = \mathcal{C}_{free} \cup \mathcal{C}_{obs}$.

Collision modeling of robot

- Collision mesh of each link is defined in URDF
- Use forward-kinematics to get robot collision mesh from q
- Run collision check to see if q is in C_{free}



Robot representation in cuRobo



Robot Vision and Learning

– From a Perspective of Embodied AI

Next week: Lecture 4,
Robotics III

