Development of All-Day Visual Inertial Odometry with Infra-Red Sensors

Standarised Use of Quaternion for $SO(3)$ Representation
Draft

By
Cheng Huimin

In fulfilment for B.Eng. Dissertation

Innovation & Design-Centric Programme
Department of Electrical and Computer Engineering
National University of Singapore

Supervised by Dr. Gao Zhi Temasek Laboratories, NUS
Dr. Lin Feng Temasek Laboratories, NUS

2018

Contents

1 $SO(3)$: Rotations in Rigid-Body Motion

Rigid-body motions (also known as special Euclidean transformations) preserve **distances** (inner product) and **orientations** (cross product) [1, p. 20]. Among rigid-body motions in $\mathbb{R}^3$, we consider a special type - *rotations around the origin.*

It is important to draw distinctions between rotation (representations) from the physical phenomenon. Physical rotations follow a defined path: either following a smooth trajectory or a irregular one, either rotating fraction of a round or multiple rounds. For a rotation representation however, most of the time only the **transformation from the initial orientation to the final orientation** is represented, disregarding the exact rotation path to fulfil the motion physically. In the context of this paper, "rotation" takes the latter meaning.

In $\mathbb{R}^3$, the **group of rotations around the origin** is defined as the *rotation group* $SO(3)$[2, p. 13][1, p. 24]. By definition:

Properties of Rotation Group $SO(3)$

1. Preserves origin
2. Preserves vector length (Euclidean distance)
3. Preserves relative vector orientation (i.e. handedness)

We could define an operator $r()$ that satisfy the above properties:

$$r : \mathbb{R}^3 \to \mathbb{R}^3; \boldsymbol{v} \to r(\boldsymbol{v}) \qquad (\boldsymbol{v} \in \mathbb{R}^3)$$
$$\boldsymbol{v'} = r(\boldsymbol{v}) \tag{1.1}$$

There are different ways to represent $r()$. For example, $r()$ could be represented by Euler angles (a set of three parameters α, β, γ which are often known as roll, pitch, yaw angles). It is closely simulated to mechanical gimbal structure, but it suffers from the gimbal lock issues as well as numerical instability. Therefore it is not in the scope of discussion here.

Another two important ways to represent rotation $r()$ are **axis-angle representation** and **matrix representation**. The matrix representation comes directly from the fact that rotation is a linear transformation (proved by geometry or Rodrigues' rotation formula).

1.1 Axis-Angle Representation

Axis-angle representation is arguably the most intuitive representation of a rotation. A rotation can be expressed by a *rotation axis* and a *rotation angle.* The well known **Rodrigues' rotation formula** is:

$$\boldsymbol{x'} = \boldsymbol{x}_{\parallel} + \boldsymbol{x}_{\perp} \cos\phi + (\boldsymbol{u} \times \boldsymbol{x} \sin\phi) \tag{1.2}$$

or its equivalent form

$$\boldsymbol{x'} = \boldsymbol{x} \cos\phi + (\boldsymbol{u} \times \boldsymbol{x}) \sin\phi + \boldsymbol{u}(\boldsymbol{u} \cdot \boldsymbol{x})(1 - \cos\phi) \tag{1.3}$$

where $\boldsymbol{x}$ is the vector before rotation, and $\boldsymbol{x'}$ is the vector after rotation. The rotation is defined by the rotation axis $\boldsymbol{u}$ (unit vector) and rotation angle ϕ (right-hand rule).

As mentioned, it is helpful to consider the axis $\boldsymbol{u}$ and angle ϕ not as the descriptions for the physical rotation path, but merely a way to describe the transformation from the initial to the final orientation. (i.e. The physical rotation in between may not follow the axis and angle defined.) Furthermore, this representation allows us to represent rotations with multiple revolutions with $|\phi|$ greater than π. Therefore, there is a multiple-to-one mapping between axis-angle representation and the rotation group $SO(3)$.

Therefore, to summarise:

Properties of Axis-Angle Representation:

1. Axis-angle representation obeys **right-hand rule** (positive ϕ corresponds to anticlockwise rotation);
2. The formula proves that $r()$ is a **linear transformation**, since it is defined from the scalar and vector products only [2, p. 15].
3. There is a **multiple-to-one** mapping between axis-angle representation $(\boldsymbol{u}, \phi)$ to rotation group $SO(3)$.

Notice that axis-angle representation works directly with vectors, without relying on matrix operations. Nevertheless, we could show that such representation could be converted to matrix form, since it is a linear transformation after all. Similarly, axis-angle representation could be converted to quaternion form as well.

1.2 Matrix Representation

All linear transformations could be represented by matrix multiplications, so does the rotation transformation $r()$. Specifically, rotation $r()$ in $\mathbb{R}^3$ and can be represented by a matrix $R \in \mathbb{R}^{3\times3}$.

In addition, rotation preserves length and orientation which makes the rotation matrix R orthogonal with $det(R) = +1$. Such group of R is called *special orthogonal matrices*. There exists a **one-to-one mapping** between R and $r()$. In fact, $SO(3)$ is often defined using such matrices R.

$$r(\boldsymbol{v}) = R\boldsymbol{v} \tag{1.4}$$

Properties of Rotation Matrices R

1. The rotation matrix preserve relative orientation (i.e. handedness) of the frame [3, p. 69].
2. Product of two rotation matrices and inverse of a rotation matrix produce another rotation matrix.
3. The terms rotation group $SO(3)$ and rotation matrices are interchangeable. Every rotation in the rotation group can be represented by a unique rotation matrix R.

We proceed to define the elements in R. In linear algebra, a linear transformation can be completely determined by the transformations made to the basis vectors (as any vectors are linear combination of the basis vectors). We could define a fixed world frame W (with principle axes X, Y, Z) and a rotating body frame C (with basis vectors $\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3$).

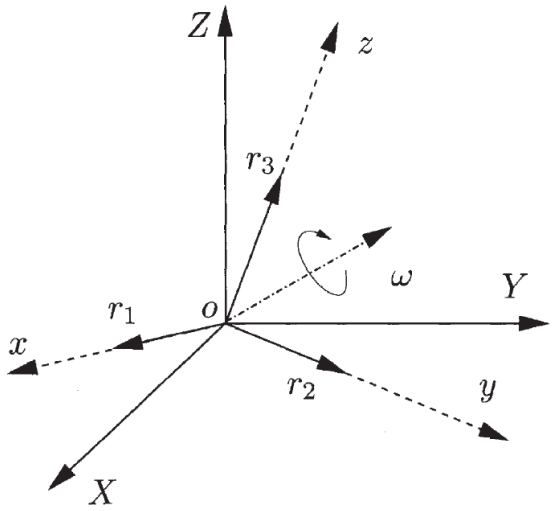

Figure 1: Relative rotation between world frame W and body frame C [1, p. 23]

Note: The standard *right-handed Cartesian coordinate system* is used throughout the discussion. Rotations by definition preserves this handedness.

C is aligned with W before rotation. After the rotation illustrated in Figure 1, the corresponding rotation matrix is:

$$R = [\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3] \tag{1.5}$$

Meaning of Elements in Rotation Matrix

The columns of a rotation matrix represents the rotation of each basis vectors. As each vector to be rotated is defined by the linear combination of those basis vectors, the vector rotation in the column space ($\mathbb{R}^3$) is fully defined by the rotation matrix.

While it is always true that rotations can be represented by matrices, matrix representation may not be the most concise and numerically stable one. Rotation matrix has the advantages of doing vector rotation operation efficiently. Quaternion equivalent operation requires implicit quaternion-matrix conversion which is more costly compared to direct matrix operation [4]. However, quaternion representation gives superior numerical properties and reduction on data storage space.

1.3 Axis-Angle Represented in Matrix Form

As we have shown, the original axis-angle representation describe rotation operation by some algebraic equation. It has been successful in expressing rotation using 4 numbers (unit vector $\boldsymbol{u}$ and scalar angle ϕ) or 3 numbers (if we define a rotation vector $\boldsymbol{v} = \phi\boldsymbol{u}$). It is a big advantage over the matrix representation where 9 numbers are needed, with 6 constraints.

However, algebraic operation is less structured than matrix multiplication, and algebraic operation struggles to perform composition (combining multiple rotation into one). Matrix representation is superior in terms of computation efficiency and ease of implementation on computer. Therefore, we have the motivation to convert axis-angle representation into the equivalent matrix form.

1.3.1 Convert Axis-Angle to Rotation Matrix (Rodrigues' Formula)

If we factor and expand the Rodrigues' rotation formula (1.3) into matrix form, we could obtain **Rodrigues' rotation matrix**:

Rodrigues' Rotation Formula: $\boldsymbol{x}' = \boldsymbol{x}\cos\phi + (\boldsymbol{u}\times\boldsymbol{x})\sin\phi + \boldsymbol{u}(\boldsymbol{u}\cdot\boldsymbol{x})(1-\cos\phi)$ (from Eq. 1.3)

Rodrigues' Rotation Matrix:
$$R(\boldsymbol{u},\phi) = I + \sin\phi[\boldsymbol{u}]_\times + (1-\cos\phi)[\boldsymbol{u}]_\times{}^2 \tag{1.6}$$

where $[\boldsymbol{u}]_\times$ means skew-symmetric matrix defined by vector $\boldsymbol{u}$,

$$[\boldsymbol{u}]_\times \triangleq \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix} \tag{1.7}$$

We could visually inspect that every 2π repetition in ϕ results in the same R. Expanding into elements would obtain:

$$\begin{bmatrix} \mathrm{c}_\theta + \hat{\omega}_1^2(1-\mathrm{c}_\theta) & \hat{\omega}_1\hat{\omega}_2(1-\mathrm{c}_\theta) - \hat{\omega}_3\mathrm{s}_\theta & \hat{\omega}_1\hat{\omega}_3(1-\mathrm{c}_\theta) + \hat{\omega}_2\mathrm{s}_\theta \\ \hat{\omega}_1\hat{\omega}_2(1-\mathrm{c}_\theta) + \hat{\omega}_3\mathrm{s}_\theta & \mathrm{c}_\theta + \hat{\omega}_2^2(1-\mathrm{c}_\theta) & \hat{\omega}_2\hat{\omega}_3(1-\mathrm{c}_\theta) - \hat{\omega}_1\mathrm{s}_\theta \\ \hat{\omega}_1\hat{\omega}_3(1-\mathrm{c}_\theta) - \hat{\omega}_2\mathrm{s}_\theta & \hat{\omega}_2\hat{\omega}_3(1-\mathrm{c}_\theta) + \hat{\omega}_1\mathrm{s}_\theta & \mathrm{c}_\theta + \hat{\omega}_3^2(1-\mathrm{c}_\theta) \end{bmatrix}$$

Figure 2: Axis-angle representation to $SO(3)$ matrix elements [3, p. 86]

1.3.2 Convert Axis-Angle to Rotation Matrix (Exponential Map)

The conversion described in the previous section is still cumbersome, and there exists a more mathematically concise expression converting between axis-angle and matrix form - *exponential map*.

Given a rotation trajectory $R(t) : \mathbb{R} \to SO(3)$ which describe a continuous rotation motion, at any moment we must have:

$$R(t)R^T(t) = I \tag{1.8}$$

By taking derivative on both sides, we could prove that $\dot{R}(t)R^T(t)$ is a *skew-symmetric matrix*. We can then define:

$$[\boldsymbol{\omega}(t)]_\times = \dot{R}(t)R^T(t) \tag{1.9}$$
$$\dot{R} = [\boldsymbol{\omega}(t)]_\times R(t) \tag{1.10}$$

We also know that the space of all 3×3 skew-symmetric matrices is denoted by:

$$so(3) = \{[\boldsymbol{\omega}]_\times \in \mathbb{R}^{3\times 3} \mid \boldsymbol{\omega} \in \mathbb{R}^3\} \tag{1.11}$$

By assigning $R(t)$ as identity, we can interpret $so(3)$ as the *tangent space* or *velocity space* at the identity of the rotation group $SO(3)$. Intuitively, we can think of $\boldsymbol{\omega}(t)$ as the vector of instantaneous angular velocities. [2, p. 17][1, p. 26]

We could proceed to express $SO(3)$ in terms of $so(3)$. By assuming $\boldsymbol{\omega}(t) = constant = \boldsymbol{\omega}$, we could solve equation (1.10) by:

$$R(t) = e^{[\boldsymbol{\omega}]_\times t}R(0) \tag{1.12}$$

where $e^{[\boldsymbol{\omega}]_\times t}$ is also the *state transition matrix* for linear ODE $\dot{x}(t) = [\boldsymbol{\omega}]_\times x(t)$. Since $R(t)$ and $R(0)$ are rotation matrices, $e^{[\boldsymbol{\omega}]_\times t}$ must be rotation matrix as well. By assigning $\boldsymbol{v} = \boldsymbol{w}\Delta t$, in general we can state that the exponential of a 3×3 skew-symmetric matrix is a rotation matrix [2, p. 17]:

$$R = e^{[\boldsymbol{v}]_\times} \tag{1.13}$$

Therefore, we could relate each skew-symmetric matrix with a rotation matrix. This is known as the exponential map:

$$\exp:\ so(3) \to SO(3); [\boldsymbol{v}]_\times \mapsto e^{[\boldsymbol{v}]_\times} \tag{1.14}$$

From $\mathbb{R}^3$ to Rotation Matrix R

Any vector in $\mathbb{R}^3$ could be mapped to a rotation matrix using the exponential map (by first converting $\mathbb{R}^3$ into a unique skew-symmetric matrix). In fact the vector is called the rotation vector $\boldsymbol{v} = \phi\boldsymbol{u}$.

After manipulation and Taylor expansion [2, p. 18], we could obtain:

$$R(\boldsymbol{v}) = e^{[\boldsymbol{v}]_\times} = e^{\phi[\boldsymbol{u}]_\times} = I + \sin\phi[\boldsymbol{u}]_\times + (1 - \cos\phi)[\boldsymbol{u}]_\times{}^2 = R(\boldsymbol{u}, \phi) \quad \text{using } \boldsymbol{v} = \phi\boldsymbol{u} \tag{1.15}$$

which is the Rodrigues' rotation matrix. This shows the axis-angle representation and exponential map are **equivalent**. The exponential form of rotation is another way of expressing axis-angle rotation.

Properties of Exponential Map:

1. Exponential map takes in three independent parameters in $\mathbb{R}^3$, which could be interpreted as rotation vector.
2. Exponential map converts the axis-angle representation (i.e. the rotation vector) to a rotation matrix.

Therefore, we could define a rotation matrix from a rotation vector $\boldsymbol{v} = \phi\boldsymbol{u}$. This definition allow as to reduce the parameters from 9 to 3. We could interpolate rotations conveniently using such rotation vector. However, we still have difficulties to use axis-angle representation to perform rotation composition. This inspires us to explore alternative multiplication rules to allow rotations to be combined directly in the axis-angle "domain", without explicit conversion to matrices. This is perhaps one of the motivation to use a new quaternion representation.

2 $SO(3)$ Represented by Quaternions

2.1 Motivations

It is well-accepted to describe orientation in two-dimension using complex numbers $a+b\boldsymbol{i}$, and a 2D rotation could be performed as multiplications of the complex numbers, which is mathematically defined. Following the multiplication rule, Euler's Formula could be verified and used for conversion between complex number and polar representation. The complex number construction makes describing rotation in 2D systematic.

Inspired by this, we could extend the concept of complex number to describe rotation in higher dimensions. In $\mathbb{R}^3$, it turns out to be **quaternions**.

Quaternions gives an alternative mathematical construct to 3D rotations, allowing composition of rotations without relying on linear transformation and matrices. There are less redundant elements compared to matrix counterpart (one versus six), and it is numerically stable.

However, there is no fixed single convention on how multiplication of quaternions should be defined. This induces much confusion across literature. Therefore, we will explore those conventions and decide the convention to use in our research.

2.2 Two Definitions of Quaternion Multiplications

If we ignore the ordering of scalar and vector components (which makes no major differences), the definition of quaternion is universal:

$$q = q_w + q_x\boldsymbol{i} + q_y\boldsymbol{j} + q_z\boldsymbol{k} \quad \Leftrightarrow \quad q = q_w + \boldsymbol{q}_v \tag{2.1}$$

where $\boldsymbol{q}_v$ is the vector component of the quaternion, and q_w is the scalar part.

There are two different popular conventions to define multiplication of two quaternions. Under the *Hamilton* quaternion multiplication $\odot$ [2, p. 6], the quaternion vectors $\boldsymbol{i},\boldsymbol{j},\boldsymbol{k}$ obey **right-hand rule** (e.g. $\boldsymbol{ij} = -\boldsymbol{ji} = \boldsymbol{k}$); whereas under the *JPL* quaternion multiplication $\otimes$, the quaternion vectors $\boldsymbol{i},\boldsymbol{j},\boldsymbol{k}$ obey **left-hand rule** (e.g. $\boldsymbol{ji} = -\boldsymbol{ji} = \boldsymbol{k}$).

> **Note**: The handedness here is not relevant to the handedness of the coordinate system. In both cases, the standard *right-handed Cartesian coordinate system* is used.

The $\odot$ quaternion multiplication can be referred as the *Hamilton*, *standard* or *historical* convention, whereas the $\otimes$ counterpart can be referred as the *JPL*, *alternative* or *natural* convention.

Following the above definition, by doing term-by-term multiplication:

$$\text{Hamilton: } p \odot q \triangleq \begin{bmatrix} p_w q_w - \boldsymbol{p}_v^T \boldsymbol{q}_v \\ p_w \boldsymbol{q_v} + q_w \boldsymbol{p}_v + \boldsymbol{p}_v \times \boldsymbol{q}_v \end{bmatrix} \qquad (\boldsymbol{ij} = -\boldsymbol{ji} = \boldsymbol{k}) \tag{2.2}$$

$$\text{JPL: } p \otimes q \triangleq \begin{bmatrix} p_w q_w - \boldsymbol{p}_v^T \boldsymbol{q}_v \\ p_w \boldsymbol{q_v} + q_w \boldsymbol{p}_v - \boldsymbol{p}_v \times \boldsymbol{q}_v \end{bmatrix} \qquad (\boldsymbol{ji} = -\boldsymbol{ji} = \boldsymbol{k}) \tag{2.3}$$

The operator $\times$ denotes cross product (standard right-hand rule convention). $p \odot q$ can be called **right-handed** multiplication, whereas $p \otimes q$ is called **left-handed** multiplication.

Due to the anti-commutativity of the vector cross products, we observe [5]:

$$p \odot q = q \otimes p \tag{2.4}$$

The proof of equation (2.4) can be found in [6, p. 467].

Effectively, the two conventions are defined to have **flipped multiplication operations**. The underlying reason for the two different conventions lies in the process of relating quaternions with either active or passive rotations. This will be explained in the following sections.

Some Useful Properties of Quaternions
The **identity quaternion** is defined as,

$$q_1 = 1 = \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix} \tag{2.5}$$

The **conjugate** of a quaternion is defined as,

$$q^* = q_w - \boldsymbol{q}_v = \begin{bmatrix} q_w \\ \boldsymbol{q}_v \end{bmatrix} \tag{2.6}$$

and

$$(p \odot q)^* = q^* \odot p^* \tag{2.7}$$

The **inverse** of a quaternion is defined by,

$$q \odot q^{-1} = q^{-1} \odot q = q_1 \tag{2.8}$$

and for unit quaternion

$$q^{-1} = q^* \tag{2.9}$$

2.3 Unit Quaternion and Axis-Angle Representation

Quaternion is just one of many possible representations of $SO(3)$. To reiterate, $SO(3)$ is the *special orthogonal matrices* in $\mathbb{R}^{3\times3}$, commonly referred as the *rotation group*[1, p. 24] (orthogonal matrices with determinant of 1). The advantages of using quaternions to represent rotations include their numerical efficiencies (4 parameters compared to 9 in rotation matrices) and lack of singularity (i.e. no gimbal lock issues). In contrast to Euler angles which operates on the concept of row, pitch and yaw, **quaternion representation is linked to rotation group through the axis-angle representation**.

The axis-angle rotation representation utilises four parameters (three for unit vector of rotation axis $\boldsymbol{u}$, and one for rotation angle ϕ). Similar to the idea of representing two-dimensional rotation using complex numbers $a + b\boldsymbol{i}$, two more components are introduced to represent rotation in 3D, making the representation in the form $a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}$.

The four components in quaternions can be closely related (but not the same!) to the four parameters used to represent axis-angle rotation. It turns out that all axis-angle rotation could be represented by a **unit quaternion**. The mapping between the a unit quaternion and a axis-angle rotation are defined to be (an extension of Euler's formula) [2, p. 22]:

$$\mathbf{q} \triangleq e^{\phi \mathbf{u}/2} = cos\frac{\phi}{2} + \mathbf{u}sin\frac{\phi}{2} = \begin{bmatrix} cos\frac{\phi}{2} \\ \mathbf{u}sin\frac{\phi}{2} \end{bmatrix} \tag{2.10}$$

Note: The exponential mapping from rotation vectors to quaternions are identical regardless of the conventions used. Right-hand rule always applies.

As we can see, this formula maps rotation vector (axis-angle representation) to a *unit* quaternion $||q|| = 1$. In fact, we will represent any rotation using unit quaternions. From here onwards, quaternion q will be assumed as a unit quaternion, unless stated otherwise.

For unit quaternions, its conjugate equals to its inverse:

$$q^{-1} = q^* \tag{2.11}$$

To visualise, taking the inverse is equivalent to keep the rotation angle ϕ but to reverse the rotation axis $\boldsymbol{u}$. Therefore, by taking an inverse, q^{-1} performs a opposite rotation compared to q.

Also, negated quaternions map to the same rotation matrix [1, p. 41]:

$$R(q) = R(-q) \tag{2.12}$$

Notice, quaternions with ϕ multiples of 2π results in the same quaternion.

In summary:

Properties of Unit Quaternions:

1. Unit quaternions represent rotations.
2. Inverse or conjugate of a unit quaternion represents the inverse rotation.
3. Negated unit quaternions represent the same rotation.
4. There is a two-to-one mapping from unit quaternions to rotation group $SO(3)$ (double cover [2, p. 23]).

2.4 Double Quaternion Product as Rotation Group

In previous sections, we have shown that axis-angle representation perform rotations to vectors based on defined algebraic operations or matrix multiplications. By mapping axis-angle representation to quaternions, we could define another way to perform rotation - *double quaternion product*.

For Hamilton convention, the double quaternion product is defined as:

$$\boldsymbol{x}' = q \odot \boldsymbol{x} \odot q^* \tag{2.13}$$

Whereas for JPL convention, the double quaternion product is:

$$\begin{aligned} \boldsymbol{x}' &= q \otimes \boldsymbol{x} \otimes q^* && (2.14) \\ &= (\boldsymbol{x} \otimes q^*) \odot q \\ &= q^* \odot \boldsymbol{x} \odot q \\ &= q^{-1} \odot \boldsymbol{x} \odot (q^{-1})^* && (2.15) \end{aligned}$$

It could be proved that the double quaternion operation is equivalent a rotation operation, where the components of q (converting to $\boldsymbol{u}$ and ϕ) can be interpreted as axis-angle representation, using the inverse of equation (2.10). For actual C++ implementation of the inversion could be found in Appendix A.

Since q^{-1} multiply q is defined to be identity quaternion $q_1 = 1$, the double quaternion products in the two conventions perform the exactly the opposite rotation (think of the rotation vector).

Rotation using Double Quaternion Products:

The different definitions of multiplication of quaternions in Hamilton and JPL conventions result in **opposite rotation operations**, from the **same quaternion**. This distinctions create the concept of active and passive rotations, discussed later.

To further convince ourselves that the two conventions results in opposite rotations from the same quaternion number, we could expand out the double quaternion product to the rotation matrix form:

$$\text{Hamilton [2, p. 25]: } R(q) = (q_w^2 - 1)I + 2\boldsymbol{q}_v\boldsymbol{q}_v^T + 2q_w[\boldsymbol{q}_v]_\times \quad (2.16)$$

$$\text{JPL [7, p. 8]: } C(q) = (q_w^2 - 1)I + 2\boldsymbol{q}_v\boldsymbol{q}_v^T - 2q_w[\boldsymbol{q}_v]_\times \quad (2.17)$$

Note, the skew operator $[\ ]_\times$ is defined consistently in both conventions [2][7].

Evidently, the two rotation matrix are transpose (i.e. inverse) of each other:

$$R(q) = C^T(q) \quad (2.18)$$

2.5 Activeness of Transformation

We would first define the active and passive transformations. For an **active rotation**, what the transformation does is to rotate the vectors (e.g. body subject) physically in space, with reference to a fixed frame (e.g. world). This type of rotation is used in most mathematics literature. In contrast, a **passive rotation** is where the vector remains at the same location physically, but the rotation is performed to the reference frame instead. Geometrically, the **active and passive rotations are inverse operations** of each other.

2.5.1 Activeness of Rotation Matrix

By inspecting the Rodrigues' rotation formula, axis-angle representation is an active rotation. This is because the formula assumes rotation on the vector $\boldsymbol{x}$ in a fixed frame (standard basis), referring to the equation 1.3. .

In contrast, rotation matrix R by itself does not contain any information about activeness. Matrix merely represents linear transformation in its most intrinsic format: transformations made to each

basis vectors (equation 1.5). However, if we start to construct mapping from a particular rotation representation to rotation matrices, we have to adopt a defined activeness.

Rotation matrix $R(\boldsymbol{u}, \phi)$ formulated by Rodrigues' Rotation Matrix (1.6) is an active rotation, due inheriting the activeness of axis-angle representation.

Similarly, rotation matrix $R(\boldsymbol{v})$ defined using exponential map is an active rotation as well, as $R(\boldsymbol{v}) = R(\boldsymbol{u}, \phi)$ are equivalent.

Now we are left with the activeness of $R(q)$ and $C(q)$. Strictly speaking, quaternion utilises the concept of axis-angle representation, and therefore quaternion q by itself represents an active rotation. However, the two conventions gives are two different mapping rules from quaternion to rotation matrix, namely $R(q)$ and $C(q)$, we effectively have both choices open.

It could be proved that [2, p. 23]:

$$q \odot \boldsymbol{x} \odot q^* = R(q)\boldsymbol{x} = R(\boldsymbol{u}, \phi)\boldsymbol{x} \tag{2.19}$$

Therefore, $R(q)$ is an active rotation. Since $C(q) = R^T(q)$ and the fact that active and passive rotation is inverse of each other, $C(q)$ is a passive rotation.

Summary of Activeness of Transformations:

Representations	Activeness
Axis-Angle $(\boldsymbol{u}, \phi)$	Active
Rotation Matrix R	-
Rotation Matrix $R(\boldsymbol{u}, \phi)$ or $R(\boldsymbol{v})$	Active
Quaternion q	Active
Rotation Matrix $R(q)$	Active
Rotation Matrix $C(q)$	Passive
toRotationMatrix() [a]	Active

[a] Implementation from Eigen C++ Library, more from Appendix B

To further illustrate the distinctions between active and passive notations, refer to Figure 3.

2.5.2 Standarising Notations

From the previous section, we found out the same quaternion could correspond to totally opposite rotation matrices. To better interpret their significance, let us define the notation:

$$q_b^a = \text{rotation from basis b to basis a} \tag{2.20}$$
$$= \text{orientation of basis a w.r.t. basis b} \tag{2.21}$$
$$q_b^a = (q_a^b)^{-1} = (q_a^b)^* \tag{2.22}$$

As q correspond to a pair of $\boldsymbol{u}$ and ϕ, this quaternion represent the rotation of basis b to align with basis a, using the axis-angle pair $(\boldsymbol{u}, \phi)$. In other words, quaternion q_b^a represents the active rotation of basis b to align with basis a.

To reiterate, **Hamilton convention represents active rotation**, and we denote the corresponding rotation matrix as $R(q)$; **JPL convention represents passive rotation**, and we denote the corresponding rotation matrix as $C(q) = R^T(q)$.

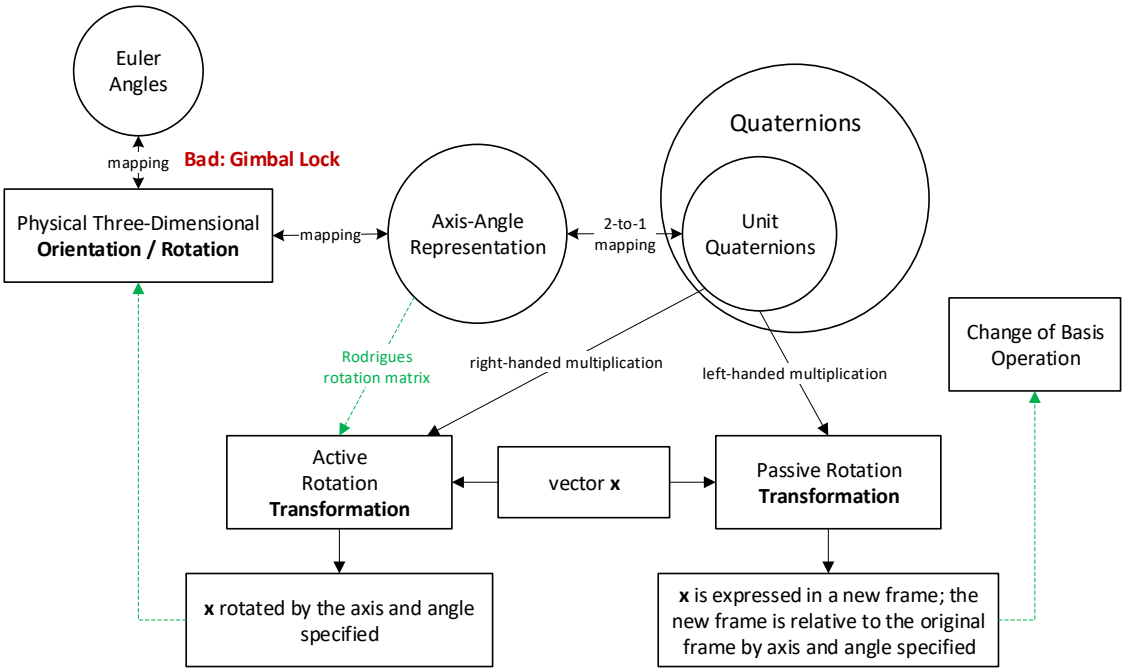

Figure 3: Illustration of Relationships between Active and Passive Transformations

Interpreting $R(q)$ and $C(q)$:

Representation	Meaning
q_b^a	orientation of basis a w.r.t. basis b,
$R(q_b^a) = C^T(q_b^a)$	rotate vector expressed in basis b, by amount specified by quaternion q
	change the vector basis from a to b
$C(q_b^a) = R^T(q_b^a)$	change the vector basis from b to a
	rotate vector expressed in basis a, by amount specified by quaternion $q^{-1} = q_a^b$

As a rule of thumb: for rotating vectors physically in space, we use $R(q)$; for changing of basis from b to a, we use $C(q_b^a)$ or equivalently $R^T(q_b^a)$.

2.6 Rotation Composition

For Hamilton convention, rotation composition is performed in the flipped order [2, p. 26]. Rotation from c to a means rotation from b to a, and then c to b.

$$q_c^a = q_c^b \odot q_b^a \tag{2.23}$$

$$R(q_c^a) = R(q_c^b)R(q_b^a) \tag{2.24}$$

That is, to rotate vector by amount specified by q_c^a (i.e. orientation of basis a in basis c), $R(q_b^a)$ is performed before $R(q_c^b)$. This is a **local-to-global** composition approach [2, p. 34]. In this case, basis a is the most local, and basis c is the most global.

In contrast, for JPL convention, rotation composition is more "natural": change of basis from c to a means change from c to b, and then b to a.

$$q_c^a = q_b^a \otimes q_c^b \tag{2.25}$$
$$C(q_c^a) = C(q_b^a)C(q_c^b) \tag{2.26}$$

That is, to change basis specified by q_c^a (i.e. change from basis c to basis a), $C(q_c^b)$ is performed before $C(q_b^a)$. This is a **global-to-local** composition approach.

Again, the distinctions between the rotation composition rule is determined by the difference of definition of quaternion multiplication in the two conventions. They are describing the same rotation phenomenon using the reversed language.

2.7 Some Example Applications

Error Quaternions (Local) In a physical system, if we want to express the true quaternion q in terms of the quaternion estimate $\hat{q}$, we could define the error quaternion δq in both conventions equivalently [7, p. 16][2, p. 43]:

$$q = \delta q \otimes \hat{q} \tag{2.27}$$
$$q = \hat{q} \odot \delta q \tag{2.28}$$

Note that δq is defined in the local frame of reference in this case.

If we assume the error quaternion is very small in rotation ($\phi \approx 0$), we can use small angle approximation [7, p. 8] [2, p. 43]:

$$\delta q = \begin{bmatrix} \cos(\phi/2) \\ \boldsymbol{u}\sin(\phi/2) \end{bmatrix} \approx \begin{bmatrix} 1 \\ \frac{1}{2}\delta\boldsymbol{v} \end{bmatrix} \quad (\boldsymbol{v} = \phi\boldsymbol{u}) \tag{2.29}$$

By Taylor expansion up to the linear term [2, p. 43]:

$$R(\delta q) = e^{[\boldsymbol{v}]_\times} \approx I + [\delta\boldsymbol{v}]_\times \tag{2.30}$$

For JPL convention, similar results could be obtained [7, pp. 8–9]:

$$C(\delta q) = e^{-[\boldsymbol{v}]_\times} \approx I - [\delta\boldsymbol{v}]_\times \tag{2.31}$$

We could call $R(\delta q)$ and $C(\delta q)$ the **error rotation matrices**, and

$$R(\delta q) = C^T(\delta q) \tag{2.32}$$

Time Derivatives (Local) Following from the error quaternions, we could express the time derivatives of quaternion. We define:

$$\boldsymbol{\omega}(t) \equiv \frac{d\delta\boldsymbol{v}}{dt} \equiv \lim_{\Delta t \to 0} \frac{\Delta\delta\boldsymbol{v}}{\Delta t} \tag{2.33}$$

Then it could be shown that the time derivative of q is [2, p. 44]

$$\begin{aligned} \dot{q} &= \frac{1}{2} q \odot \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix} \\ &= \frac{1}{2} q \odot \boldsymbol{\omega} \qquad (2.34) \\ &= \frac{1}{2} \Omega(\boldsymbol{\omega}) q \qquad (2.35) \end{aligned}$$

where

$$\Omega(\boldsymbol{\omega}) \equiv \begin{bmatrix} 0 & -\boldsymbol{\omega}^T \\ \boldsymbol{\omega} & -[\boldsymbol{\omega}]_\times \end{bmatrix} = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix} \tag{2.36}$$

This also has a matrix equivalent form,

$$\dot{R} = R[\boldsymbol{\omega}]_\times \tag{2.37}$$

Error Rotation Matrices For a given scenario, we know the relationship between inertial frame and world frame:

$$C(q_w^i) = \text{change of frame from world to inertial} \tag{2.38}$$

or

$$R(q_w^i) = C^T(q_w^i) = C(q_i^w) = \text{change of frame from inertial to world} \tag{2.39}$$

We could obtain the following identity equations:

$$C(q_w^i) = C(q_{\hat{i}}^i)C(q_w^{\hat{i}}) \iff q_w^i = q_{\hat{i}}^i \otimes q_w^{\hat{i}} \tag{2.40}$$

$$R(q_w^i) = R(q_w^{\hat{i}})R(q_{\hat{i}}^i) \iff q_w^i = q_w^{\hat{i}} \odot q_{\hat{i}}^i \tag{2.41}$$

where $\hat{i}$ is the estimated inertial frame, and $q_{\hat{i}}^i$ is the error quaternion δq.

Appendix A Quaternion to Axis-Angle Conversion

Code excerpt from Eigen C++ Library which convert quaternion back into axis-angle representation [8].

```
EIGEN_DEVICE_FUNC AngleAxis<Scalar>& AngleAxis<Scalar>::operator=
        (const QuaternionBase<QuatDerived>& q)
{
    EIGEN_USING_STD_MATH(atan2)
    EIGEN_USING_STD_MATH(abs)
    Scalar n = q.vec().norm();
    if(n<NumTraits<Scalar>::epsilon())
    n = q.vec().stableNorm();

    if (n != Scalar(0))
    {
    m_angle = Scalar(2)*atan2(n, abs(q.w()));
    if(q.w() < Scalar(0))
        n = -n;
    m_axis  = q.vec() / n;
    }
    else
    {
    m_angle = Scalar(0);
    m_axis << Scalar(1), Scalar(0), Scalar(0);
    }
    return *this;
}
```

Appendix B Quaternion to Rotation Matrix Conversion in C++

Code excerpt from Eigen C++ Library which convert quaternion to rotation matrix representation [9]. It shows that the mapping gives a active rotation (comparing with the formula in [2, p. 25]).

```
template<class Derived>
EIGEN_DEVICE_FUNC inline typename QuaternionBase<Derived>::Matrix3
QuaternionBase<Derived>::toRotationMatrix(void) const
{
    Matrix3 res;

    const Scalar tx  = Scalar(2)*this->x();
    const Scalar ty  = Scalar(2)*this->y();
    const Scalar tz  = Scalar(2)*this->z();
    const Scalar twx = tx*this->w();
    const Scalar twy = ty*this->w();
    const Scalar twz = tz*this->w();
    const Scalar txx = tx*this->x();
    const Scalar txy = ty*this->x();
    const Scalar txz = tz*this->x();
    const Scalar tyy = ty*this->y();
    const Scalar tyz = tz*this->y();
    const Scalar tzz = tz*this->z();

    res.coeffRef(0,0) = Scalar(1)-(tyy+tzz);
    res.coeffRef(0,1) = txy-twz;
    res.coeffRef(0,2) = txz+twy;
    res.coeffRef(1,0) = txy+twz;
    res.coeffRef(1,1) = Scalar(1)-(txx+tzz);
    res.coeffRef(1,2) = tyz-twx;
    res.coeffRef(2,0) = txz-twy;
    res.coeffRef(2,1) = tyz+twx;
    res.coeffRef(2,2) = Scalar(1)-(txx+tyy);

    return res;
}
```

References

[1] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An invitation to 3-d vision: from images to geometric models*. Springer Science & Business Media, 2012, vol. 26.

[2] J. Sola, "Quaternion kinematics for the error-state kalman filter," *arXiv preprint arXiv:1711.02508*, 2017.

[3] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.

[4] D. Eberly, "Rotation representations and performance issues," *Magic Software, Inc., Chapel Hill, NC*, 2002.

[5] H. Sommer, I. Gilitschenski, M. Bloesch, S. M. Weiss, R. Siegwart, and J. Nieto, "Why and how to avoid the flipped quaternion multiplication," *arXiv preprint arXiv:1801.07478*, 2018.

[6] M. D. Shuster, "A survey of attitude representations," *Navigation*, vol. 8, no. 9, pp. 439–517, 1993.

[7] N. Trawny and S. I. Roumeliotis, "Indirect kalman filter for 3d attitude estimation," *University of Minnesota, Dept. of Comp. Sci. & Eng., Tech. Rep*, vol. 2, p. 2005, 2005.

[8] (2008). Angleaxis.h, eigen c++ library, [Online]. Available: `https://eigen.tuxfamily.org/dox/AngleAxis_8h_source.html`.

[9] (2009). Quaternion.h, eigen c++ library, [Online]. Available: `https://eigen.tuxfamily.org/dox/Quaternion_8h_source.html`.