Basic

SLAM Tutorials for Everyone
Lec. #7. Rotation, and Transformation Matrix

Hyungtae Lim, Ph.D.

<shapelim [at] mit [dot] edu> / <fudxo5143+slam [at] gmail [dot] com>

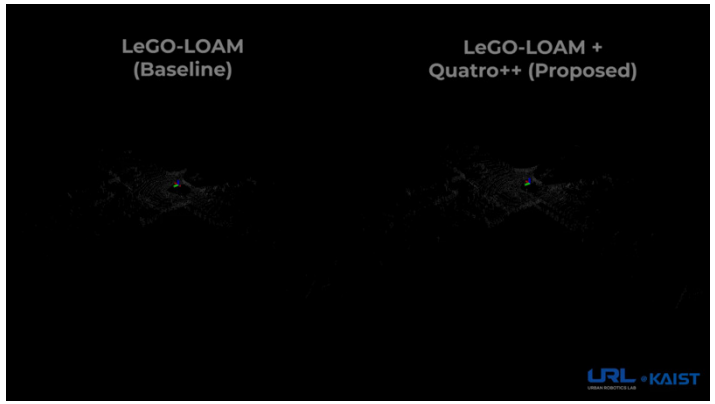
ORORA lab
Open Robotics and Resilient AI

What We've Learned so far

$$\mathcal{X}^{\text{MAP}} = \underset{\mathbf{X}\in\mathcal{X}}{\operatorname{argmin}} \sum_i \left\| h\left(\mathbf{X}_i, \mathbf{X}_{i+1}\right) \boxminus \mathbf{z}_i^{\text{odom}} \right\|^2_{\Sigma_{\text{odom}_i}} + \sum_{(j,k)\in\mathcal{I}} \left\| s_{jk}\left(h\left(\mathbf{X}_j, \mathbf{X}_k\right) \boxminus \mathbf{z}_{jk}^{\text{loop}}\right) \right\|^2_{\Sigma_{\text{loop}_{jk}}}$$

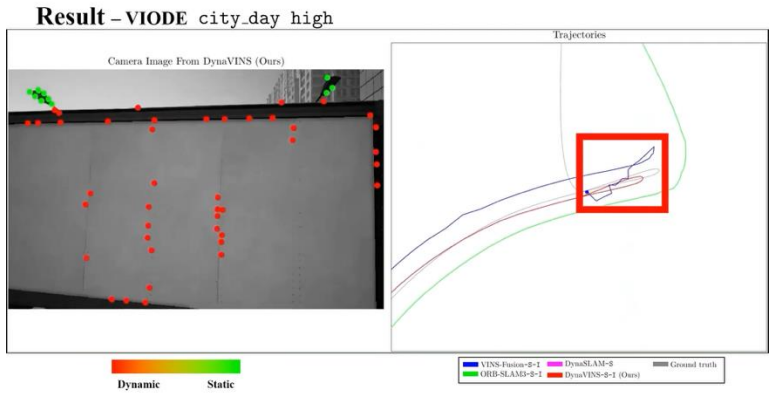
- *SLAM* is a *MAP* problem
  - which solves *weighted least squares*
  - to *estimate* the *state* that best explains all *measurements* and *prior* knowledge.

LiDAR



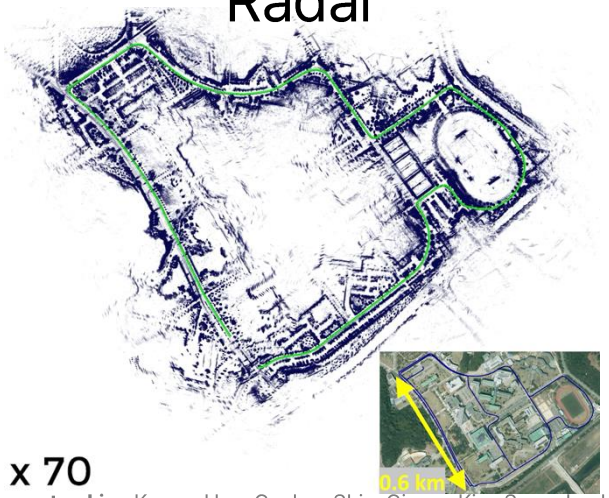
- **Hyungtae Lim***, Daebeom Kim*, and Hyun Myung, Multi-Mapcher, *IEEE T-IV, 2025.*
- **Hyungtae Lim** *et al.*, KISS-Matcher, *IEEE ICRA*, 2025
- **Hyungtae Lim** *et al.*, Quatro++, *IJRR*, 2023.
- **Hyungtae Lim** ***et al.***, A Single Correspondence Is Enough, IEEE ICRA, 2022.

Camera



- Seungwon Song, **Hyungtae Lim**, Alex Junho Lee, and Hyun Myung, DynaVINS++, *IEEE RA-L*, 2024.
- Seungwon Song, **Hyungtae Lim**, Alex Junho Lee, and Hyun Myung, DynaVINS, *IEEE RA-L*, 2022.

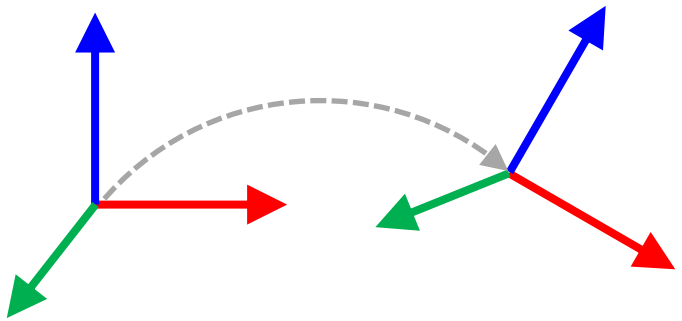
Radar



- **Hyungtae Lim**, Kawon Han, Gunhee Shin, Giseop Kim, Songcheol Hong, and Hyun Myung, ORORA, *IEEE ICRA*, 2023

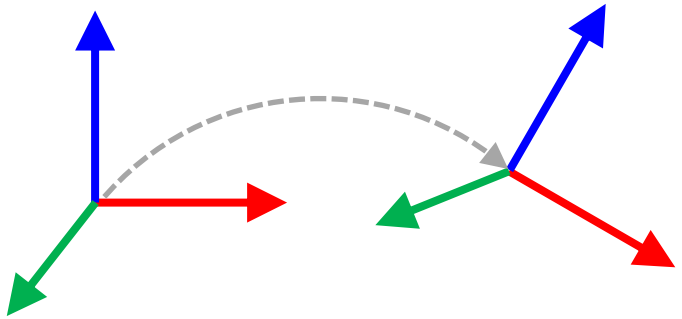
What's Left?

Q1. How to express the Pose in 3D?

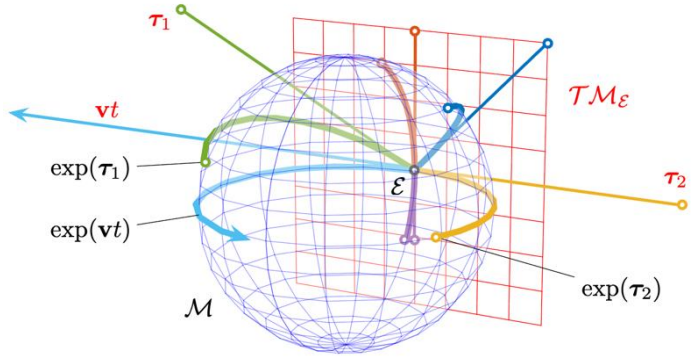


What's Left? (Cont'd)

Q1. How to express the Pose in 3D?

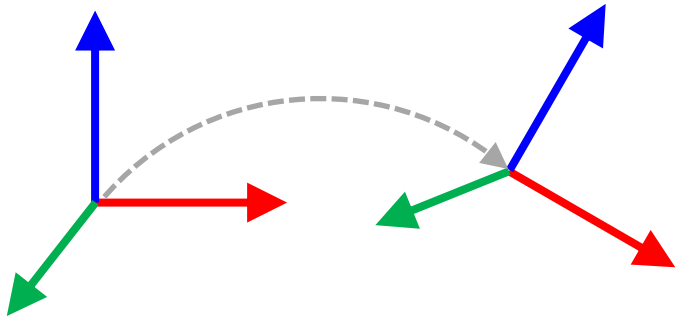


Q2. What is Lie Group & Lie Algebra?

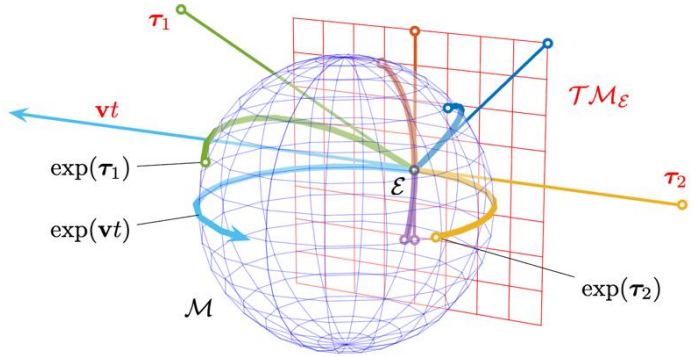


What's Left? (Cont'd)

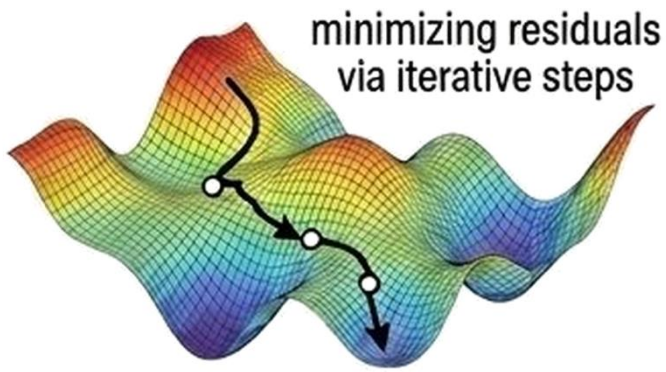
Q1. How to express the Pose in 3D?



Q2. What is Lie Group & Lie Algebra?

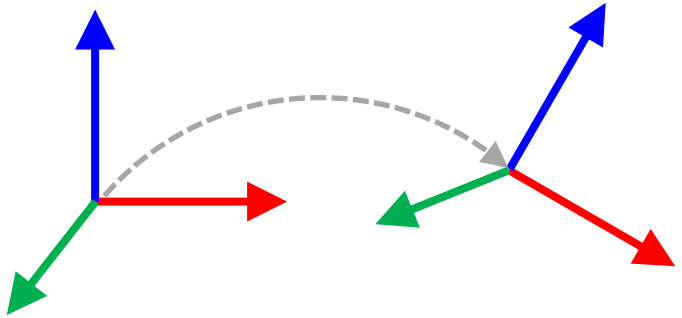


Q3. How to optimize?

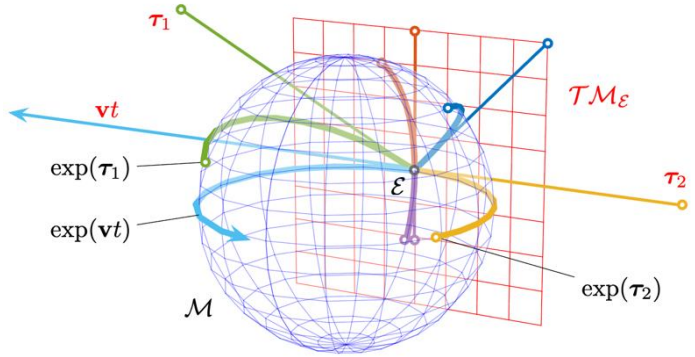


What's Left? (Cont'd)

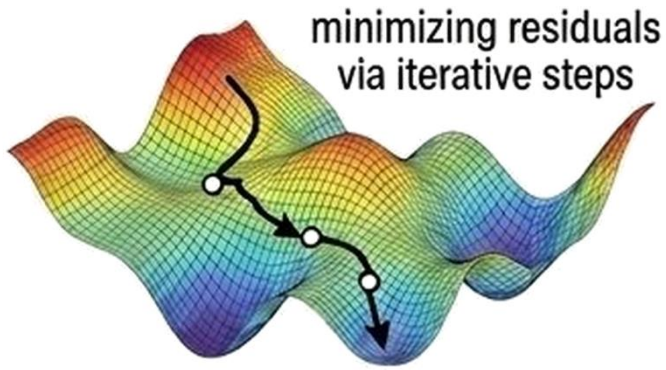
Q1. How to express the Pose in 3D?



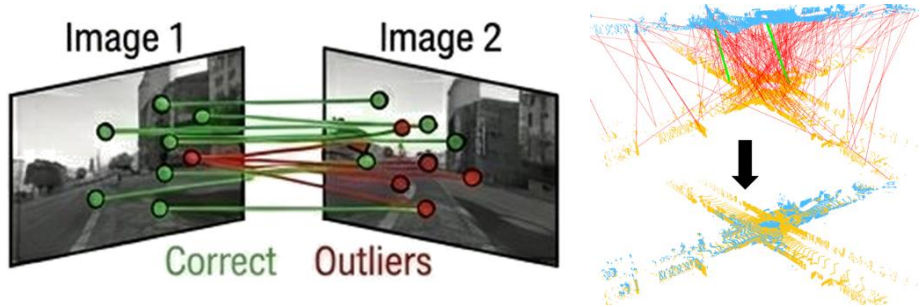
Q2. What is Lie Group & Lie Algebra?



Q3. How to optimize?

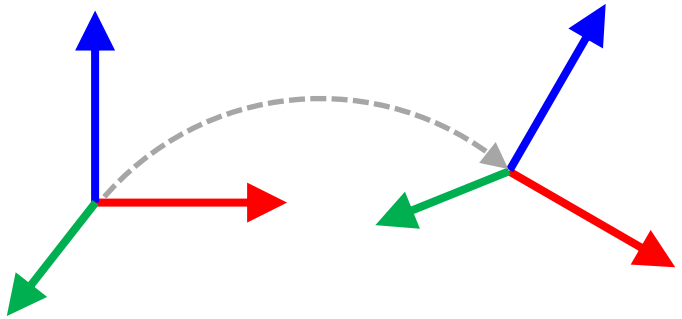


Q4. How to perform data association?



What's Left? (Cont'd)

Q1. How to express the Pose in 3D?



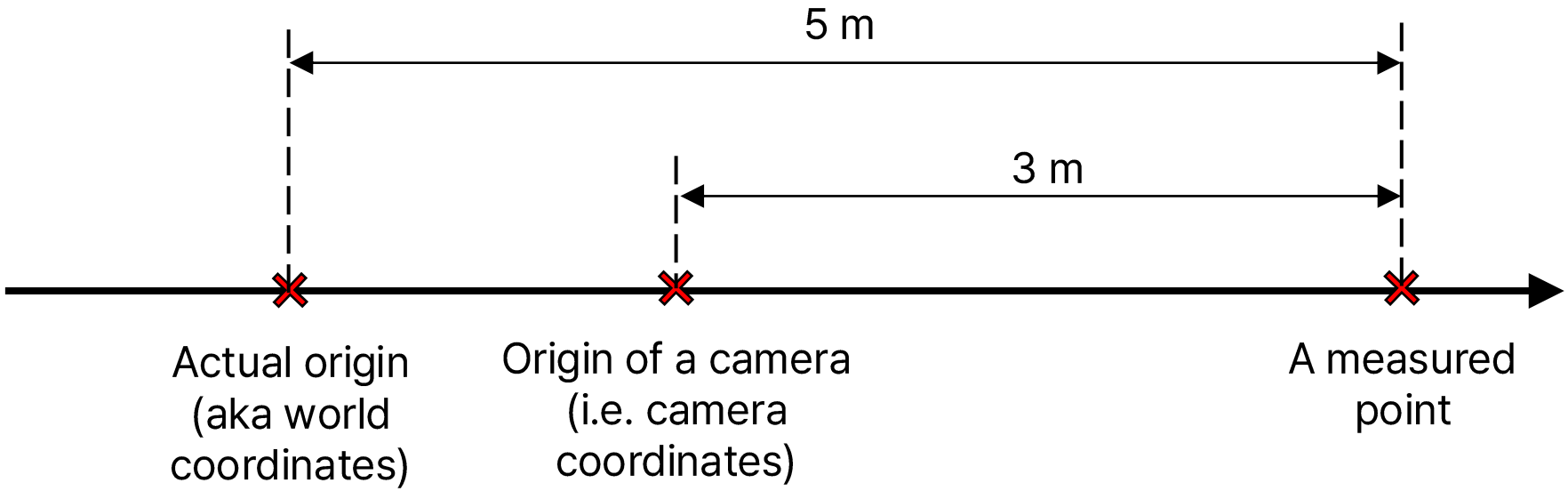
Q2. What is Lie Group & Lie Algebra?

Q3. How to optimize?

Q4. How to perform data association?

Relative Pose Transformation In 1D: Simple Addition or Subtraction

- For easy understanding, let's think about it in 1D space



- What if a point is described w.r.t. the world coordinates, i.e. 5m?
  - Just subtract (5 – 3) from 5
    - Here, (5 – 3) = 2 is the external effect due to the pose discrepancy

Pose Representation: The Coupling of *Translation* and *Rotation*

- Every rigid-body motion decomposes into exactly two operations

Translation by t

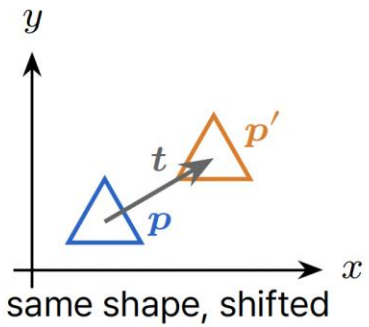
$$p' = p + t$$

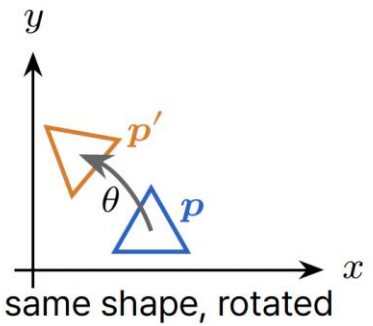
- Shifts every point by the same vector t.
- **Preserves orientation**: directions do not rotate.
- Fully described by $t \in \mathbb{R}^n$.

Rotation by R

$$p' = Rp$$

- Rotates every point around the origin.
- **Preserves distances** from the origin.
- Various forms of rotation representations exist





Representing Rotation: Four Perspectives

- Rotation matrix is *one* representation. Let us map all major options

	Angle	Angle-axis	Quaternion	Rot. Matrix $SO(n)$
2D	$\theta \in \mathbb{R}$	axis fixed: $\theta \in \mathbb{R}$	$z = e^{i\theta} \in \mathbb{C}$	$\boldsymbol{R}(\theta) \in SO(2)$
3D	(ϕ, θ, ψ) (RPY)	$\theta\hat{\boldsymbol{a}},\ \hat{\boldsymbol{a}} \in \mathbb{R}^3$	$\boldsymbol{q} \in \mathbb{H},\ \|\boldsymbol{q}\|=1$	$\boldsymbol{R} \in SO(3)$

- All four encode the same rotation
  - they differ in compactness, singularities, and computational properties.

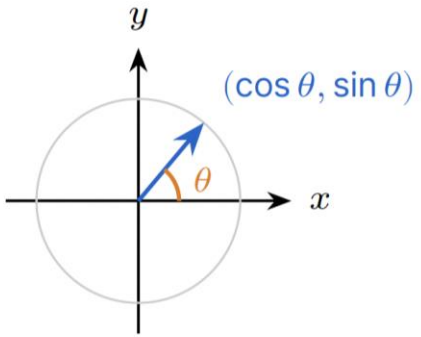
Representation 1: Angle (Euler Angles)

2D — Yaw angle θ

- A single scalar $\theta \in \mathbb{R}$ fully describes 2D rotation.
- Rotation matrix follows directly:

$$\boldsymbol{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

- Composition: simply add angles, $\theta_1 + \theta_2$.



3D — Roll-Pitch-Yaw (ϕ, θ, ψ)

- Three Euler angles, each around one axis:

$$\boldsymbol{R} = \boldsymbol{R}_z(\psi)\,\boldsymbol{R}_y(\theta)\,\boldsymbol{R}_x(\phi)$$

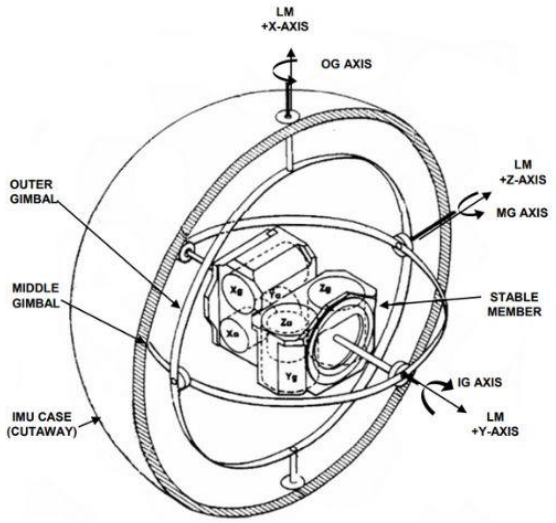
- **Intuitive** — matches everyday sense of orientation.
- Easy for humans to read; hard for machines in general cases.

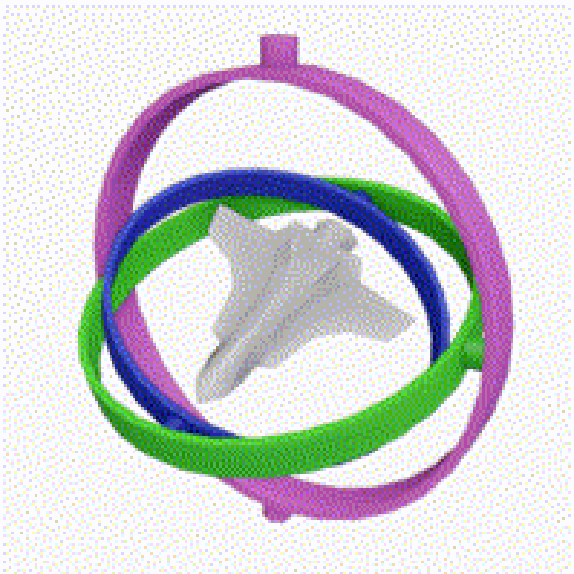
Gimbal Lock (3D only)

When pitch $\theta = \pm 90°$, roll and yaw axes align — one degree of freedom is irreversibly lost. Interpolation and averaging also break for large rotations.

Q. Does Gimbal Lock Actually Happen?

- Gimbal lock occurs when pitch reaches ±90°
  - causing roll and yaw to become indistinguishable
  - Real case: Apollo 11 (1969)
- In case of ground robots (e.g., Curiosity rover from NASA)
  - pitch = ±90° is physically impossible → Euler angles are safe in practice
- The problem is not Euler angles themselves, but whether your configuration space can reach the singularity





https://apollo11space.com/how-did-apollos-inertial-navigation-system-work/

Representation 2: Angle-Axis

2D — Fixed axis, variable angle

- In 2D the rotation axis is always the out-of-plane z-axis ($\hat{\boldsymbol{a}} = [0, 0, 1]^\top$), so only the angle θ varies.
- Reduces to the angle representation: the axis is trivially fixed.
- Nothing new in 2D, but the concept generalizes cleanly to 3D.

2D: 1 parameter (θ)
$\hat{\boldsymbol{a}} = [0, 0, 1]^\top$ is fixed — no choice of axis.

3D — General axis $\hat{a}$ and angle θ

- Any 3D rotation = angle θ around unit axis $\hat{\boldsymbol{a}} \in \mathbb{R}^3$.
- Stored as a single 3D vector (3 DOF, **minimal**):

$$\boldsymbol{\phi} = \theta\, \hat{\boldsymbol{a}} \in \mathbb{R}^3, \qquad \theta = \|\boldsymbol{\phi}\|$$

- To rotation matrix via Rodrigues' formula:

$$\boldsymbol{R} = \exp([\boldsymbol{\phi}]_\times)$$

- $\boldsymbol{\phi}$ is exactly the **Lie algebra** $\mathfrak{so}(3)$ element — the basis for manifold optimization (covered later).

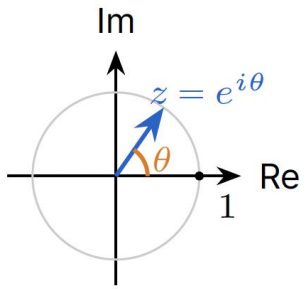
Representation 3: Complex Numbers and Quaternions

2D — Unit complex number

- Rotation by $\theta \leftrightarrow$ multiplication by $z = e^{i\theta}$:

$$z = \cos\theta + i\sin\theta, \quad |z| = 1$$

- Rotate point $p = x + iy$: $p' = z \cdot p$.
- Composition: $z_1 z_2 = e^{i(\theta_1+\theta_2)}$ — elegant!



3D — Unit quaternion q

- Extends complex:
  $\boldsymbol{q} = q_w + q_x i + q_y j + q_z k$, $\|\boldsymbol{q}\| = 1$.
- Encodes axis-angle $(\hat{\boldsymbol{a}}, \theta)$ as:

$$\boldsymbol{q} = \left(\cos\tfrac{\theta}{2},\ \sin\tfrac{\theta}{2}\,\hat{\boldsymbol{a}}\right)$$

- Rotation: $\boldsymbol{p}' = \boldsymbol{q}\,\boldsymbol{p}\,\boldsymbol{q}^*$.

Trade-offs

+ Faster multiply; easy SLERP interpolation.
− Double cover: $\boldsymbol{q} \equiv -\boldsymbol{q}$.
− Constrained optimization ($\|\boldsymbol{q}\| = 1$).

Representation 4: Rotation Matrix SO(n)

2D — $SO(2)$

$$\boldsymbol{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \in \mathbb{R}^{2\times 2}$$

- Constraints: $\boldsymbol{R}^\top \boldsymbol{R} = \boldsymbol{I}$, $\det(\boldsymbol{R}) = 1$.
- 4 entries, **1 DOF** (3 orthonormality constraints).
- Composition: $\boldsymbol{R}_1 \boldsymbol{R}_2 \in SO(2)$.
- Inverse: $\boldsymbol{R}^{-1} = \boldsymbol{R}^\top$.

3D — $SO(3)$

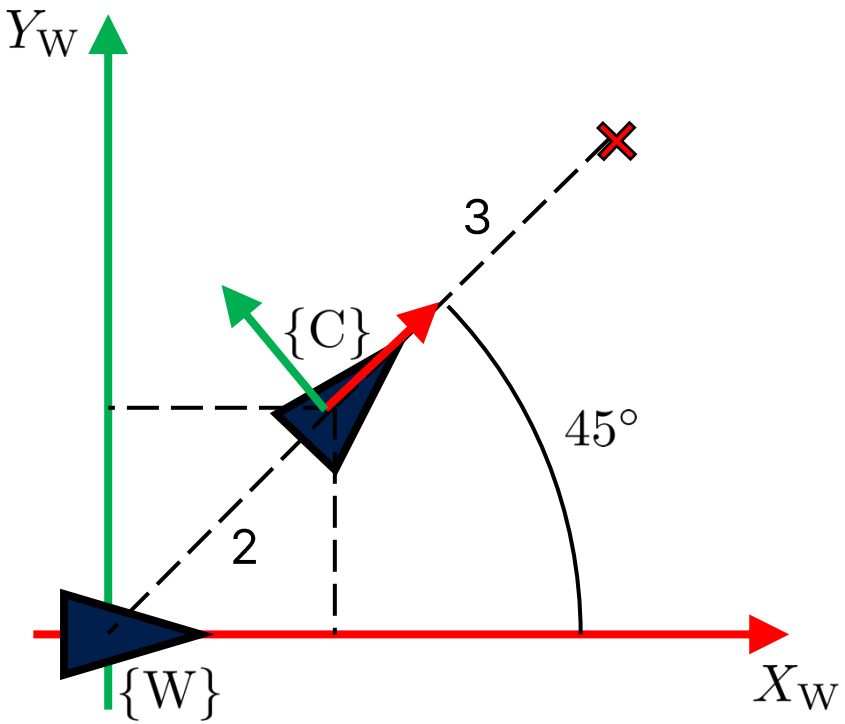
$$\boldsymbol{R} \in \mathbb{R}^{3\times 3}, \quad \boldsymbol{R}^\top \boldsymbol{R} = \boldsymbol{I}, \quad \det(\boldsymbol{R}) = 1$$

- 9 entries, **3 DOF** (6 orthonormality constraints).
- Same closure and inverse structure as $SO(2)$.

Why SLAM prefers $SO(3)$

- Jacobians have **closed analytic forms**.
- Covariance (uncertainty) expressed naturally.
- Familiar linear-algebra operations.

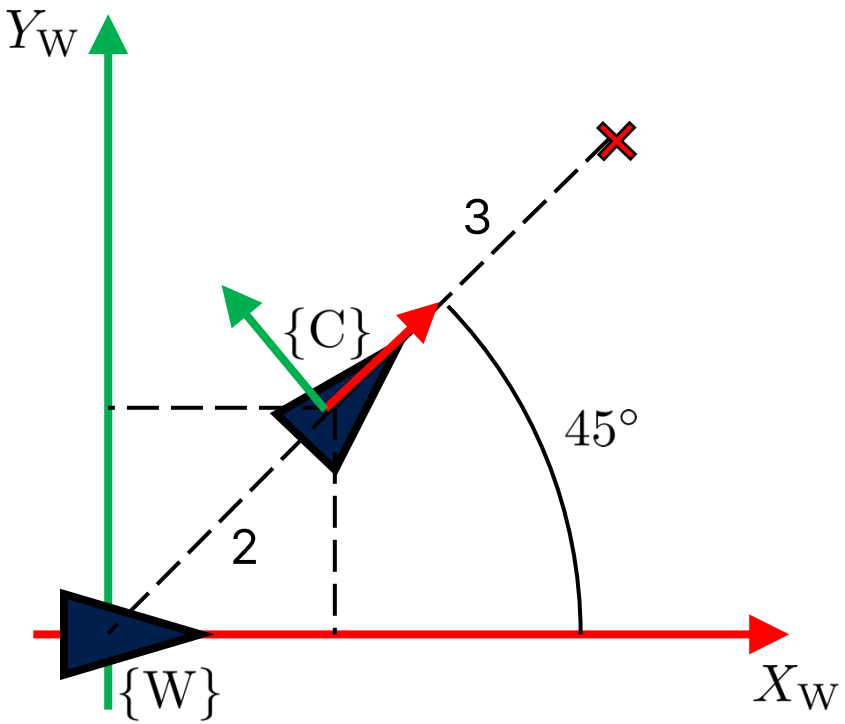
An Example of Transformation In 2D



$$\mathbf{p}_W = (\frac{5\sqrt{2}}{2}, \frac{5\sqrt{2}}{2})$$

$$\mathbf{p}_C = (3, 0)$$

An Example of Transformation In 2D (Cont'd)



$$\mathbf{p}_W = (\frac{5\sqrt{2}}{2}, \frac{5\sqrt{2}}{2})$$

$$\mathbf{p}_C = (3, 0)$$

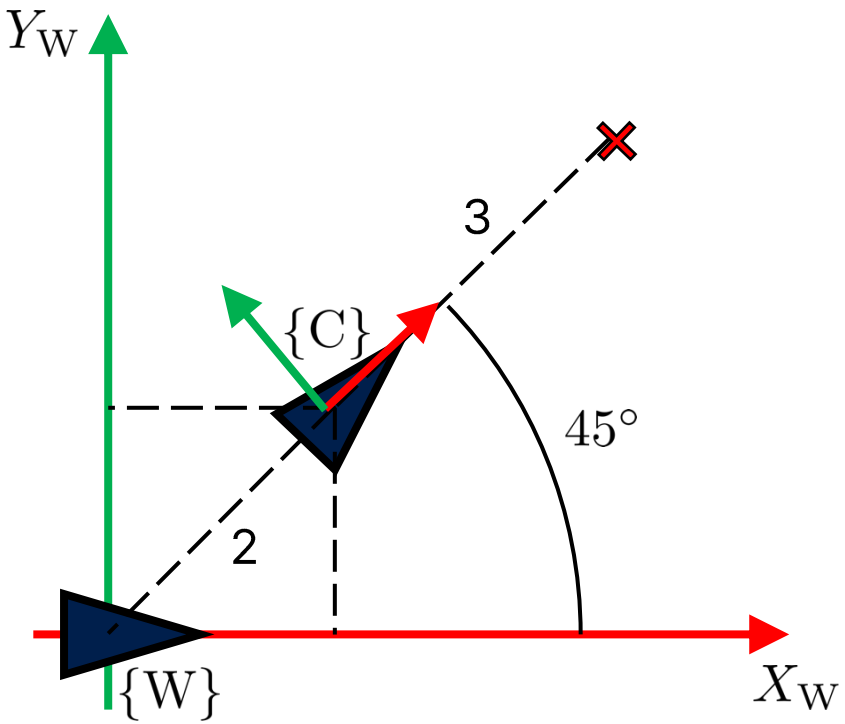
1. Rotate $\mathbf{p}_W$ by -45 degrees
2. Translate the rotated one by (-2, 0)

Q. Why -45 degrees, not +45 degrees?

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

- From the viewpoint of a point:
  - Counter-clockwise
- From the viewpoint of the coordinate:
  - Clockwise

An Example of Transformation In 2D (Cont'd)



$$\mathbf{p}_W = (\frac{5\sqrt{2}}{2}, \frac{5\sqrt{2}}{2})$$

$$\mathbf{p}_C = (3, 0)$$

1. Rotate $\mathbf{p}_W$ by -45 degrees
2. Translate the rotated one by (-2, 0)

In a matrix form,

Rotation matrix Translation vector

$$\begin{bmatrix} \cos(-45°) & -\sin(-45°) & -2 \\ \sin(-45°) & \cos(-45°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{5\sqrt{2}}{2} \\ \frac{5\sqrt{2}}{2} \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 1 \end{bmatrix}$$

2D Rigid Body Motion: Rotation Then Translation

A 2D pose combines rotation and translation:

$$p' = R(\theta)\, p + t, \qquad R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \quad t \in \mathbb{R}^2.$$

In **homogeneous coordinates**, this becomes a single matrix multiply:

$$\begin{bmatrix} p' \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} R(\theta) & t \\ 0^\top & 1 \end{bmatrix}}_{T \in SE(2)} \begin{bmatrix} p \\ 1 \end{bmatrix}.$$

Composing two motions $T_1 \cdot T_2$:

$$T_1 T_2 = \begin{bmatrix} R_1 & t_1 \\ 0^\top & 1 \end{bmatrix} \begin{bmatrix} R_2 & t_2 \\ 0^\top & 1 \end{bmatrix} = \begin{bmatrix} R_1 R_2 & R_1 t_2 + t_1 \\ 0^\top & 1 \end{bmatrix}.$$

Rotation is unaffected by translation

$$R_{\text{combined}} = R_1 R_2$$

No t_1 or t_2 appears here.

Translation depends on rotation

$$t_{\text{combined}} = R_1 t_2 + t_1$$

t_2 is rotated by R_1 before adding!

Generalization: Rigid Body Motion Group SE(3)

- A full 3D pose (position + orientation) is represented as:

Definition

$$SE(3) = \left\{ \boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^\top & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \;\middle|\; \boldsymbol{R} \in SO(3),\ \boldsymbol{t} \in \mathbb{R}^3 \right\}$$

6 degrees of freedom: 3 for rotation + 3 for translation.

Rigid body transformation of a point $\boldsymbol{p} \in \mathbb{R}^3$:

$$\boldsymbol{p}' = \boldsymbol{R}\boldsymbol{p} + \boldsymbol{t} \qquad \Longleftrightarrow \qquad \begin{bmatrix} \boldsymbol{p}' \\ 1 \end{bmatrix} = \boldsymbol{T} \begin{bmatrix} \boldsymbol{p} \\ 1 \end{bmatrix}$$

Composition of transforms (e.g., sensor → body → world):

$$^{\text{world}}\boldsymbol{T}_{\text{sensor}} = {}^{\text{world}}\boldsymbol{T}_{\text{body}} \cdot {}^{\text{body}}\boldsymbol{T}_{\text{sensor}}$$

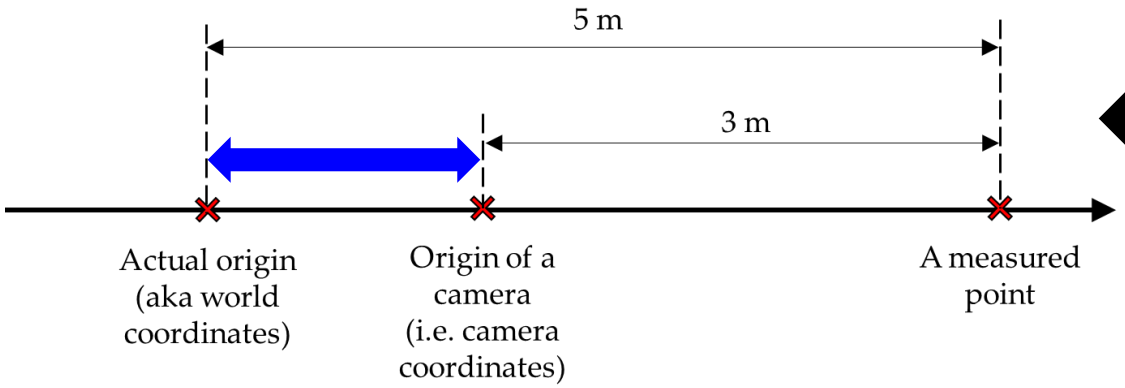
Inverse:

$$\boldsymbol{T}^{-1} = \begin{bmatrix} \boldsymbol{R}^\top & -\boldsymbol{R}^\top \boldsymbol{t} \\ \boldsymbol{0}^\top & 1 \end{bmatrix}$$

Summary of Transformation Matrix

- Don't be afraid!
  - The only thing that has changed is ± in 1D has become a *multiplication*
    - To express the rotation and translation simultaneously

In 1D case



In 2D/3D case

$$\mathbf{T}_{\mathrm{W}}^{\mathrm{C}} = \begin{bmatrix} \mathbf{R}_{n\times n} & \mathbf{t}_n \\ \mathbf{0}_{1\times n} & 1 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}$$

$$\mathbf{T}_{\mathrm{W}}^{\mathrm{C}} \cdot \mathbf{p}_{\mathrm{W}} \rightarrow \mathbf{p}_{\mathrm{C}} \text{ where } \mathbf{p}_{\mathrm{C}} = [x\ y\ z\ 1]^{\intercal}$$

Conclusion

- Every rigid-body motion = rotation + translation
- Four rotation representations — same rotation, different trade-offs

	Angle	Angle-axis	Quaternion	Rot. Matrix $SO(n)$
2D	$\theta \in \mathbb{R}$	axis fixed: $\theta \in \mathbb{R}$	$z = e^{i\theta} \in \mathbb{C}$	$\boldsymbol{R}(\theta) \in SO(2)$
3D	(ϕ, θ, ψ) (RPY)	$\theta\hat{\boldsymbol{a}},\ \hat{\boldsymbol{a}} \in \mathbb{R}^3$	$\boldsymbol{q} \in \mathbb{H},\ \|\boldsymbol{q}\|=1$	$\boldsymbol{R} \in SO(3)$

- Full pose = SE(3) transformation matrix
  - Unifies rotation and translation into a single 4×4 matrix $\boldsymbol{T} \in SE(3)$

```
73        // Compute initial_guess for ICP
74        const auto prediction = GetPredictionModel();
75        const auto last_pose = !poses_.empty() ? poses_.back() : Sophus::SE3d();
76        const auto initial_guess = last_pose * prediction;
```

Now, we know the meaning of `last_pose * prediction`!

https://github.com/cocel-postech/genz-icp/blob/03cf3e59f0ab3b5cc4e29e7a038b1ab04da24da7/cpp/genz_icp/pipeline/GenZICP.cpp#L73