9.7 Symptoms associated with RTOS stack problems

Most problems encountered with FreeRTOS relate to the configuration of stack and heap sizes in FreeRTOSConfig.h. If properly configured, the call to vTaskStartScheduler() in main() should never return.

9.8 Magnetic Interference

We've discussed magnetic interference numerous times in this text, but it bears repeating: You should assume that indoor magnetic environments vary significantly over even short distances. If your development board is near a laptop computer or telephone, it will encounter distorted fields. This is not a sensor problem, it is an environmental problem. Sensor fusion can help to alleviate, but not cure, this issue.

10. Theory: Orientation Representations

Sections 10.1 and 10.2 are variations of two postings (Orientation Representations, Part 1 & Part 2) which initially appeared on the Freescale (now NXP) Embedded Beat blog site in 2012. To have a "happy experience" integrating sensor fusion into an application, you need to understand the underlying theory presented here.

10.1 Part 1: Euler Angles and Rotation Matrices

10.1.1 Rotation Matrices

If you look at the software source code at the heart of these AR & VR apps, you'll discover a common component: a mathematical model relating the position and orientation of your portable device within an earth frame of reference. Perhaps the three most common are Euler angles, rotation matrices and quaternions. This section focuss on the Euler angles and rotation matrices. Quaternions will be covered in the next section.

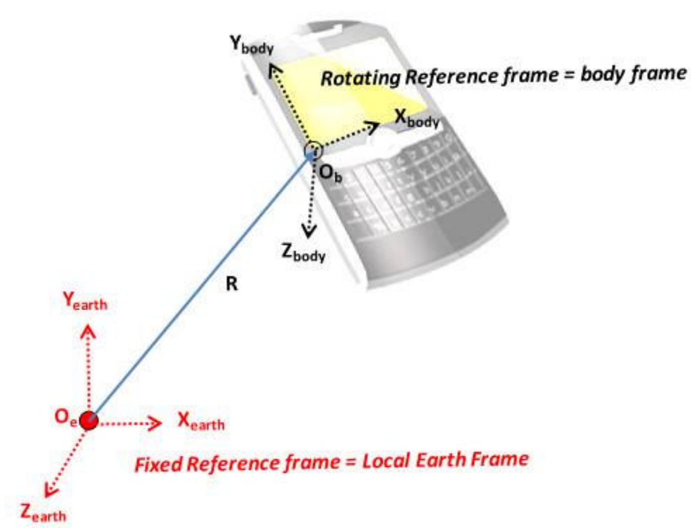


Fig 45. Local earth versus the body frame

Let's arbitrarily choose the phone screen X,Y coordinate system as our rotating frame of reference for this discussion. You can intuitively see that any X,Y point (Z assumed zero) on the phone screen will map to some X, Y, Z point in the earth frame of reference.

It is also clear that the Cartesian axes in the phone (or body) frame of reference will only rarely align themselves with the local earth frame axes. Since the position offset, R, doesn't affect orientation, we can collapse the figure down to that shown in the figure below.

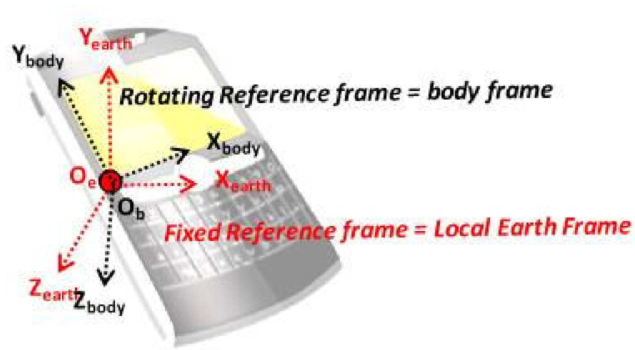


Fig 46. Using a common origin for both earth and body frames

Now let's remove the phone from the picture and focus just on points in the X-Y plane. The figure below shows the earth frame rotated into the body frame by an angle Psi (ψ).

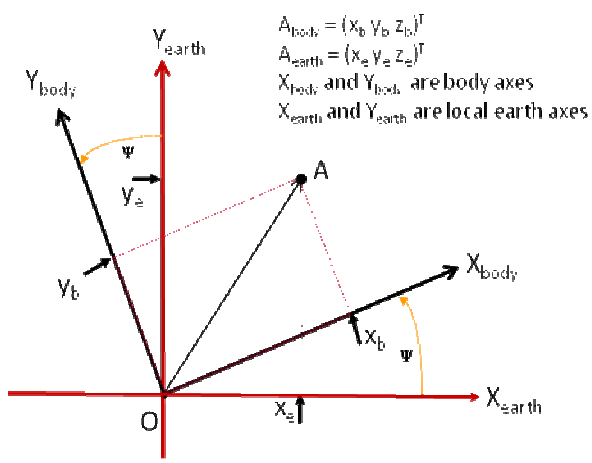


Fig 47. The X-Y plane illustrates rotation from earth into body frame about the Z axis

You probably first saw a figure like this in your high school geometry or trig course. It allows us to map any point "A" in any standard X,Y Cartesian system to any other X,Y Cartesian system, which is rotated from it by some angle Psi (ψ), with a simple linear transformation. To see how, let's deconstruct that figure using a number of right triangles.

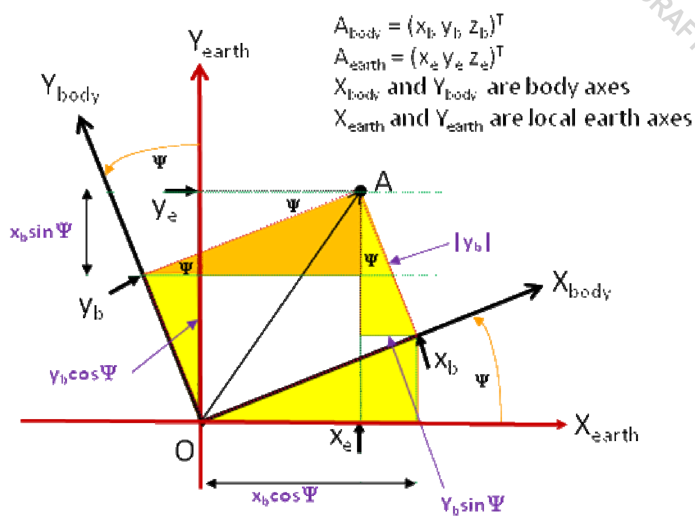


Fig 48. Physical justification for terms in the rotation matrix

If you study the figure above, you can see that the rotation angle, ψ, is present in each of the 4 right triangles we added. Additionally, the hypotenuse of each triangle is either x_b or y_b. Given that information, we can see how to compute x_e and y_e from x_b and y_b:

$$x_e = x_b \cos(Y) - y_b \sin(Y) \quad \text{(Eqn. 1)}$$

$$y_e = x_b \sin(Y) + y_b \cos(Y) \quad \text{(Eqn. 2)}$$

In matrix form:

$$\mathbf{A}_{earth} = \mathbf{C}(-\psi)\ \mathbf{A}_{body} \quad \text{(Eqn. 3)}$$

The inverse of which is:

$$\mathbf{A}_{body} = \mathbf{C}(\psi)\ \mathbf{A}_{earth} \quad \text{(Eqn. 4)}$$

where $\mathbf{A}_{earth}$ and $\mathbf{A}_{body}$ are of the form $[x\ y]^T$ and:

$$\mathbf{C}(\psi) = \mathbf{C}_\psi = \begin{bmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{bmatrix} \quad \text{(Eqn. 5)}$$

$$\mathbf{C}(-\psi) = \mathbf{C}_{-\psi} = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix} \quad \text{(Eqn. 6)}$$

All other relationships included in this discussion can be similarly mapped to a diagram of the rotation. The analysis also extends naturally to three dimensions. If $\mathbf{A}_{earth}$ and $\mathbf{A}_{body}$ are of the form $[x\ y\ z]^T$, then:

$$\mathbf{C}(\psi) = \mathbf{C}_\psi = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{(Eqn. 7)}$$

$$\mathbf{C}(-\psi) = \mathbf{C}_{-\psi} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{(Eqn. 8)}$$

We will use the three dimensional form in the remainder of this discussion. In both two and three dimensional cases, **C**(ψ) and **C**(-ψ) are known as *rotation matrices.*

Notice that **C**(ψ) is the transpose of **C**(-ψ), and vice-versa: $\mathbf{C}(\psi)^T = \mathbf{C}(-\psi)$. This is a special property of all rotation matrices. You can reverse the sense of rotation simply by taking the transpose of the original matrix.

$$\mathbf{C}(\psi)\ \mathbf{C}(-\psi) = \mathbf{I}_{3x3}$$

Where $\mathbf{I}_{3x3}$ is simply the identity matrix:

$$\mathbf{I}_{3x3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This also implies that the inverse of a rotation matrix is simply its own transpose:

$$\mathbf{C}(\psi)^{-1} = \mathbf{C}(\psi)^T$$

Similar relationships hold for rotations in the X-Z (about the Y axis) plane:

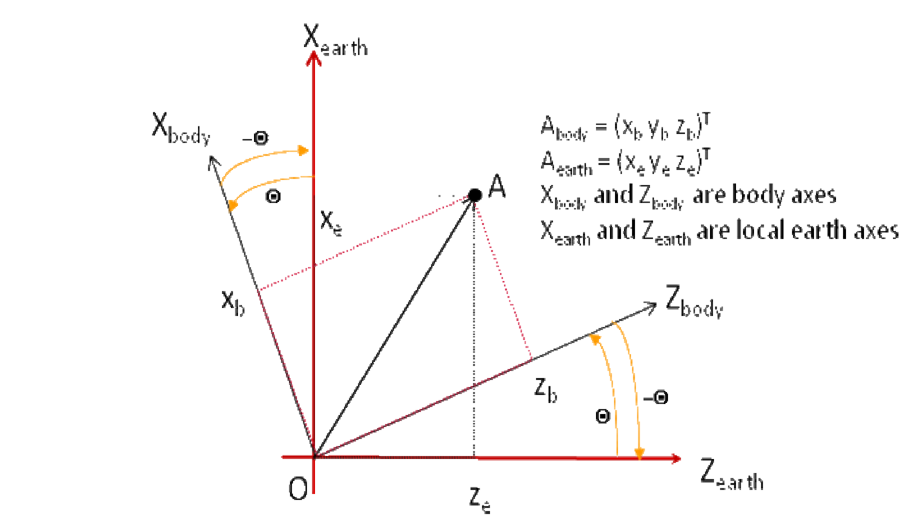


Fig 49. The X-Z plane illustrates rotation about the Y axis

$x_e = x_b \cos(\theta) + z_b \sin(\theta)$ (Eqn. 12)

$z_e = -\, x_b \sin(\theta) + z_b \cos(\theta)$ (Eqn. 13)

$\mathbf{A}_{earth} = \mathbf{C}(-\theta)\ \mathbf{A}_{body}$ (Eqn. 14)

$\mathbf{A}_{body} = \mathbf{C}(\theta)\ \mathbf{A}_{earth}$ (Eqn. 15)

$$\mathbf{C}(-\Theta) = \mathbf{C}_{-\theta} = \begin{bmatrix} cos\Theta & 0 & sin\Theta \\ 0 & 1 & 0 \\ -sin\Theta & 0 & cos\Theta \end{bmatrix} \quad \text{(Eqn. 16)}$$

$$\mathbf{C}_{\theta} = \begin{bmatrix} cos\Theta & 0 & -sin\Theta \\ 0 & 1 & 0 \\ sin\Theta & 0 & cos\Theta \end{bmatrix} \quad \text{(Eqn. 17)}$$

and for rotations in the Z-Y (about the X axis) plane:

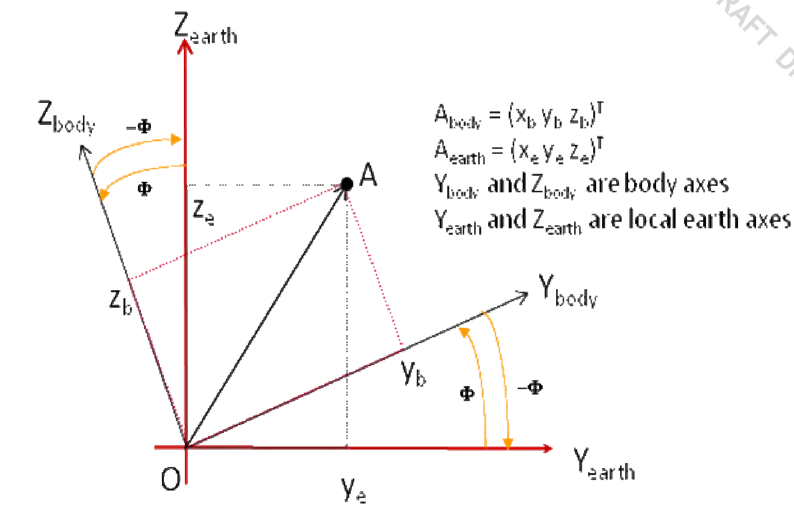


Fig 50. The Y-Z plane illustrates rotation about the X axis

$y_e = y_b \cos(\Phi) - z_b \sin(\Phi)$ (Eqn. 18)

$z_e = y_b \sin(\Phi) + z_b \cos(\Phi)$ (Eqn. 19)

$\mathbf{A}_{earth} = \mathbf{C}(-\Phi)\ \mathbf{A}_{body}$ (Eqn. 20)

$\mathbf{A}_{body} = \mathbf{C}(\Phi)\ \mathbf{A}_{earth}$ (Eqn. 21)

$$\mathbf{C}(-\Phi) = \mathbf{C}_{-\Phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\Phi & -sin\Phi \\ 0 & sin\Phi & cos\Phi \end{bmatrix}$$ (Eqn. 22)

$$\mathbf{C}_{\Phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\Phi & sin\Phi \\ 0 & -sin\Phi & cos\Phi \end{bmatrix}$$ (Eqn. 23)

Phi, Theta and Psi (Φ, θ and ψ) rotations <u>in sequence</u> can map a point in any right-hand-rule (RHR) 3-dimensional space into any other RHR 3-dimensional space.

A rotation from earth by body frame about Z, then Y then X axes (the "aerospace" sequence) is represented by:

$$\mathbf{C}_{\mathbf{RPY}} = \mathbf{C}_{\Phi}\mathbf{C}_{\Theta}\mathbf{C}_{\Psi} = \begin{bmatrix} cos\Psi cos\Theta & cos\Theta sin\Psi & -sin\Theta \\ sin\phi sin\Theta cos\Psi - cos\phi sin\Psi & sin\phi sin\Theta sin\Psi + cos\phi cos\Psi & sin\phi cos\Theta \\ cos\phi sin\Theta cos\Psi + sin\phi sin\Psi & cos\phi sin\Theta sin\Psi - sin\phi cos\Psi & cos\phi cos\Theta \end{bmatrix}$$

(Eqn 24)

The composite rotation matrix is computed simply by multiplying the three individual matrices in the specified order. Consistent with the discussion above, the inverse of this expression is:

$$\mathbf{C_{YPR}} = \mathbf{C_{-\Psi}C_{-\Theta}C_{-\Phi}} = \begin{bmatrix} cos\Psi cos\Theta & sin\phi sin\Theta cos\Psi - cos\phi sin\Psi & cos\phi sin\Theta cos\Psi + sin\phi sin\Psi \\ cos\Theta sin\Psi & sin\phi sin\Theta sin\Psi + cos\phi cos\Psi & cos\phi sin\Theta sin\Psi - sin\phi cos\Psi \\ -sin\Theta & sin\phi cos\Theta & cos\phi cos\Theta \end{bmatrix}$$

(Eqn. 25)

10.1.2 Euler Angles

Collectively, Φ, θ and ψ are known as ***Euler angles***. You may also see Φ, θ and ψ referred to as "roll", "pitch" and "yaw" respectively. The subscript "RPN" in the expression above refers to roll-pitch-yaw, and "YPR" refers to yaw-pitch-roll.

Euler angles are sometimes sub-divided into "Tait-Bryan" angles (in which rotations occur about all three axes) and "proper" Euler angles (in which the first and third axes of rotation are the same). Regardless of which type you use, it is important to specify the order of the rotations – which IS significant. In the table below, the right-most rotation is performed first, consistent with the matrix operations that will be required to implement the rotation. Possible variants are:

Alpha	Angles	Comments	
YRP	ψ-Φ-θ	Tait-Bryan angles (AKA Nautical or Cardan angles)	All of these are sometimes referred to simply as "Euler Angles"
YPR	ψ-θ-Φ		
PYR	θ-ψ-Φ		
PRY	θ-Φ-ψ		
RYP	Φ-ψ-θ		
RPY	Φ-θ-ψ		
RYR	Φ-ψ-Φ	Proper Euler Angles	
RPR	Φ-θ-Φ		
PYP	θ-ψ-θ		
PRP	θ-Φ-θ		
YRY	ψ-Φ-ψ		
YPY	ψ-θ-ψ		

A full discussion of Euler angles is beyond the scope of this article. But some key points you need to take away are:

- Again, order of rotation matters!
- There almost as many notations for Euler angles as the references in which they appear. Be specific in *your* notation.
- There may be multiple Euler angle combinations which map to the same physical rotation. They are not unique.

- There are typically limits on the range of Euler rotation for each axis (+/- π or +/- π/2). These affect how the total rotation is spread across the three angles. They also tend to introduce discontinuities when Euler angles are computed over time.
- Each Euler triad will map to 3x3 rotation matrix (which IS unique) in a manner similar to that shown in Equations 24 and 25 above.

If you define a "reference orientation" for any object, then you can define its current orientation as some rotation relative to that reference. Tracking orientation over time is then equivalent to tracking rotation from that reference over time. Because Euler angles are relatively easy to visualize, they enjoyed early popularity in a number of fields. But because of the shortcomings listed above, anyone who has used them extensively invariably learns to despise them.

Rotation matrices don't require a master's degree to be able to use them. Anyone with a college freshman geometry class under their belt can figure them out. They don't have the ambiguities associated with Euler angles, and can be found at the heart of many algorithms. But you need 9 numbers for each rotation in this form. That can chew up a lot of storage. A better representation is the "quaternion", which we will explore in the next section.

10.1.3 References:

[1] Quaternions and Rotation Sequences, Jack B. Kuipers, Princeton University Press, 1999

[2] Euler Angles from the Wolfram Demonstrations Project by Frederick W. Strauch

[3] Diversified Redundancy in the Measurement of Euler Angles Using Accelerometers and Magnetometers, Chirag Jagadish and Bor-Chin Chang, Proceedings of the 46th IEEE Conference on Decision and Control, Dec. 2007

[4] "Euler Angles" at Wikipedia

10.2 Part 2: Quaternions

10.2.1 Discussion

In the previous section, we explored the use of rotation matrices and Euler angles. At the end of that discussion, We alluded to the fact that there might be more efficient ways of describing rotations. Let's start with the rotation of a simple rigid body (in this case a cylinder) as shown in the figure below. Here, the cylinder is rotated such that a point on its surface originally at "A" is rotated to point "B" in space

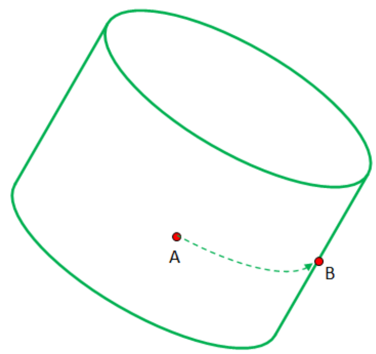


Fig 51. Rotation of a rigid body such that a reference point moves from “A” to “B”

For this simple case, we’ve kept the axis of rotation along the vertical axis of the cylinder as shown in the figure below. But that is not a requirement for the underlying mathematics to work. So long as we have a rigid body, we can always describe the rotation in the manner that follows.

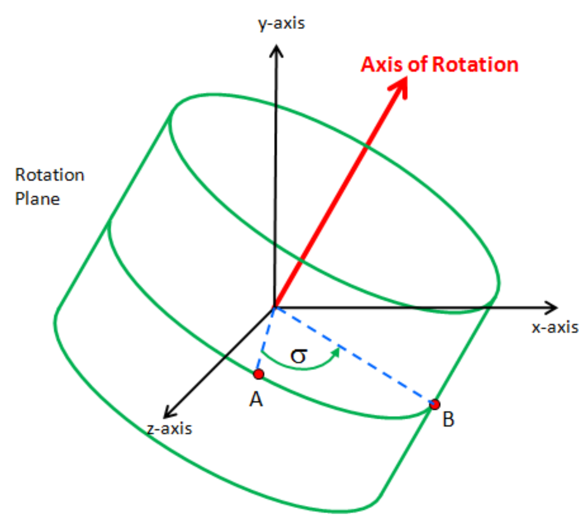


Fig 52. Overlay of Cartesian Coordinates onto System of Figure 1

Fig 53 deals with the same rotation, but focuses on the fact that we have a rotation plane that is perpendicular with the axis of rotation. The movement of the cylinder is a rotation equal of angle α, about the axis of rotation, where the point of interest is constrained to lie within the rotation plane.

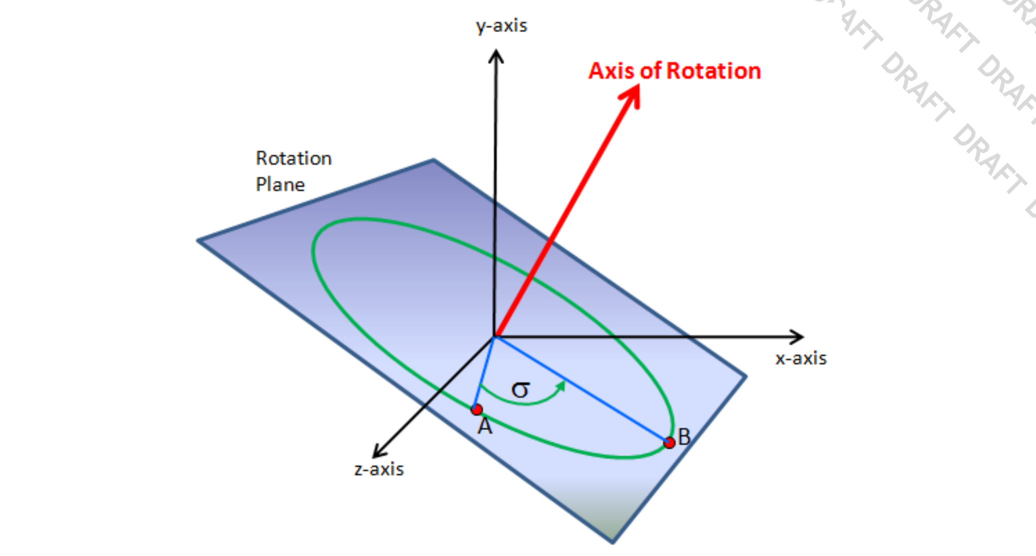


Fig 53. Looking at Just the Rotation Plane and Axis of Rotation

The rotation is fully described by the three components of the normalized rotation axis and the rotation angle α, which may be in radians or degrees, depending upon the system in use. As an example, consider OpenGL ES graphics programming. This system is very popular on portable devices. We used it to program the **Device** screen in the Sensor Fusion Toolbox for Android. In OpenGL ES, you build up 3 dimensional objects as a collection of triangles, which can then be offset and/or rotated to change perspective. As an example, our cylinder might be crudely drawn as shown in Figure 4.

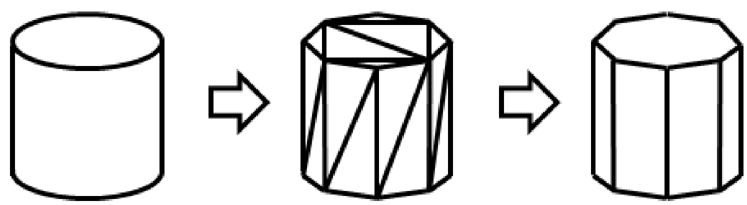


Fig 54. OpenGL ES Drawing of a Cylinder

In this case, We've modeled the top and bottom of the cylinder with 6 triangles each, and the other side is modeled using a total of 16 triangles arranged in a strip. OpenGL ES is optimized to draw such structures efficiently, and it is possible to then "render" textures onto the drawn surfaces. What's really neat is that once drawn, we get a reasonable approximation of the cylinder of on the left of Fig 54 simply by doing a -30 degrees rotation about the Z axis (presumed to be out of the page) using a single OpenGL ES instruction:

```
gl.glRotatef(-30.0f, 0.0f, 0.0f, 1.0f);
```

At this point, you're probably thinking: "Yeah, that makes sense, but how does it work at the math level?" This is the where we need to introduce the concept of a quaternion. Conceptually, a quaternion encodes the same axis and angle as above. But for mathematical reasons it deals with 1/2 of the rotation angle as shown below.

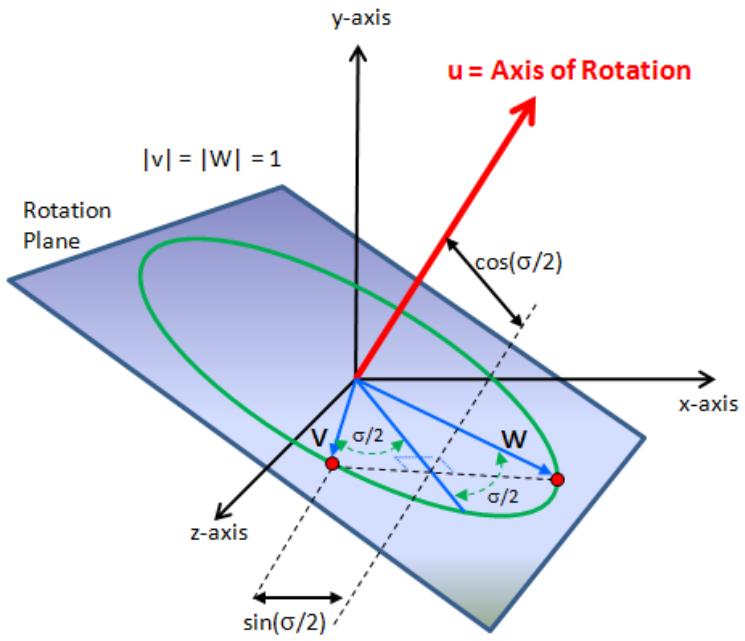


Fig 55. System of Figure 4 in Terms of Quaternion Components

Before overwhelming you with the underlying math, you should know that unless you are planning to implement your own quaternion utility library, you only need to know a few key points:

1. It takes four numbers to fully describe a quaternion (commonly q0 through q3).
2. Not all quaternions are rotation quaternions. Rotation quaternions have unit length ($q0^2 + q1^2 + q2^2 + q3^2 = 1$). The discussion below will be restricted to rotation quaternions.
3. These same rotations can be described using Euler angles, rotation matrices, etc. as discussed in the previous section. It is possible (and common) to translate between formats and use multiple formats. Rotation matrices have the advantage of always being unique. Euler angles are subject to gimbal lock, and should not be used for internal calculations (only input/output of results).
4. You can rotate a vector $\mathbf{V}$ using a quaternion q using the equation: $\mathbf{W} = q\mathbf{V}q^*$ (quaternion products and complex conjugates are defined later)
5. A sequence of rotations represented by quaternions q1 followed by q2 can be collapsed into a single rotation simply by computing the quaternion product q=q2*q1 and then appling the rotation operator as above.

We will be presenting the mathematical definition first, and without proof. If you really, REALLY want to know the underlying theory, let us suggest that you pick up a copy of Jack Kuiper's excellent text: Quaternions and Rotation Sequences. This appears to be (by far) the most extensive treatment on the topic, even while remaining very readable.

Notice that rotation quaternions deal with α/2, not α. We can define a rotation quaternion "q" in one of several equivalent fashions.

$q = (q0, q1, q2, q3)$ (Eqn.1)

$q = q0 + \mathbf{q}$, where $\mathbf{q} = \mathbf{i}q1 + \mathbf{j}q2 + \mathbf{k}q3$ (Eqn. 2)

$q = \cos(\alpha/2) + u \sin(\alpha/2)$, where u is the vector axis of rotation (Eqn. 3)

We use the quaternion form where q0 = cos(α/2). Some texts will reorder the quaternion components so that the vector portion **q** is contained in q0-2 and q3 = cos(α/2). Be sure you understand which form your text/software library supports.

Quaternions are a form of hyper-complex number where instead of a single real and single imaginary component, we have one real and THREE imaginary components (i, j & k). Rules for these imaginary components are:

$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$ (Eqn. 4)

$\mathbf{ij} = \mathbf{k} = -\mathbf{ji}$ (Eqn. 5)

$\mathbf{jk} = \mathbf{i} = -\mathbf{kj}$ (Eqn. 6)

$\mathbf{ki} = \mathbf{j} = -\mathbf{ik}$ (Eqn. 7)

Two quaternions, p and q, are equal to one another only if the individual components are equal. You add two quaternions by adding the individual components. If

$p = p0 + ip1 + jp2 + kp3$; and (Eqn. 8)

$q = q0 + iq1 + jq2 + kq3$ (Eqn. 9)

Then

$p + q = (p0 + q0) + \mathbf{i}(p1+q1) + \mathbf{j}(p2+q2) + \mathbf{k}(p3+q3)$ (Eqn. 10)

The addition operation commutes. That is p+q = q+p. Multiplication of a quaternion by a scalar real number is trivial, just multiply each of the four components by the scalar. Multiplication of two quaternions is NOT so trivial:

$pq = p0q0 – \mathbf{p}.\mathbf{q} + p0\mathbf{q} + q0\mathbf{p} + \mathbf{p} \times \mathbf{q}$ (Eqn. 11)

Multiplying one quaternion by another quaternion results in a third quaternion. Notice that the 1st two components (p0q0 – p.**q**) makes up the scalar portion of the result, and the last three (p0**q** + q0**p** + **p** x **q**) comprise the vector portion. The quaternion product operation is not commutative pq≠qp. Order matters. Multiplication of two quaternions includes scalar, cross product and dot product terms. Unless you are writing your own quaternion library, you are likely never to use the expression above. Instead, you will use a function that does the quaternion multiplication for you.

The complex conjugate of

q = q0 + **i**q1 + **j**q2 + **k**q3 is q* = q0 – **i**q1 – **j**q2 – **k**q3 (Eqn. 12)

Related to this, we have

(pq)* = q*p* (Eqn. 13)

q+q* = 2q0 (Eqn. 14)

q-1 = q* for any unit quaternion (Eqn. 15)

Eqn. 15 is interesting. If you think of a quaternion as a rotation operator, it says you can reverse the sense of rotation by inverting the axis of rotation. Given our usual standard of using the Right Hand Rule to describe the polarity of rotations, this makes perfect sense. Reversing the direction of the axis is equivalent to reversing the direction of rotation.

Another interesting take on the above is that rotation quaternions are not unique:

q = -q (Eqn. 16)

Any rotation quaternion can be multiplied by -1 and still result in the same rotation! That's because we reversed both the angle AND the axis of rotation (which then cancel each other). It is conventional therefore to remove the ambiguity by negating a rotation quaternion if its scalar component is negative.

At this point, you are surely wondering why in the world you might, or might not, choose to use quaternions instead of rotation matrices. Here's a brief summary of the pros and cons.

Table 26. Pros & Cons of the Different Orientation Representations

Topic	Quaternion	Rotation Matrix
Storage	Requires **16 bytes** of storage in single precision floating point (4 elements at 4 bytes each)	Requires **36 bytes** of storage (9 elements at 4 bytes each)
Computation (for 2 sequential rotations)	4 elements each requiring 4 multiplies and 3 additions = **28 operations**	9 elements, each requiring 3 multiplies and 2 additions = **45 operations**
Vector rotation	Rotating a vector by pre- and post-multiplication of quaternion requires **52 operations**	Rotating a vector via rotation matrix requires **15 operations** (3 elements each requiring 3 multiplies and 2 additions)
Discontinuities	Generally, we force the scalar part of the quaternion to be positive, which can cause a discontinuity in the rotation axis (it flips).	None
Ease of Understanding	Generally takes a lot of study to understand the details	Easily understood by most engineers

Topic	Quaternion	Rotation Matrix
Conversion	From rotation matrix = $\begin{bmatrix} m11 & m12 & m13 \\ m21 & m22 & m23 \\ m31 & m32 & m33 \end{bmatrix}$ We have q0 = 0.5 sqrt(m11 + m22 + m33 + 1) q1 = (m32 – m23) / (4q0) q2 = (m13 – m31) / (4q0) q3 = (m21 – m12) / (4q0) (Eqn. 17)	RM = $\begin{bmatrix} 2q02 - 1 + 2q12 & 2q1q2 - 2q0q3 & 2q1q3 + 2q0q2 \\ 2q1q2 + 2q0q3 & 2q02 - 1 + 2q22 & 2q2q3 - 2q0q1 \\ 2q1q3 - 2q0q2 & 2q2q3 + 2q0q1 & 2q02 - 1 + 2q32 \end{bmatrix}$ (Eqn. 18)

Equations 17 and 18 are consistent with regards to direction of rotation. If instead of rotating a vector in a fixed frame of reference, you rotate the frame of reference itself, you will need to use the transpose of Eqn. 18 and invert q1, q2 and q3 in Eqn. 17.

Returning to the quaternion rotation operator **W** = q**V**q* , note that **V** needs to be expressed as a quaternion of the form [0, vx, vy, vz], and the multiplications are quaternion multiplies as defined in Eqn. 11. q* is the complex conjugate defined in Eqn. 12.

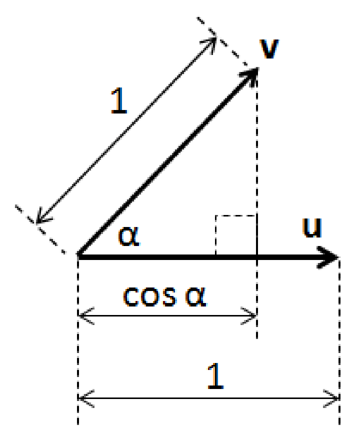
If you do a lot of graphics or sensor fusion work, you will probably find yourself constantly switching between the various representations we've considered. You'll find it useful to remember a couple of identities from your high school geometry course:

10.2.1.1 The Dot Product

u. **v** = | **u** | | **v** | cos α (Eqn. 19)

If both **u** and **v** are unit vectors, then:

u. **v** = cos α (Eqn. 20)



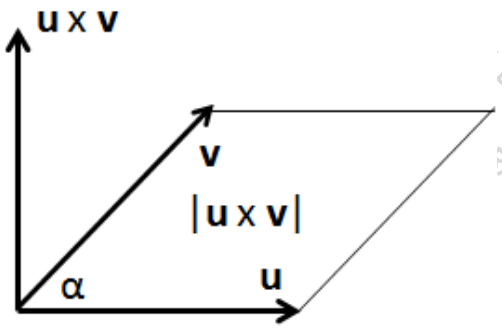
10.2.1.2 The Cross Product

u x **v** = | **u** | | **v** | sin α **n** (Eqn. 21)

where **n** is a unit vector perpendicular to the plane containing **u** and **v** (the polarity of **n** follows the right hand rule).

If both **u** and **v** are unit vectors, then:

n = **u** x **v** / (sin α) (Eqn. 22)



If you've been paying attention, you will see that α is the rotation of u into v about the axis of rotation defined by u x v. See! It's simple! Axis and angle!

10.2.2 References

[1] Quaternions and Rotation Sequences, Jack B. Kuipers, Princeton University Press, 1999

[2] Euler Angles from the Wolfram Demonstrations Project by Frederick W. Strauch

[3] Diversified Redundancy in the Measurement of Euler Angles Using Accelerometers and Magnetometers, Chirag Jagadish and Bor-Chin Chang, Proceedings of the 46th IEEE Conference on Decision and Control, Dec. 2007

[4] "Euler Angles" at Wikipedia

11. Theory: Hard & Soft Iron Magnetic Compensation

This section is a variation on a posting (Hard & Soft Iron Magnetic Compensation Explained) which initially appeared on the Freescale (now NXP) Embedded Beat blog site in 2011.

This section explores issues that you may encounter when using any magnetic sensor in consumer applications. To keep things simple, let us consider the case where you're integrating a magnetic sensor into a smart phone. Nominally, you would like to use your magnetic sensor to implement compass and navigations features. In a pristine environment, free from interference, we could take measurements taken directly from our sensor. The real world is not that simple.