Quaternion to Euler angles conversion: a direct, general and computationally efficient method

Evandro Bernardes, Stéphane Viollet

Quaternion to Euler angles conversion: a direct, general and computationally efficient method

Evandro Bernardes[1], Stéphane Viollet[1*]

1 Aix-Marseille Université, CNRS, ISM, Marseille cedex 09, France

* stephane.viollet@univ-amu.fr

Abstract

Current methods of the conversion between a rotation quaternion and Euler angles are either a complicated set of multiple sequence-specific implementations, or a complicated method relying on multiple matrix multiplications. In this paper a general formula is presented for extracting the Euler angles in any desired sequence from a unit quaternion. This is a direct method, in that no intermediate conversion step is required (no quaternion-to-rotation matrix conversion, for example) and it is general because it works with all 12 possible sequences of rotations. A closed formula was first developed for extracting angles in any of the 12 possible sequences, both "Proper Euler angles" and "Tait-Bryan angles". The resulting algorithm was compared with a popular implementation of the matrix-to-Euler angle algorithm, which involves a quaternion-to-matrix conversion in the first computational step. Lastly, a single-page pseudo-code implementation of this algorithm is presented, illustrating its conciseness and straightforward implementation. With an execution speed 30 times faster than the classical method, our algorithm can be of great interest in every aspect.

1 Introduction

When dealing with 3D orientation problems, many different formalisms can be used to describe a given rotation [1], each of which has its own set of advantages and shortcomings. Arguably the most direct representation of a 3D rotation is a matrix $R \in SO(3)$, where $SO(3)$ is the group of invertible 3×3 matrices such that $\det(R) = 1$ and $RR^T = R^T R = \mathbb{I}$, where $\mathbb{I}$ is the identity matrix. These rotation matrices represent direct linear transformations such that, with $\boldsymbol{v} \in \mathbb{R}^3$:

$$\boldsymbol{v}_{\text{rotated}} = R\boldsymbol{v} \tag{1}$$

Apart from being simple to use, a rotation matrix also has the advantage of being continuous, and a simple matrix multiplication can be used to compose rotations: $R = R_2\, R_1$ is the rotation matrix corresponding to a rotation by R_1 followed by a rotation by R_2. 3D rotation matrices have some numerical shortcomings, however. For example, as many as 9 numbers (and 6 constraints) are required to represent a 3 degree of freedom rotation, and it can be difficult and computationally costly to orthogonalize a rotation matrix numerically [2] (i.e., to check that the matrix has its determinant equal to 1 and its inverse equal to its transpose, which is necessary to compensate for the accumulated floating point errors).

However, it is possible to parametrize this rotation matrix with a smaller set of numbers [3]. One of the most usual set of parameters are the Euler angles. The

approach consists in decomposing the 3D rotation matrix into the product of three rotations:

$$R = R_{\theta_3 \boldsymbol{e}_3}\, R_{\theta_2 \boldsymbol{e}_2}\, R_{\theta_1 \boldsymbol{e}_1} \tag{2}$$

Where $R_{\theta \boldsymbol{e}}$ is a rotation by the angle θ around the axis $\boldsymbol{e}$, and the consecutive axes are orthogonal ($\boldsymbol{e}_1 \cdot \boldsymbol{e}_2 = \boldsymbol{e}_2 \cdot \boldsymbol{e}_3 = 0$). The advantages of using Euler angles include the fact that only three numbers have to be stored, and due to their familiarity, they can be more easily understood, which explains why they are still being so widely used, even in cases where other forms of representation may be more appropriate. The use of Euler angles also has several disadvantages. For example, they are discontinuous and it is difficult to directly compose two 3D rotations expressed in Euler angles. Euler angles are also affected by the phenomenon commonly called "gymbal lock": when two axes become aligned, making the system underdetermined, special care has to be taken. In addition, since there are 12 possible axis sequences (24, when considering the difference between "intrinsic" and "extrinsic" rotations), the correct sequence has to be checked in the case of each application. An arguably preferable parametrization are quaternions. A quaternion is a hypercomplex number defined by one real part and three distinct imaginary parts (which can also be regarded as the vector part). When the norm of a quaternion is equal to 1, quaternions are a useful and efficient representation of 3D orientation: they can be composed as easily as rotation matrices, they are continuous, and they are easily constructed from the axis-angle representation. In addition, quaternions can be normalized trivially, which is much more efficient than having to cope with the corresponding matrix orthogonalization problem. For these reasons, most 3D graphical applications and rotation engines carry quaternions under the hood. Besides these advantages, Euler angles are still being preferred by many authors: Euler angles are the most familiar concept to most engineers and researchers. In addition, in the case of many problems in which there exists only one degree of freedom, angles can suffice.

To be able to perform fast calculations with quaternions and at the same time analyze rotations using angles, it might be necessary to have an efficient method of converting the one set of parameters to the other. Calculating the corresponding quaternion (or rotation matrix) for a given set of Euler angles is trivial. Extracting the Euler angles is much harder, however. One of the following two methods has generally been used up to now. The first method consists in adopting a different set of formulas for each possible angle sequence [4], which is difficult to implement and debug. The second method is that described in [5]. SciPy [7], for example, a widely used scientific library for the Python programming language, implements this method. It converts rotation matrices into Euler angles and involves many different matrix multiplications, including the inverse trigonometric functions required, which are naturally computationally costly. In addition, if rotations are stored in the form of quaternions (as is usually the case in many of the 3D rendering software tools dealing with rotations), an additional conversion step from quaternions to rotation matrices is necessary.

Since many robotic, graphic and other high-level applications involve the use of quaternions (even if they are hidden from the user), it can be necessary to have a concise, efficient method for the conversion between quaternions and Euler angles. The direct conversion formula from quaternions to Euler angles presented here requires fewer computational steps and less expensive computational resources. Moreover, this conversion formula is much simpler to implement and debug, making it a great option for any new applications needing to implement this kind of conversions.

2 Quaternion algebra summary

In this section, the key properties of quaternions are summarized. It is assumed in this work that we are dealing with the classical Hamilton quaternions. Since the definitions concerning quaternion algebra are not perfectly consistent in the literature [8], some of the main notations and definitions used in this study are then presented. Quaternions form a non-commutative division algebra denoted by $\mathbb{H}$, which extends the complex numbers. A quaternion $\mathrm{q} \in \mathbb{H}$ consists of four components:

$$\mathrm{q} = q_\mathrm{r} + q_\mathrm{x}\boldsymbol{i} + q_\mathrm{y}\boldsymbol{j} + q_\mathrm{z}\boldsymbol{k} \tag{3}$$

Where $q_\mathrm{r}, q_\mathrm{x}, q_\mathrm{y}, q_\mathrm{z} \in \mathbb{R}$. All the properties of quaternions can be obtained using its fundamental property, as given by Hamilton:

$$\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1 \tag{4}$$

Using the above properties, the product of two quaternions q and p can be expressed by the Hamilton product:

$$\begin{aligned}\mathrm{q}\,\mathrm{p} = {} & (p_r q_\mathrm{r} - p_x q_\mathrm{x} - p_y q_\mathrm{y} - p_z q_\mathrm{z}) + (p_r q_\mathrm{x} + p_x q_\mathrm{r} - p_y q_\mathrm{z} + p_z q_\mathrm{y})\,\boldsymbol{i} \\ & + (p_r q_\mathrm{y} + p_x q_\mathrm{z} + p_y q_\mathrm{r} - p_z q_\mathrm{x})\,\boldsymbol{j} + (p_r q_\mathrm{z} - p_x q_\mathrm{y} + p_y q_\mathrm{x} + p_z q_\mathrm{r})\,\boldsymbol{k}\end{aligned} \tag{5}$$

For the sake of simplicity, quaternions will be written here as 4×1 vectors (with the scalar q_r as the first element):

$$q = \begin{bmatrix} q_\mathrm{r} \\ q_\mathrm{x} \\ q_\mathrm{y} \\ q_\mathrm{z} \end{bmatrix} = \begin{bmatrix} q_\mathrm{r} \\ \boldsymbol{q} \end{bmatrix} \tag{6}$$

Where $\boldsymbol{q} = \begin{bmatrix} q_\mathrm{x} & q_\mathrm{y} & q_\mathrm{z} \end{bmatrix}^T$ is the the imaginary/vector part of q. The Hamilton product between two quaternions in 4-vector form will be denoted by:

$$q \odot p = \begin{bmatrix} q_\mathrm{r} \\ \boldsymbol{q} \end{bmatrix} \odot \begin{bmatrix} p_r \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} q_\mathrm{r} p_r - \boldsymbol{q} \cdot \boldsymbol{p} \\ q_\mathrm{r}\boldsymbol{p} + p_r \boldsymbol{q} + \boldsymbol{q} \times \boldsymbol{p} \end{bmatrix} \tag{7}$$

Defining the conjugate $q^* = \begin{bmatrix} q_\mathrm{r} \\ -\boldsymbol{q} \end{bmatrix}$ and the absolute value as $|q| = \sqrt{q_\mathrm{r}^2 + q_\mathrm{x}^2 + q_\mathrm{y}^2 + q_\mathrm{z}^2}$, the inverse q^{-1} of q is given by:

$$q^{-1} = \frac{q^*}{|q|^2} \tag{8}$$

And for any quaternion q:

$$q \odot q^{-1} = q^{-1} \odot q = \begin{bmatrix} 1 \\ \boldsymbol{0} \end{bmatrix} \tag{9}$$

If q is a unit quaternion, which means that $|q| = 1$ and $q^{-1} = q^*$, it can be used to represent the rotation between two reference frames. Denoting $\mathbf{v}_A$ and $\mathbf{v}_B$ a vector $\mathbf{v}$ in frames A and B, respectively, and $q = q_A^B$ the unit quaternion corresponding to the rotation from A to B:

$$\begin{bmatrix} 0 \\ \mathbf{v}_B \end{bmatrix} = q_A^B \odot \begin{bmatrix} 0 \\ \mathbf{v}_A \end{bmatrix} \odot (q_A^B)^* \tag{10}$$

The equivalent rotation matrix is given by:

$$R_A^B = \begin{bmatrix} q_\mathrm{r}^2 + q_\mathrm{x}^2 - q_\mathrm{y}^2 - q_\mathrm{z}^2 & -2q_\mathrm{r}q_\mathrm{z} + 2q_\mathrm{x}q_\mathrm{y} & 2q_\mathrm{r}q_\mathrm{y} + 2q_\mathrm{x}q_\mathrm{z} \\ 2q_\mathrm{r}q_\mathrm{z} + 2q_\mathrm{x}q_\mathrm{y} & q_\mathrm{r}^2 - q_\mathrm{x}^2 + q_\mathrm{y}^2 - q_\mathrm{z}^2 & -2q_\mathrm{r}q_\mathrm{x} + 2q_\mathrm{y}q_\mathrm{z} \\ -2q_\mathrm{r}q_\mathrm{y} + 2q_\mathrm{x}q_\mathrm{z} & 2q_\mathrm{r}q_\mathrm{x} + 2q_\mathrm{y}q_\mathrm{z} & q_\mathrm{r}^2 - q_\mathrm{x}^2 - q_\mathrm{y}^2 + q_\mathrm{z}^2 \end{bmatrix} \tag{11}$$

And the equivalent quaternion for a rotation of an angle θ around an axis $\boldsymbol{e}$ is given by:

$$q_{\theta \boldsymbol{e}} = \begin{bmatrix} \cos(\theta/2) \\ \sin(\theta/2)\boldsymbol{e} \end{bmatrix} \tag{12}$$

3 Formula development

In the section, the formula for the conversion between a quaternion and any of the 6 proper Euler angle sequences is derived, and then an adaptation for the 6 remaining Tait-Bryan sequences is demonstrated.

3.1 Case 1: Proper Euler angles

Assuming $q = [q_\mathrm{r},\, \boldsymbol{q}^T]^T$ is unit an known, it can be decomposed as follows:

$$q = \begin{bmatrix} c_3 \\ s_3\boldsymbol{e} \end{bmatrix} \odot \begin{bmatrix} c_2 \\ s_2\boldsymbol{e}' \end{bmatrix} \odot \begin{bmatrix} c_1 \\ s_1\boldsymbol{e} \end{bmatrix} \tag{13}$$

In which (for $0 \leq \theta_2 \leq \pi$):

$$\begin{aligned} s_1 &\equiv \sin\left(\theta_1/2\right),\ c_1 \equiv \cos\left(\theta_1/2\right) \\ s_2 &\equiv \sin\left(\theta_2/2\right),\ c_2 \equiv \cos\left(\theta_2/2\right) \\ s_3 &\equiv \sin\left(\theta_3/2\right),\ c_3 \equiv \cos\left(\theta_3/2\right) \end{aligned} \tag{14}$$

Where $s_2 \geq 0,\ c_2 \geq 0$. Taking $\boldsymbol{e}$ and $\boldsymbol{e}'$ to be orthogonal unit vectors ($\boldsymbol{e} \cdot \boldsymbol{e}' = 0$), there is a third unit vector which is orthogonal to the other two such that:

$$\boldsymbol{e}'' \equiv \varepsilon\, \boldsymbol{e} \times \boldsymbol{e}' \tag{15}$$

Where $\varepsilon = (\boldsymbol{e} \times \boldsymbol{e}') \cdot \boldsymbol{e}'' = \pm 1$. Together, $\boldsymbol{e}, \boldsymbol{e}'$ and $\boldsymbol{e}''$ form an orthonormal basis. We also define:

$$\begin{aligned} \theta_+ &= \frac{\theta_1 + \theta_3}{2} \\ \theta_- &= \frac{\theta_1 - \theta_3}{2} \end{aligned} \tag{16}$$

And:

$$\begin{aligned} s_+ &\equiv \sin\left(\theta_+\right) = s_1 c_3 + c_1 s_3 \\ s_- &\equiv \sin\left(\theta_-\right) = s_1 c_3 - c_1 s_3 \\ c_+ &\equiv \cos\left(\theta_+\right) = s_1 s_3 - c_1 c_3 \\ c_- &\equiv \cos\left(\theta_-\right) = s_1 s_3 + c_1 c_3 \end{aligned} \tag{17}$$

Analyzing Eq. 13:

$$\begin{aligned} q &= \begin{bmatrix} c_3 \\ s_3\boldsymbol{e} \end{bmatrix} \odot \begin{bmatrix} c_2 \\ s_2\boldsymbol{e}' \end{bmatrix} \odot \begin{bmatrix} c_1 \\ s_1\boldsymbol{e} \end{bmatrix} \\ &= c_2 \begin{bmatrix} c_3 \\ s_3\boldsymbol{e} \end{bmatrix} \odot \begin{bmatrix} c_1 \\ s_1\boldsymbol{e} \end{bmatrix} + s_2 \begin{bmatrix} c_3 \\ s_3\boldsymbol{e} \end{bmatrix} \odot \begin{bmatrix} 0 \\ \boldsymbol{e}' \end{bmatrix} \odot \begin{bmatrix} c_1 \\ s_1\boldsymbol{e} \end{bmatrix} \\ &= c_2 \begin{bmatrix} c_+ \\ s_+\boldsymbol{e} \end{bmatrix} + s_2 \begin{bmatrix} 0 \\ c_-\boldsymbol{e}' + s_-\boldsymbol{e} \times \boldsymbol{e}' \end{bmatrix} \end{aligned} \tag{18}$$

And noting that $\boldsymbol{e} \times \boldsymbol{e}' = \varepsilon\, \boldsymbol{e}''$:

$$q = \begin{bmatrix} c_2 c_+ \\ c_2 s_+ \boldsymbol{e} + s_2 c_- \boldsymbol{e}' + s_2 s_- \varepsilon \boldsymbol{e}'' \end{bmatrix} \tag{19}$$

Defining the following four components:

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \equiv \begin{bmatrix} q_{\mathrm{r}} \\ \boldsymbol{q} \cdot \boldsymbol{e} \\ \boldsymbol{q} \cdot \boldsymbol{e}' \\ \varepsilon\, \boldsymbol{q} \cdot \boldsymbol{e}'' \end{bmatrix} \tag{20}$$

We obtain:

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} c_2 c_+ \\ c_2 s_+ \\ s_2 c_- \\ s_2 s_- \end{bmatrix} \tag{21}$$

Alternatively, we can see that $\begin{bmatrix} b & c & d \end{bmatrix}^T$ is simply a permutation of the components of $\boldsymbol{q}$:

$$\begin{bmatrix} b \\ c \\ d \end{bmatrix} = \begin{bmatrix} \boldsymbol{e} & \boldsymbol{e}' & \boldsymbol{e} \times \boldsymbol{e}' \end{bmatrix}^T \boldsymbol{q} \tag{22}$$

3.1.1 Extraction of angles

Using complex numbers, we can define:

$$\begin{aligned} z_+ &\equiv a + ib = c_2(c_+ + i s_+) \\ z_- &\equiv c + id = s_2(c_- + i s_-) \end{aligned} \tag{23}$$

Since $c_2, s_2 \geq 0$, we know that $|z_+| = c_2$, $\arg(z_+) = \theta_+$, $|z_-| = s_2$ and $\arg(z_-) = \theta_-$. We can then rewrite:

$$\begin{aligned} z_+ &= a + ib = c_2 \exp(i\theta_+) \\ z_- &= c + id = s_2 \exp(i\theta_-) \end{aligned} \tag{24}$$

And we know that:

$$\begin{aligned} \theta_+ &= \frac{\theta_3 + \theta_1}{2} = \arg\{a + ib\} = \operatorname{atan2}(b, a) \\ \theta_- &= \frac{\theta_3 - \theta_1}{2} = \arg\{c + id\} = \operatorname{atan2}(d, c) \end{aligned} \tag{25}$$

3.1.2 Singularities

There are two different singularities in these expressions. When $\theta_2 = 0$, we have $s_2 = 0$ and θ_- is undefined. When $\theta_2 = \pi$, we have $c_2 = 0$ and θ_+ is undefined. In both cases, one degree of freedom is lost and we can argue that θ_1 (or alternatively, θ_3) loses its geometrical meaning. We can then either set θ_1 to zero, or keep it fixed in its latest value (for example, when updating an estimator, for the sake of continuity). Defining:

$$\theta_1 \equiv \hat{\theta}_1 \text{ , if } \theta_2 = 0 \text{ or } \theta_2 = \pi \tag{26}$$

Taking $\hat{\theta}_1$ to be some constant (zero or otherwise), we can calculate:

$$\begin{cases} \theta_3 = 2 \operatorname{atan2}(b, a) - \hat{\theta}_1 \text{ , when } \theta_2 = 0 \\ \theta_3 = 2 \operatorname{atan2}(d, c) + \hat{\theta}_1 \text{ , when } \theta_2 = \pi \end{cases} \tag{27}$$

3.1.3 General formula for θ_1 and θ_3 in the absence of singularities

If $\theta_2 \neq 0$ and $\theta_2 \neq \pi/2$, multiplying z_+ and z_- yields:

$$\begin{aligned} z_+ \, z_- &= (a + ib)(c + id) \\ &= c_2 s_2 \exp\left(i \frac{\theta_3 + \theta_1 + \theta_3 - \theta_1}{2}\right) \\ &= c_2 s_2 \exp(i\theta_3) \end{aligned} \tag{28}$$

On similar lines, multiplying z_+ and the conjugate of z_- yields:

$$\begin{aligned} z_+ \, z_-^* &= (a + ib)(c - id) \\ &= c_2 s_2 \exp(i\theta_1) \end{aligned} \tag{29}$$

The angles can then be obtained using:

$$\begin{aligned} \theta_1 &= \arg(z_+ \, z_-^*) = \arg((a + ib)(c - id)) \\ \theta_3 &= \arg(z_+ \, z_-) = \arg((a + ib)(c + id)) \end{aligned} \tag{30}$$

Or, more simply, from Eq. 25:

$$\begin{aligned} \theta_1 &= \arg(a + ib) - \arg(c + id) \\ \theta_3 &= \arg(a + ib) + \arg(c + id) \end{aligned} \tag{31}$$

Or:

$$\begin{aligned} \theta_1 &= \theta_+ - \theta_- \\ \theta_3 &= \theta_+ + \theta_- \end{aligned} \tag{32}$$

It is worth noting that Eq. 32 requires fewer operations than Eq. 30: only 2 calls to atan2, one addition and one subtraction, but a final wrapping step may be required in order to either keep the angles either in $(-\pi, \pi]$ or $[0, 2\pi)$.

3.1.4 General formulas for calculating θ_2

From Eq. 24, we know that:

$$\begin{aligned} c_2 &= \cos(\theta_2/2) = |z_+| = \sqrt{a^2 + b^2} \\ s_2 &= \sin(\theta_2/2) = |z_-| = \sqrt{c^2 + d^2} \end{aligned} \tag{33}$$

And we can use any of the following equivalent formulas obtained directly from the definition:

$$\theta_2 = 2 \operatorname{asin}\left(\sqrt{\frac{c^2 + d^2}{n^2}}\right) = 2 \operatorname{acos}\left(\sqrt{\frac{a^2 + b^2}{n^2}}\right) = 2 \operatorname{atan}\left(\sqrt{\frac{c^2 + d^2}{a^2 + b^2}}\right) \tag{34}$$

Where the factor $n^2 = a^2 + b^2 + c^2 + d^2 = |q|^2$ can be ignored if the quaternion is already normalized. Using the properties of inverse trigonometric functions, we can also find the following formula, which avoids the need for a square root:

$$\theta_2 = \operatorname{acos}\left(2 \, \frac{a^2 + b^2}{n^2} - 1\right) \tag{35}$$

3.2 Case 2: Tait-Bryan angles

We now define:

$$q = \begin{bmatrix} c_3 \\ s_3\boldsymbol{e}'' \end{bmatrix} \odot \begin{bmatrix} c_2 \\ s_2\boldsymbol{e}' \end{bmatrix} \odot \begin{bmatrix} c_1 \\ s_1\boldsymbol{e} \end{bmatrix} \tag{36}$$

Where $-\pi/2 < \phi_2 < \pi/2$. Again assuming that $\boldsymbol{e}$, $\boldsymbol{e}'$ and $\boldsymbol{e}''$ are orthogonal unit vectors and $\boldsymbol{e}'' = \varepsilon\,\boldsymbol{e} \times \boldsymbol{e}'$, where $\varepsilon = (\boldsymbol{e} \times \boldsymbol{e}') \cdot \boldsymbol{e}'' = \pm 1$, we define:

$$\lambda \equiv \begin{bmatrix} \cos(\pi/4) \\ \sin(\pi/4)\boldsymbol{e}' \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ \boldsymbol{e}' \end{bmatrix} \tag{37}$$

We note that:

$$\begin{aligned} \lambda^* \odot \begin{bmatrix} c_3 \\ s_3\varepsilon\boldsymbol{e} \end{bmatrix} \odot \lambda &= \begin{bmatrix} c_3 \\ s_3\varepsilon\boldsymbol{e} \times \boldsymbol{e}' \end{bmatrix} \\ &= \begin{bmatrix} c_3 \\ s_3\boldsymbol{e}'' \end{bmatrix} \end{aligned} \tag{38}$$

Which gives:

$$\begin{aligned} q &= \begin{bmatrix} c_3 \\ s_3\boldsymbol{e}'' \end{bmatrix} \odot \begin{bmatrix} c_2 \\ s_2\boldsymbol{e}' \end{bmatrix} \odot \begin{bmatrix} c_1 \\ s_1\boldsymbol{e} \end{bmatrix} \\ q &= \lambda^* \odot \begin{bmatrix} c_3 \\ s_3\varepsilon\boldsymbol{e} \end{bmatrix} \odot \lambda \odot \begin{bmatrix} c_2 \\ s_2\boldsymbol{e}' \end{bmatrix} \odot \begin{bmatrix} c_1 \\ s_1\boldsymbol{e} \end{bmatrix} \\ q' &= \begin{bmatrix} c_3' \\ s_3'\boldsymbol{e} \end{bmatrix} \odot \begin{bmatrix} c_2' \\ s_2'\boldsymbol{e}' \end{bmatrix} \odot \begin{bmatrix} c_1 \\ s_1\boldsymbol{e} \end{bmatrix} \end{aligned} \tag{39}$$

Where:

$$\begin{aligned} s_2' &\equiv \sin\theta_2'/2\ (\geq 0) \\ c_2' &\equiv \cos\theta_2'/2\ (\geq 0) \\ s_3' &\equiv \sin\theta_3'/2 \\ c_3' &\equiv \cos\theta_3'/2 \end{aligned} \tag{40}$$

Where $\theta_2' = \theta_2 + \pi/2$ and $\theta_3' = \varepsilon\theta_3$, and:

$$\begin{aligned} q' &\equiv \lambda \odot q \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ \boldsymbol{e}' \end{bmatrix} \odot \begin{bmatrix} a \\ b\boldsymbol{e} + c\boldsymbol{e}' + d\boldsymbol{e}'' \end{bmatrix} \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} a - c \\ (b+d)\boldsymbol{e} + (c+a)\boldsymbol{e}' + (d-b)\boldsymbol{e}'' \end{bmatrix} \end{aligned} \tag{41}$$

3.2.1 General formula

Using Eq. 41, we can define:

$$\begin{bmatrix} a' \\ b' \\ c' \\ d' \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} a - c \\ b + d \\ c + a \\ d - b \end{bmatrix} \tag{42}$$

And then calculate θ_1, θ'_2 and θ'_3 using the formulas obtained in the proper case. Using the acos formula for θ_2, we have:

$$\begin{aligned}\theta_2 &= \theta'_2 - \pi/2 \\ &= \operatorname{acos}\left(2\,\frac{a'^2 + b'^2}{n'^2} - 1\right) - \pi/2\end{aligned} \tag{43}$$

Which results in singularities when $\theta'_2 = 0$ or $\theta'_2 = \pi$, which is equivalent to $\theta_2 = -\pi/2$ or $\theta_2 = \pi/2$, as was to be expected. In addition, we know that when no singularities are present:

$$\begin{aligned}\theta_1 &= \operatorname{atan2}(b', a') - \operatorname{atan2}(d', c') \\ \theta_3 &= \varepsilon\left(\operatorname{atan2}\left(b', a'\right) + \operatorname{atan2}\left(d', c'\right)\right)\end{aligned} \tag{44}$$

3.3 Example of a proper sequence: the sequence ZYZ

If we decide to use the sequence ZYZ, then $\boldsymbol{e} = \boldsymbol{e}_z$, $\boldsymbol{e}' = \boldsymbol{e}_y$ and $\boldsymbol{e}'' = \boldsymbol{e}_z \times \boldsymbol{e}_y = -\boldsymbol{e}_x$. This leads to:

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} q_\mathrm{r} \\ q_\mathrm{z} \\ q_\mathrm{y} \\ -q_\mathrm{x} \end{bmatrix} \tag{45}$$

And the general formulas for θ_1, θ_2 and θ_3 (when no singularities are present, and assuming q to have been normalized) are:

$$\begin{aligned}\theta_1 &= \operatorname{atan2}(q_\mathrm{z}, q_\mathrm{r}) - \operatorname{atan2}(-q_\mathrm{x}, q_\mathrm{y}) \\ \theta_2 &= \operatorname{acos}\left(2\,\left(q_\mathrm{r}^2 + q_\mathrm{z}^2\right) - 1\right) \\ \theta_3 &= \operatorname{atan2}(q_\mathrm{z}, q_\mathrm{r}) + \operatorname{atan2}(-q_\mathrm{x}, q_\mathrm{y})\end{aligned} \tag{46}$$

3.4 Second example: the sequence XYZ

Using the sequence XYZ, equivalent to the common aeronautical angles, then $\boldsymbol{e} = \boldsymbol{e}_x$, $\boldsymbol{e}' = \boldsymbol{e}_y$ and $\boldsymbol{e}'' = \boldsymbol{e}_z$. This leads to:

$$\begin{bmatrix} a' \\ b' \\ c' \\ d' \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} q_\mathrm{r} - q_\mathrm{y} \\ q_\mathrm{x} + q_\mathrm{z} \\ q_\mathrm{y} + q_\mathrm{r} \\ q_\mathrm{z} - q_\mathrm{x} \end{bmatrix} \tag{47}$$

And the general formulas for θ_1, θ_2 and θ_3 are:

$$\begin{aligned}\theta_1 &= \operatorname{atan2}(q_\mathrm{x} + q_\mathrm{z}, q_\mathrm{r} - q_\mathrm{y}) - \operatorname{atan2}(q_\mathrm{z} - q_\mathrm{x}, q_\mathrm{y} + q_\mathrm{r}) \\ \theta_2 &= \operatorname{acos}\left(\left(q_\mathrm{r} - q_\mathrm{y}\right)^2 + \left(q_\mathrm{x} + q_\mathrm{z}\right)^2 - 1\right) - \pi/2 \\ \theta_3 &= \operatorname{atan2}(q_\mathrm{x} + q_\mathrm{z}, q_\mathrm{r} - q_\mathrm{y}) + \operatorname{atan2}(q_\mathrm{z} - q_\mathrm{x}, q_\mathrm{y} + q_\mathrm{r})\end{aligned} \tag{48}$$

4 Complete algorithm

Algorithm 1, presented in this section, implements the conversion method from this work. Assuming that our inputs are $q \in \mathbb{R}^4$, the rotation quaternion and i, j and $k \in \mathbb{N}$, an array of integers defining the sequence of angles (for example, $[i, j, k] = [323]$ is equivalent to the sequence ZYZ). A Python implementation can be found on [9].

Input: $q \in \mathbb{R}^4$, and $i, j, k \in \{1, 2, 3\}$, where $i \neq j, j \neq k$
Output: $\theta_1, \theta_2, \theta_3$
if $i == k$ **then**
  $not\_proper \leftarrow$ False
  $k \leftarrow 6 - i - j$ // because $i + j + k = 1 + 2 + 3 = 6$
else
  $not\_proper \leftarrow$ True
end
$\varepsilon \leftarrow (i - j) \times (j - k) \times (k - i)/2$ // equivalent to the Levi-Civita symbol
if not_proper **then**
  $a \leftarrow q[0] - q[j]$
  $b \leftarrow q[i] + q[k] \times \varepsilon$
  $c \leftarrow q[j] + q[0]$
  $d \leftarrow q[k] \times \varepsilon - q[i]$
else
  $a \leftarrow q[0]$
  $b \leftarrow q[i]$
  $c \leftarrow q[j]$
  $d \leftarrow q[k] \times \varepsilon$
end
$\theta_2 \leftarrow \text{acos}\left[2\left(\frac{a^2+b^2}{a^2+b^2+c^2+d^2}\right) - 1\right]$
$\theta^+ \leftarrow \text{atan2}(b, a)$
$\theta^- \leftarrow \text{atan2}(d, c)$
switch *value of* θ_2 **do**
  case 0 **do**
    $\theta_1 \leftarrow 0$ // For simplicity, we are setting $\hat{\theta}_1 = 0$
    $\theta_3 \leftarrow 2 \times \theta^+ - \theta_1$
  case $\pi/2$ **do**
    $\theta_1 \leftarrow 0$
    $\theta_3 \leftarrow 2 \times \theta^- + \theta_1$
  otherwise do
    $\theta_1 \leftarrow \theta^+ - \theta^-$
    $\theta_3 \leftarrow \theta^+ + \theta^-$
  end
end
if not_proper **then**
  $\theta_3 \leftarrow \varepsilon \times \theta_3$
  $\theta_2 \leftarrow \theta_2 - \pi/2$
end
$\theta_1, \theta_3 \leftarrow \texttt{wrap}(\theta_1, \theta_3)$ // ''wrap'' assures $\theta_1, \theta_3 \in (-\pi, \pi]$ or $\theta_1, \theta_3 \in [0, 2\pi)$

Algorithm 1: Complete implementation of conversion between a rotation quaternion and Euler angles in any sequence, setting $\theta_1 = 0$ in case of singularity.

Many operations are required to convert a quaternion into a rotation matrix. Using

the homogeneous formula from Eq. 11, for example, if special care is taken in order not to repeat any operations, we have to perform at least $4^2 = 16$ floating point multiplications (all the possible products between two different components of the quaternion, plus all the squares of each component), $4 \times 3 = 12$ multiplications by 2 and $3 \times 3 + 6 = 15$ additions/subtractions. This conversion step alone is more than enough to make an algorithm based on [5] much slower than the proposed method. In addition, multiple matrix multiplications also have to be computed. By comparison, our algorithm replaces all the conversions and matrix multiplications by a simple permutation of the quaternion elements and in the case of Tait-Bryan angles, only 5 additional additions/subtractions and possibly a sign change are required.

5 Results

In this section, a performance comparison between our method and the Shuster method is presented. We adapted the SciPy library in order to compile the algorithm as described in Section 4. A real data set comprising the orientation of a spinning object with 3284 data points was used to compare the efficiency of the two algorithms. The full implementation and data set can be downloaded from [9]. First we noted that both methods give the same results: adding the absolute value of the differences between the two methods in a whole data set gives an error of the order of 10^{-12}. The execution times required in our tests for each sequence (and their ratios) are presented in the Table 1. From this test, it can be clearly seen that the method presented here is about 30 times faster.

Table 1. Comparison of execution times between the two methods. The Python module *timeit* was used to check the execution time required to convert the whole data set 500 times on an Intel® Core™ i3-4030U CPU with a 1.90GHz clock speed.

seq	new method	[5] implemented in [7]	ratio
ZYZ	0.487 s	13.770 s	28.261
ZXZ	0.384 s	13.361 s	34.805
XYX	0.382 s	13.381 s	34.998
XZX	0.414 s	13.187 s	31.832
YXY	0.359 s	13.029 s	36.262
YZY	0.375 s	13.078 s	34.884
ZYX	0.371 s	13.152 s	35.408
ZXY	0.364 s	13.124 s	36.048
XYZ	0.373 s	13.170 s	35.291
XZY	0.385 s	13.157 s	34.213
YXZ	0.365 s	13.087 s	35.838
YZX	0.425 s	13.122 s	30.844

6 Conclusion

The Euler angles are still a useful intuitive 3D orientation parametrization. A fast method of conversion to/from any other set of parameters can therefore be of great interest for displaying or analyzing data, for instance. In this study, we therefore developed a general formula for this conversion which is concise, easy to implement and easy to debug. In addition, the fact that our method is about 30 times faster than the method proposed by [5], which required an intermediate conversion into rotation matrices, we believe that our proposed method can be of great interest. This faster

execution time also makes this method suitable for use in embedded real time applications such as inertial measurement units (IMUs). We propose that this method could be adopted as the new standard method for converting quaternions into Euler angles, and we are now planning to contributing to several scientific libraries accordingly. Moreover, a possible further development is to generalize this formula for the Davenport angles [6], a generalization of the Euler angles in which any set of distinct non-orthogonal axes are used.

Acknowledgments

We thank J. Blanc for the English improvement.

References

1. Shuster M. Survey of attitude representations. Journal of the Astronautical Sciences. 1993;41(4):439–517.

2. Sarabandi S, Shabani A, Porta JM, Thomas F. On Closed-Form Formulas for the 3-D Nearest Rotation Matrix Problem. IEEE Transactions on Robotics. 2020;36(4):1333–1339. doi:10.1109/TRO.2020.2973072.

3. Stuelpnagel J. On the Parametrization of the Three-Dimensional Rotation Group. Siam Review. 1964;6(4):422–430.

4. Henderson D. Euler Angles, Quaternions, and Transformation Matrices. NASA JSC Report. 1977; p. 42.

5. Shuster M, Markley L. General Formula for Extracting the Euler Angles. Journal of Guidance Control and Dynamics. 2006;29(1):215–221. doi:10.2514/1.16622.

6. Shuster M, Markley L. Generalization of the Euler Angles. Journal of the Astronautical Sciences. 2003;51(2):132–123. doi:10.1007/BF03546304.

7. Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods. 2020;17:261–272. doi:10.1038/s41592-019-0686-2.

8. Sommer H, Gilitschenski I, Bloesch M, Weiss S, Siegwart R, Nieto J. Why and how to avoid the flipped quaternion multiplication. Aerospace. 2018;5(3):1–15. doi:10.3390/aerospace5030072.

9. Quaternion to Euler Scipy implementation; 2022. `https://github.com/evbernardes/quaternion_to_euler`.