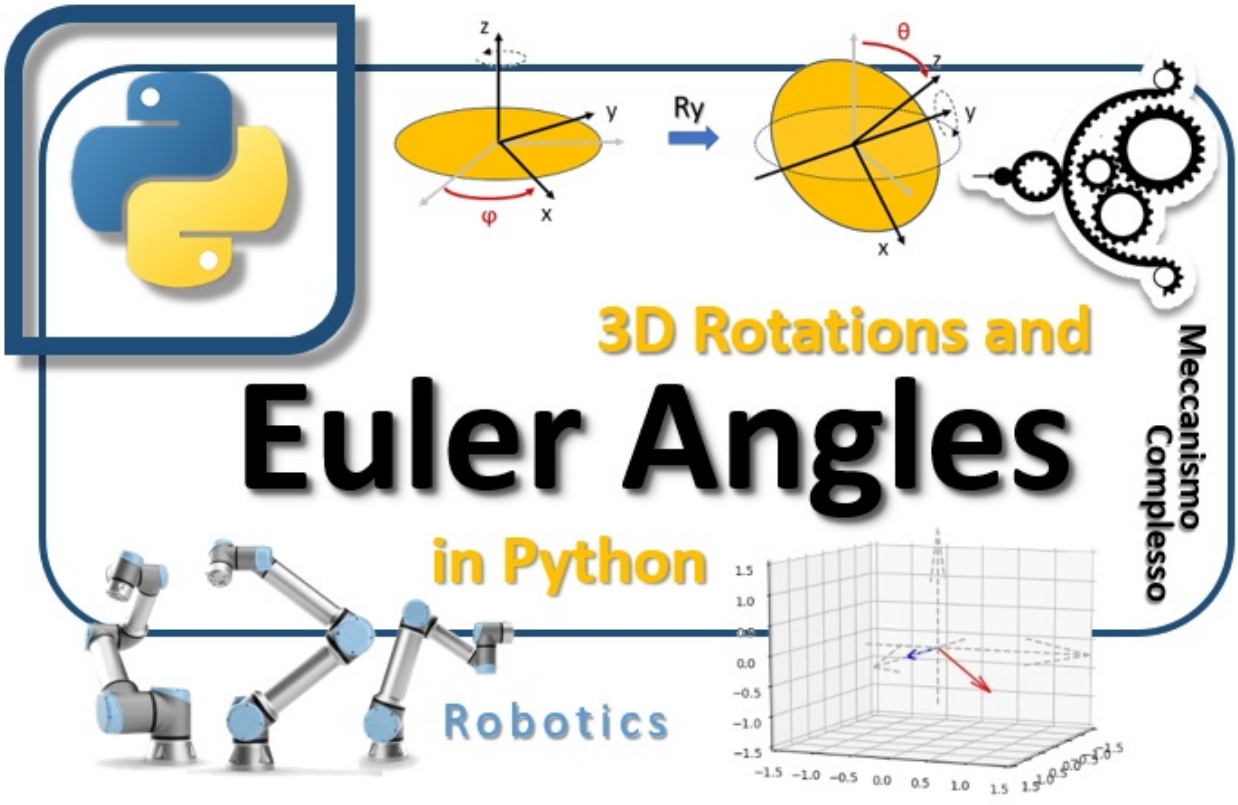
3D Rotations and Euler angles in Python

meccanismocomplesso.org/en/3d-rotations-and-euler-angles-in-python

webmaster 7 September 2020

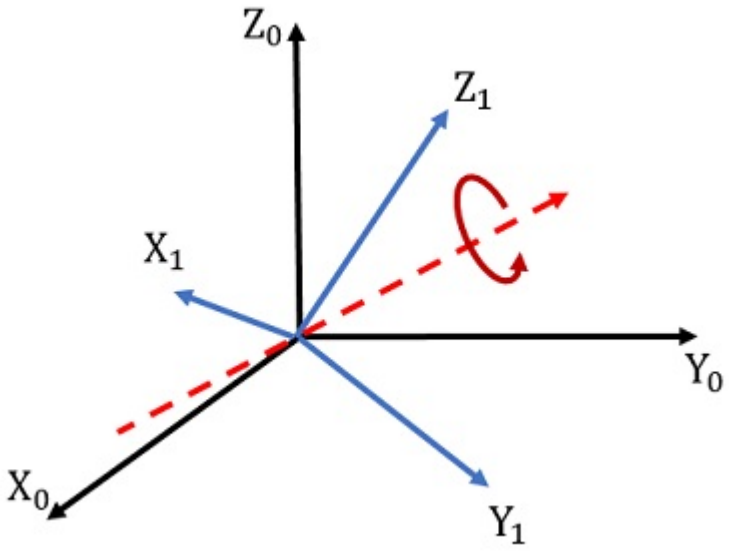


Euler's theorem

"Each movement of a rigid body in three-dimensional space, with a point that remains fixed, is equivalent to a single rotation of the body around an axis passing through the fixed point"

This theorem was formulated by Euler in 1775.

In other words, if we consider two Cartesian reference systems, one (X_0,Y_0,Z_0) and the other (X_1,Y_1,Z_1) which have the same origin point O, but different orientation, there will always be a single axis of rotation with which the first system will assume the same configuration as the second system.

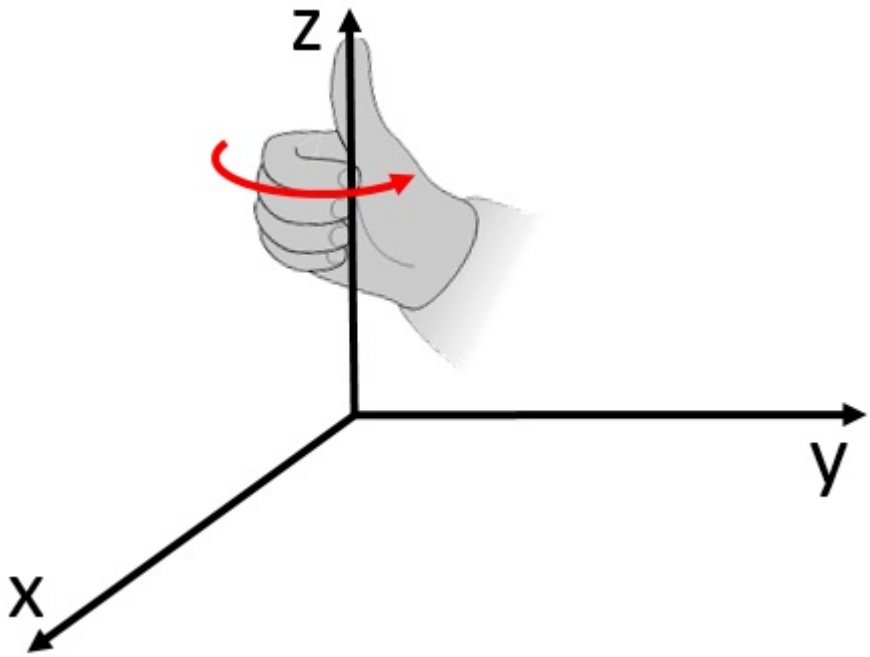


In an even simpler way, any rotation can be described by a sequence of three successive rotations, also called **elementary rotations**, which occur around one of the three coordinate axes X, Y and Z

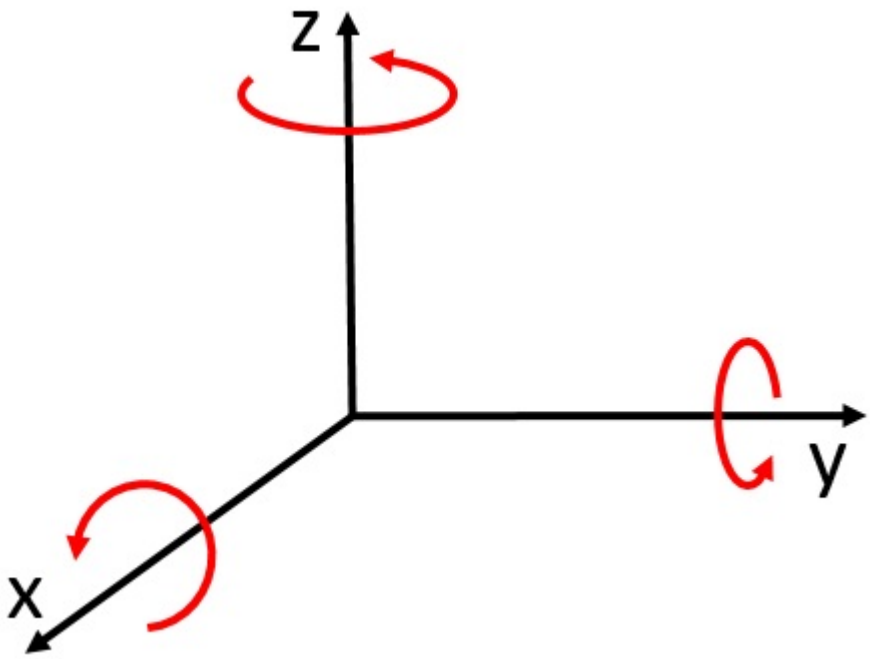
The elementary rotations

There are therefore three elementary rotations, each around its Cartesian reference axis X, Y and Z. But the rotation around an axis can occur in two opposite directions. But which of the two is the positive one?

Rotation around an axis is positive if it meets the **right hand rule**. For example in the case of rotation around the z axis, the rotation will be positive depending on the arrangement of the X and Y axes in the representation.



Therefore the direction of the three elementary rotations will be the one shown in the following figure.



Each elementary rotation can be transcribed as a 3×3 matrix (**homogeneous transformation**).

Rotation on the X axis

$$Rx = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$$

Rotation on the Y axis

$$Ry = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

Rotation on the Z axis

$$Rz = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Python

In Python, the matrix object of the numPy library exists to express matrices.

In fact, it can be tempting to use the more common **np.array**. But even though the declarations of **np.array** objects from **np.matrix** look very similar, their behavior can be very different in many contexts.

Here are the three elementary rotations:

```
import numpy as np
import math as m

def Rx(theta):
return np.matrix([[ 1, 0           , 0           ],
[ 0, m.cos(theta),-m.sin(theta)],
[ 0, m.sin(theta), m.cos(theta)]])

def Ry(theta):
return np.matrix([[ m.cos(theta), 0, m.sin(theta)],
[ 0           , 1, 0           ],
[-m.sin(theta), 0, m.cos(theta)]])

def Rz(theta):
return np.matrix([[ m.cos(theta), -m.sin(theta), 0 ],
[ m.sin(theta), m.cos(theta) , 0 ],
[ 0           , 0            , 1 ]])


```

Euler's angles

Therefore a generic rotation is described in turn by a **rotation matrix R.** Any matrix of this type can be described as the product of successive rotations around the principal axes of the XYZ coordinates, taken in a precise order.

So any rotation could be decomposed into the sequence of three elementary matrices. For example, the most intuitive is that which is obtained first by performing a rotation on the X axis by an angle φ, then on the Y axis by an angle θ and finally on the Z axis by an angle ψ

The triplet of the angles used in these elementary rotations are the **Euler angles** and are normally indicated (φ, θ, ψ).

$$Rx(\varphi) \rightarrow Ry(\theta) \rightarrow Rz(\psi)$$

Let's take an example in Python. We choose three euler angles and then we multiply the elementary rotation matrices R ZYZ

```
phi = m.pi/2
theta = m.pi/4
psi = m.pi/2
print("phi =", phi)
print("theta  =", theta)
print("psi =", psi)


R = Rz(psi) * Rv(theta) * Rx(phi)
print(np.round(R, decimals=2))
```

By executing we will obtain the following rotation matrix

```
[[ 0. -0. 1. ]
[ 0.71 0.71 -0. ]
[-0.71 0.71 0. ]]
```

But it is also possible to perform the reverse operation. That is, knowing the rotation matrix, it is possible to derive the three Euler angles.

```
import sys
tol = sys.float_info.epsilon * 10

if abs(R.item(0,0))< tol and abs(R.item(1,0)) < tol:
eul1 = 0
eul2 = m.atan2(-R.item(2,0), R.item(0,0))
eul3 = m.atan2(-R.item(1,2), R.item(1,1))
else:
eul1 = m.atan2(R.item(1,0),R.item(0,0))
sp = m.sin(eul1)
cp = m.cos(eul1)
eul2 = m.atan2(-R.item(2,0),cp*R.item(0,0)+sp*R.item(1,0))
eul3 = m.atan2(sp*R.item(0,2)-cp*R.item(1,2),cp*R.item(1,1)-
sp*R.item(0,1))

print("phi =", eul1)
print("theta  =", eul2)
print("psi =", eul3)
```

By executing we obtain the three angles, which are then the same ones we had inserted at the beginning.

```
phi = 1.5707963267948966
theta = 0.7853981633974483
psi = 1.5707963267948966
```

But the XYZ rotation sequence is only one of 12 possible combinations. there are different possible combinations of three elementary rotations, such as ZYX, ZYZ, XYX, etc. Each of these will have a different convention for expressing Euler angles.

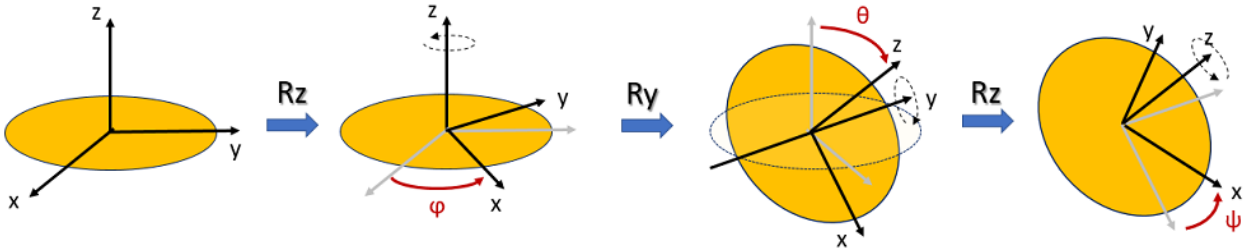
The ZYZ convention

In addition to the XYZ sequence, another very common one is the one that refers to the convention associated with the ZYZ angles characterized by the following operations:

- Rotation of an angle φ around the z axis
- Rotation of an angle θ around the y axis (current)
- Rotation of an angle ψ around the z axis (current)

$$Rz(\varphi) \rightarrow Ry(\theta) \rightarrow Rz(\psi)$$

The order of the elementary rotations changes the final result.



Where also here the angles φ, θ and ψ are the Euler angles.

For the ZYZ convection, the Euler angles have a particular nomenclature:

- **φ** is the Precession angle with values [0, 2π]
- **θ** is the angle of nutation
- **ψ** is the angle of proper rotation

Each of the three rotations can be represented mathematically by a rotation matrix. The matrix relating to the overall rotation is calculated by multiplying the 3 matrices in the reverse order.

Therefore, by multiplying in the reverse order we obtain the matrix relating to the overall rotation:

158/5000We also see this case in Python. We reinsert the same three Euler angles and multiply the three elementary rotation matrices in the right sequence

$$Rzyz = RzRyRz$$

```
phi = m.pi/2
theta = m.pi/4
psi = m.pi/2
print("phi =", phi)
print("theta  =", theta)
print("psi =", psi)


R = Rz(psi) * Rv(theta) * Rz(phi)
print(np.round(R, decimals=2))
```

We will get the following rotation matrix

```
[[-1. -0. 0. ]
[ 0. -0.71 0.71]
[-0. 0.71 0.71]]]
```

Here, too, you can perform the reverse operation. That is, knowing the general rotation matrix, obtain the three Euler angles. Since the condition is different, the mathematical expressions to derive them are also different.

```
eul1 = m.atan2(R.item(1,2),R.item(0,2))
sp = m.sin(eul1)
cp = m.cos(eul1)
eul2 = m.atan2(cp*R.item(0,2)+sp*R.item(1,2), R.item(2,2))
eul3 = m.atan2(-sp*R.item(0,0)+cp*R.item(1,0),-
sp*R.item(0,1)+cp*R.item(1,1))

print("phi =", eul1)
print("theta =", eul2)
print("psi =", eul3)
```

By executing the code above, we obtain the values of the three Euler angles, which then correspond precisely to those entered initially.

```
phi = 1.5707963267948966
theta = 0.7853981633974483
psi = 1.5707963267948966
```

The rotation of a point in space

Euler transformations with their relative angles are a wonderful tool for applying rotations of points in space. The simplest example of application of what we have already seen in the article is the rotation of a point located in a coordinate space (X, Y, Z).

A point in space can be represented by a 3-element vector that characterizes its values on the three coordinate axes.

$$v = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

So for our example, we will start from a simple point on the X axis described by the following vector.

$$v = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Which in Python is implemented

```
v1 = np.array([[1],[0],[0]])
```

If we want to apply a rotation at this point it will be sufficient to multiply this vector precisely with the rotation matrix and thus obtain another vector.

$$v2 = Rv1$$

Which in Python is implemented as follows.

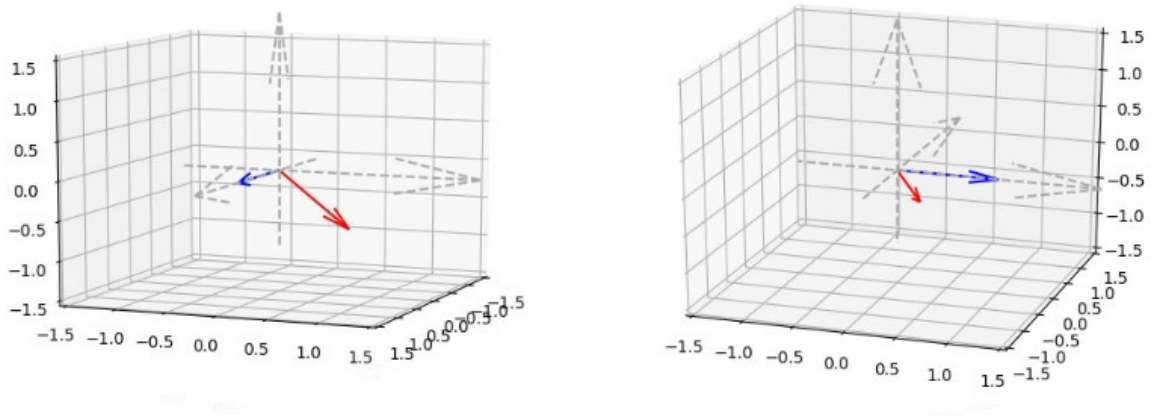
```
v2 = R * v1
print(np.round(v2, decimals=2))
```

We added the print () function to display the result of the rotation. The coordinates of the point in space after the rotation described by R will correspond to the values of the vector v2.

```
[[ 0. ]
[ 0.71]
[-0.71]]
```

Let's see the result of our rotation by plotting with Cartesian axes the position of the vector (which describes the point) before and after the rotation.

By running the code you will get the graphical representation.



Vector display before (blue) and after rotation (red)

From here you can then move on to the rotation of lines, geometric figures and three-dimensional objects. All at the basis of the 3D engines with which many video games are developed.

Limits of Euler's Representation

With these examples we have seen how with Euler angles it is possible to describe in a simple way any rotation in three-dimensional space. But there is a limitation to the use of Euler angles, which is often referred to with the term **Gimbal Lock**.

The technique we have seen is based on the use of a sequence of elementary rotations referring to one of the Cartesian axes at a time. By applying these rotations in sequence it can happen that one of the reference axes can collapse into another. For example with rotations of 90 degrees or 180 degrees.

For example, if we rotate 90 degrees around the X axis, the Y axis will collapse on the Z axis. In this situation, a degree of freedom is lost as the rotations around the Y, Z axes become equivalent.

Another drawback is that the angles depend on the sequence of the three rotations around the Cartesian axes and reported as the name of the convention: ZYZ, XZX, YXZ, etc. Each of these sequences gives a triplet of Euler angles with different values, as we have also verified above. Then the rotation matrix and the inverse formula will change accordingly.

Conclusions

Despite all these drawbacks, **Euler angles** are widely used today and are a very important reference point for those who work in the field of CAD modeling, 3D video game engines, and robotics and automation in general. However, in practice, a more complex but more effective mathematical model is often used, the **Hamilton quaternions**. In a future article we will learn how rotations in space are implemented with Hamilton quaternions, what they are and how to use them in Python.