Rotation representation

Bang-Shien Chen*

Rotations are fundamental for representing the *orientation* of a pose in robotics. To be more specific, a pose (or coordinate system) $T \in \mathrm{SE}(3)$ consists of a rotation matrix $R \in \mathrm{SO}(3)$ and a translation vector $t \in \mathbb{R}^3$, while R defines the axis of a coordinate system and t specifies the origin of the coordinate system. In other words, each object/robot/camera has its own coordinate system. In this note, we will introduce different rotation representations (since translation is rather simple), and discuss about how to manipulate them. For simplicity, we assume that all coordinate systems have the same origin.

1 Matrix Representation

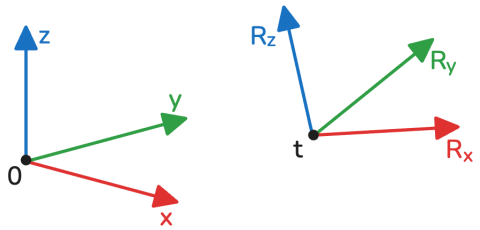

Figure 1.1: Poses or coordinate systems.

We begin with the most intuitive representation of rotations: rotation matrices. Each column of a rotation matrix represents the orthogonal unit-length axis that satisfy the right-hand rule. For example, the rotation matrix in Figure 1.1 is

$$R = \begin{pmatrix} | & | & | \\ R_x & R_y & R_z \\ | & | & | \end{pmatrix}.$$

Note that in Figure 1.1, the (right) pose has a reference (left) coordinate system, where we typically call it the world coordinate system. To distinguish between different coordinate systems, we denote the reference coordinate system as w, the current robot pose as r, and add a superscript for reference coordinate system and subscript for current coordinate system, i.e., R_r^w. Since the columns have unit length, are orthogonal to each other, and satisfy the right-hand rule, we have

$$R_r^w \in \mathrm{SO}(3) = \{R \in \mathbb{R}^{3\times 3} \mid R^\top R = I_3,\ \det(R) = +1\}. \tag{1.1}$$

1.1 Rotation matrix operations

Suppose that a robot observes a point p^r, which is represented by the robot's coordinate system (the robot does not have knowledge about the world coordinate system). To convert this point to the world coordinate system, we apply the rotation matrix:

$$p^w = R_r^w p^r. \tag{1.2}$$

*https://dgbshien.com/

This is where the superscripts and subscripts come in handy. We will discuss about transferring a point's reference coordinate system with a more concrete example in Section 5.3. Now consider the case that we have two coordinate systems, R_r^w and R_c^r, we can compute a pose c with reference w as follows:

$$R_c^w = R_r^w R_c^r. \tag{1.3}$$

Again, we will discuss about transferring coordinate systems with a more concrete example in Section 5.2. Now given a rotation R_r^w, since $R_r^w R_w^r = R_w^w = I_3$, we have the inverse

$$R_w^r = (R_r^w)^{-1} = (R_r^w)^\top. \tag{1.4}$$

This also suggests that the rows of R_r^w is the axis of coordinate system w with reference coordinate system r. Therefore, we could also extend the special orthogonal group in Equation (1.1) as $\mathrm{SO}(3) = \{R \in \mathbb{R}^{3\times 3} \mid R^\top R = RR^\top = I_3,\ \det(r) = 1\}$. Also, the special orthogonal group is not commutative since $R_r^w R_c^r \neq R_c^r R_r^w$ in general.

2 Euler angle representation

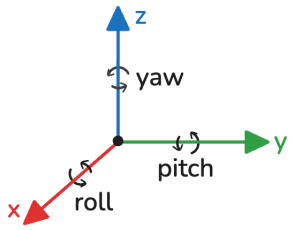

Figure 2.1: Raw, pitch, and yaw.

While rotation matrices are excellent to perform calculations due to its group structure, it clearly stores redundant information since a 3D rotation has only 3 degree of freedom (DoF). Euler's rotation theorem states that any 3D rotation can be described using just three parameters. However, Euler angles are not uniquely defined, as different Euler decomposition conventions exist [3]. A common choice is the *raw-pitch-yaw* angle representation given (α, β, γ), which represents the rotated angle around x-axis,y-axis, and z-axis, respectively:

$$\begin{aligned} R_r^w &= \underbrace{\begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{R_z(\gamma)} \underbrace{\begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{pmatrix}}_{R_y(\beta)} \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix}}_{R_x(\alpha)} \\ &= \begin{pmatrix} \cos(\beta)\cos(\gamma) & \sin(\alpha)\sin(\beta)\cos(\alpha) - \cos(\alpha)\sin(\gamma) & \cos(\alpha)\sin(\beta)\cos(\gamma) + \sin(\alpha)\sin(\gamma) \\ \cos(\beta)\sin(\gamma) & \sin(\alpha)\sin(\beta)\sin(\gamma) + \cos(\alpha)\cos(\gamma) & \cos(\alpha)\sin(\beta)\sin(\gamma) - \sin(\alpha)\cos(\gamma) \\ -\sin(\beta) & \sin(\alpha)\cos(\beta) & \cos(\alpha)\cos(\beta) \end{pmatrix}. \end{aligned} \tag{2.1}$$

Euler angles provide a minimal representation and intuitive way to describe rotations. However, the operations of angle representations typically involve trigonometric functions, which may be slower to compute. In addition, Euler angles suffers from *singularity/gimbal lock*, where one DoF is lost in certain angles. That is, certain rotations do not have a unique Euler angle representation. This also leads to different methods of computing Euler angles from rotation matrices [3, 6]. For practical implementation (as well as conversion between all other rotation representations), I would recommend using `SciPy`'s Rotation class if you are working with Python, `Eigen`'s EulerAngles class if you are working with C++, or `rotm2eul` (built-in function) if you are working with Matlab.

3 Axis-angle representation

Another common representation for rotations is the axis-angle representation (u, θ), parameterized by a unit vector u and a rotation angle θ that rotates (right-hand rule) around axis u. Let $\boldsymbol{\theta} = \theta u \in \mathbb{R}^3$, we could also store axis-angle representations in 3 parameters. In addition, axis-angle representations are isomorphic to $\mathfrak{so}(3)$, the Lie algebra of SO(3), which is convenient to use while dealing with Lie derivatives. We will discuss about Lie theory in a separate note.

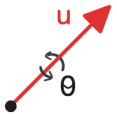

Figure 3.1: Axis-angle representation.

There are also some drawbacks of axis-angle representations. Computations can be slow because operations typically involve trigonometric functions; the inverse, however, is easy to compute by $(u, \theta)^{-1} = (-u, \theta) = (u, -\theta)$. Also, axis-angle representations are not unique, e.g., $(u, \theta) = (u, \theta + 2k\pi)$ for any k and $(u, \theta) = (-u, -\theta)$, which introduces discontinuities [9].

3.1 Axis-angle to rotation matrix

Given an axis-angle representation (u, θ), the corresponding rotation matrix can be computed using *Rodrigues' rotation formula*:

$$R = I_3 + \sin\theta [u]_\times + (1 - \cos\theta)[u]_\times^2, \tag{3.1}$$

where

$$[u]_\times = \begin{pmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{pmatrix} \tag{3.2}$$

is a skew-symmetric matrix. We will discuss about Equation (3.1) in more details, in a separate note about Lie theory[1].

3.2 Rotation matrix to axis-angle

Now given a rotation matrix R, we compute θ by the trace of R. Since all $R \sim R_x(\theta)$ and similar matrices have the same trace, by Equation (2.1), we have $\operatorname{tr}(R) = 1 + 2\cos\theta$. Thereby, we compute the rotation angle by

$$\theta = \arccos\left(\frac{\operatorname{tr}(R) - 1}{2}\right). \tag{3.3}$$

To obtain u, we first recall that since the rotation rotates around u, we have $Ru = u$. That is, u is the eigenvector corresponding to the eigenvalue 1. Note that the eigenvalues of a 3D rotation matrix are always $\{1, \cos\theta \pm \sin\theta\}$, we can derive a closed-form for computing the eigenvector u. Since $R - R^\top$ is skew-symmetric, we chose $\bar{u} = R - R^\top$. Nevertheless, $\bar{u}$ may not be a unit vector and by $[u]_\times^\top = -[u]_\times$, we have

$$\|\bar{u}\| = \|(I_3 - \sin\theta[u]_\times + (1 - \cos\theta)[u]_\times^2) - (I_3 + \sin\theta[u]_\times + (1 - \cos\theta)[u]_\times^2)\| = 2\sin\theta.$$

Thereby, we compute the rotation axis by

$$u = \frac{(R - R^\top)^\wedge}{2\sin\theta}, \tag{3.4}$$

where $(\cdot)^\wedge$ is the inverse of Equation (3.2).

[1] https://dgbshien.com/docs/blogs/lie-theory.pdf

4 Quaternion representation

3-parameter representations are ideal for storage but have singularities (in fact, there is no 3-parameter representations without singularity [8]), while matrix representations are singularity-free but over-parametrized. We next introduce quaternions, a representation that is singularity-free and uses less parameters than rotation matrices.

Quaternions are extended complex numbers found by William R. Hamilton. It is the most unintuitive representation among all, however, 3Blue1Brown has some nice visualization videos[2]. A 2D rotation can be represented by a unit-length complex number, using the Euler equation:

$$e^{i\theta} = \cos\theta + i\sin\theta.$$

Similarly, we will later see that a 3D rotation can be represented by a unit quaternion. A quaternion has a real part (scalar) and three imaginary parts (vector). We write the scalar part at last[3] as follows:

$$q = \begin{pmatrix} v \\ s \end{pmatrix} = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{pmatrix} = \underbrace{q_4}_{\text{real part}} + \underbrace{q_1 i + q_2 j + q_3 k}_{\text{imaginary part}}, \tag{4.1}$$

where s denotes the scalar part, v denotes the vector part, and i, j, k are the three imaginary parts that satisfy

$$\begin{cases} i^2 = j^2 = k^2 = -1, \\ ij = k,\ ji = -k, \\ jk = i,\ kj = -i, \\ ki = j,\ ik = -j. \end{cases}$$

Quaternions are a broad mathematical topic with nice structures and properties, however, we focus solely on unit quaternions and its relation with 3D rotations. The basic insight behind the quaternion is to rearrange the axis-angle representation (u, θ) as follows:

$$q = \begin{pmatrix} \sin(\theta/2)u \\ \cos(\theta/2) \end{pmatrix}.$$

This stores the essential information required to represent 3D rotations, and by Equation (3.1), we can recover a rotation matrix as follows:

$$R = \begin{pmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 - q_3q_4) & 2(q_1q_3 + q_2q_4) \\ 2(q_1q_2 + q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 - q_1q_4) \\ 2(q_1q_3 - q_2q_4) & 2(q_2q_3 + q_1q_4) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{pmatrix}. \tag{4.2}$$

Quaternion eliminates singularity in Euler angle representation and resolves discontinuity in axis-angle representation. The only ambiguity is that q and $-q$ represent the same rotation, however, the mapping is still smooth and does not introduce sudden jumps or gimbal lock. This is also why most neural networks opt to learn quaternions over other rotation representations.

4.1 Quaternion operations

Recall that in Section 1.1, we have $p^w = R_r^w p^r$ for rotation matrices. Quaternions also has a simple operation for 3D vectors:

$$\begin{pmatrix} p^w \\ 1 \end{pmatrix} = q_r^w \otimes \begin{pmatrix} p^r \\ 1 \end{pmatrix} \otimes (q_r^w)^{-1}, \tag{4.3}$$

where stacking an additional dimension with 1 is also known as the *homogeneous coordinates*.

[2] https://youtu.be/zjMuIxRvygQ

[3] The scalar part is more often written in the front, however, we adopt the unusual convention to keep some similarity with the homogeneous coordinates.

Again, we have $R_c^w = R_r^w R_c^r$ for rotation matrices. For quaternions, we define *quaternion product* and compose rotations with

$$q_c^w = q_r^w \otimes q_c^r := \Omega_1(q_r^w) q_c^r = \Omega_2(q_c^r) q_r^w, \tag{4.4}$$

where

$$\Omega_1(q) = \begin{pmatrix} q_4 & -q_3 & q_2 & q_1 \\ q_3 & q_4 & -q_1 & q_2 \\ -q_2 & q_1 & q_4 & q_3 \\ -q_1 & -q_2 & -q_3 & q_4 \end{pmatrix} \quad \text{and} \quad \Omega_2(q) = \begin{pmatrix} q_4 & q_3 & -q_2 & q_1 \\ -q_3 & q_4 & q_1 & q_2 \\ q_2 & -q_1 & q_4 & q_3 \\ -q_1 & -q_2 & -q_3 & q_4 \end{pmatrix}.$$

Since quaternions can be derived from axis-angle representations, the inverse is also straightforward:

$$q_w^r = \begin{pmatrix} -v_r^w \\ s_r^w \end{pmatrix} = (q_r^w)^{-1}, \tag{4.5}$$

where s_r^w and v_r^w is the scalar and vector part of q_r^w, respectively.

4.2 Is quaternion the best representation?

At first glance, quaternions appear to be the best choice for representing rotations—they provide a minimal parameterization without singularities and are computationally more efficient than rotation matrices. However, they come with one notable drawback when working with distance of quaternions. Given a distance function d, we typically minimize an optimization problem like $\min_q d(q, \tilde{q})$. Since both $\tilde{q}$ and $-\tilde{q}$ represent the same rotation, we would ideally want $d(q, \tilde{q}) = d(q, -\tilde{q})$. Unfortunately, $d(q, \tilde{q}) \neq d(q, -\tilde{q})$ in general, which can pose issues to the optimization problem. Hence, while being an effective representation, we will mostly prefer to use rotation matrix representation.

5 Rigid-body transformation

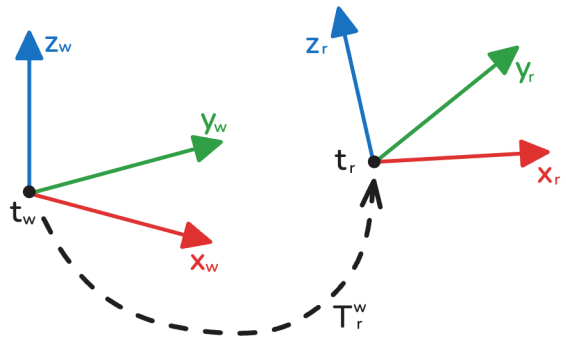

Figure 5.1: Poses or transformations.

In Section 1, we have shown that a pose or coordinate system can be represented by a rotation and a translation, while assuming the translation is 0. Now we introduce the translation vector t_r^w back, with similar superscript w denoting the reference coordinate system and subscript r denoting the current robot pose. It is convenient to assemble the rotation matrix and translation vector to a transformation matrix

$$T_r^w = \begin{pmatrix} R_r^w & t_r^w \\ 0^\top & 1 \end{pmatrix} \in \mathrm{SE}(3). \tag{5.1}$$

We next discuss about the operations of transformation matrices.

5.1 Transformation matrix operations

Similar to Section 1.1, we can transfer p_r to p_w by $p^w = R_r^w p^r + t_r^w$ or

$$\tilde{p}^w = T_r^w \tilde{p}^r, \tag{5.2}$$

where $\tilde{p} = (p^\top, 1)^\top$ is the homogeneous coordinates. For composition, we have

$$T_c^w = T_r^w T_c^r. \tag{5.3}$$

Furthermore, we have the inverse of T_r^w as follows:

$$T_w^r = (T_r^w)^{-1} = \begin{pmatrix} (R_r^w)^\top & -(R_r^w)^\top t_r^w \\ 0^\top & 1 \end{pmatrix}. \tag{5.4}$$

Proof. We verify the property of an inverse

$$T_r^w T_w^r = \begin{pmatrix} R_r^w & t_r^w \\ 0^\top & 1 \end{pmatrix} \begin{pmatrix} (R_r^w)^\top & -(R_r^w)^\top t_r^w \\ 0^\top & 1 \end{pmatrix} = \begin{pmatrix} R_r^w R_w^r & t_r^w - R_r^w R_w^r t_r^w \\ 0^\top & 1 \end{pmatrix} = I_4,$$
$$T_w^r T_r^w = \begin{pmatrix} (R_r^w)^\top & -(R_r^w)^\top t_r^w \\ 0^\top & 1 \end{pmatrix} \begin{pmatrix} R_r^w & t_r^w \\ 0^\top & 1 \end{pmatrix} = \begin{pmatrix} R_w^r R_r^w & R_w^r t_r^w - R_w^r t_r^w \\ 0^\top & 1 \end{pmatrix} = I_4.$$

□

5.2 Pose graph optimization

Here we give an example of utilizing transformation matrix composition. Suppose that a robot has a trajectory $\{T_i\}$, we call the transformation matrix that represents each robot pose T_i as *absolute pose*. On the other hand, the change of coordinate system between two robot poses T_i and T_j is also represented by a transformation matrix T_{ij}, which is called *relative pose*. By Equation (5.3), we have

$$T_j = T_i T_{ij}. \tag{5.5}$$

Pose graph optimization tries to find better absolute poses (variables) given the relative poses (constraints). A natural optimization problem arises:

$$\min_{\{T_i\}} \sum_{(i,j)\in\mathcal{E}} \sigma_{ij} \|T_j - T_i T_{ij}\|_F^2,$$

where $\mathcal{E}$ denotes the edge of the pose graph and σ_{ij} is the standard deviation of the T_{ij} measurement (in maximum likelihood sense). However, we often apply different weights to rotation and translation. Expanding Equation (5.5), we have

$$\begin{pmatrix} R_j & t_j \\ 0^\top & 1 \end{pmatrix} = \begin{pmatrix} R_i & t_i \\ 0^\top & 1 \end{pmatrix} \begin{pmatrix} R_{ij} & t_{ij} \\ 0^\top & 1 \end{pmatrix} = \begin{pmatrix} R_i R_{ij} & R_i t_{ij} + t_i \\ 0^\top & 1 \end{pmatrix}.$$

Thereby, we decouple the problem to a more commonly seen pose graph optimization [5] problem:

$$\min_{\{R_i\},\{t_i\}} \sum_{(i,j)\in\mathcal{E}} \kappa_{ij} \|R_j - R_i R_{ij}\|_F^2 + \tau_{ij} \|t_j - (R_i t_{ij} + t_i)\|_2^2.$$

Note that using the Frobenius norm to compute the distance between two rotation matrices is also known as the chordal distance. One can also use other distance metrics [2] such as *geodesic distance*, which is equivalent to extracting the angle of the axis-angle representation of the relative transformation between two rotation matrices. The corresponding optimization problem would then be

$$\min_{\{R_i\},\{t_i\}} \sum_{(i,j)\in\mathcal{E}} \kappa_{ij} \|\mathrm{Log}(R_j^\top R_i R_{ij})\|_2^2 + \tau_{ij} \|t_j - (R_i t_{ij} + t_i)\|_2^2,$$

where $\mathrm{Log}(\cdot)$ is the capitalized logarithm map that maps $R \in \mathrm{SO}(3)$ to θu. Please refer to Lie theory [7] for more details.

5.3 Camera coordinate system

In this section, we give an example of transferring a point's reference coordinate system with transformation matrix or pose. Given a point p^w in real-world coordinate, we transfer it to pixel image coordinate of a camera by

$$p^i = \underbrace{K}_{\text{intrinsic}} \quad \underbrace{E}_{\text{extrinsic}} \quad p^w. \tag{5.6}$$

For instance, in visual odometry [4], the goal is to estimate the extrinsic matrix of each camera pose. We can breakdown Equation (5.6) to two steps: first transfer p^w to $p^c = Ep^w$ in the camera's coordinate system, then transfer p^c to pixel coordinate $p^i = Kp^c$ of an image. Since intrinsic matrices are typically known information, the challenge lies in solving E, or with superscript and subscript, E^c_w. Note that absolute poses (the way we normally describe a robot's pose) are in the format of T^w_c, that is, poses and extrinsic matrices are *inverses* of each other. This distinction is crucial when working with different coordinate systems in robotics and computer vision.

References

[1] L. Carlone. 16.485: Visual navigation for autonomous vehicles (lecture notes), 2023.

[2] L. Carlone, R. Tron, K. Daniilidis, and F. Dellaert. Initialization techniques for 3d slam: A survey on rotation estimation and its use in pose graph optimization. In *2015 IEEE International Conference on Robotics and Automation*, pages 4597–4604, 2015.

[3] D. Eberly. Euler angle formulas. *Geometric Tools, LLC, Technical Report*, pages 1–18, 2008.

[4] X. Gao, T. Zhang, Y. Liu, and Q. Yan. 14 lectures on visual slam: from theory to practice. *Publishing House of Electronics Industry*, pages 206–234, 2017.

[5] D. M. Rosen, L. Carlone, A. S. Bandeira, and J. J. Leonard. Se-sync: A certifiably correct algorithm for synchronization over the special euclidean group. *The International Journal of Robotics Research*, 38(2-3):95–125, 2019.

[6] G. G. Slabaugh. Computing euler angles from a rotation matrix. *Retrieved on August*, 6(2000):39–63, 1999.

[7] J. Sola, J. Deray, and D. Atchuthan. A micro lie theory for state estimation in robotics. *arXiv:1812.01537*, 2018.

[8] J. Stuelpnagel. On the parametrization of the three-dimensional rotation group. *SIAM review*, 6(4):422–430, 1964.

[9] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5745–5753, 2019.