# Attitude with an Attitude

An Informal Intro to Spacecraft Attitude Determination

Brian Leung
University of Michigan, Class of 2022

January 2, 2021

# Contents

# Chapter 1

# Preface

## 1.1 Prerequisites

You're welcome to dive in and learn as you go, but knowing some **Calculus** and **Linear Algebra** will help immensely. Honestly, with how applicable it is, you should just learn linear algebra in general. Classes at the undergraduate level should more than enough. If you're motivated enough, there's also plenty of online classes that you can take. **Physics 1/Dynamics** and **Diff Eq** are also useful, but you can probably get through this text without them.

## 1.2 Message to the Reader

At the time of writing this, I am not a seasoned spacecraft engineer nor a cutting edge researcher. I'm currently just an undergrad in Aerospace Engineering, hoping that my understanding will help you out. I hope you learn something new and feel a little more comfortable in this field.

## 1.3 Acknowledgement

I owe much thanks to CU Boulder's "Spacecraft Dynamics and Control Specialization" on Coursera and Professor Hanspeter Schaub, the

professor of the course. Seriously, it's great. If you've got the time and access, I highly recommend you look through it.

## 1.4 What This Text Offers

This is a quick and dirty intro to the fundamentals of spacecraft attitude description and determination, minimizing lengthy proofs and derivation. This text is essentially CU Boulder's Specialization, Part 1, distilled down to the key takeaways. It offers conceptual questions along the way to test your knowledge. It can act as a springboard into more advanced topics for the aspiring spacecraft engineer.

I want to emphasize that this text is not a university textbook level reference, bursting at the seams with advanced terminology and derivations. This will not delve into dynamic attitude determination or control theory either.

Chapter 2

# What is Attitude Anyways?

## 2.1 Definition

Hint: It's not a state of mind. A fairly straightforward definition for attitude is:

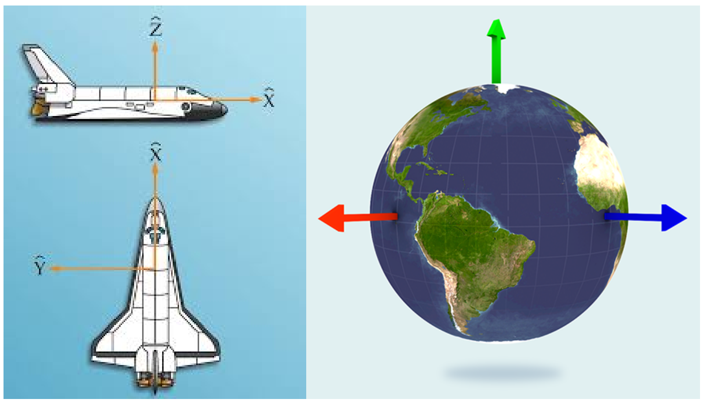
**ATTITUDE: A body's angular orientation with respect to a reference frame.**

In really basic terms, how an object is pointed. A body just refers to something like a plane, spacecraft, or satellite. Angular orientation is pretty self-explanatory. The important part of this definition is the reference frame.

Let me pose this question to you. If you're floating around in an infinite black void, how do you decide which way is up? or down? In reality, the problem is a bit more complex than that, but it's a question you can't answer without the help of a *reference frame.*

## 2.2 Frames?

When I say reference frame, or frames in general, I mean coordinate frames: XYZ, $\hat{i}\hat{j}\hat{k}$, 3 basis vectors, etc. You can pretty much glue a coordinate frame onto anything you want. Attaching frames to things helps ground them in a mathematical context, which helps in more reasons than I can name. For example, it's a lot easier to say things like "I'm about 10 meters south and 1 meter east of you with respect to your NED frame". This is just a rough example, but there are plenty of other good reasons to put frames on bodies.



A Frame on the Space Shuttle (L)
ECEF (Earth Centered, Earth Fixed) Frame (R)

**BODY: Refers to object of interest (e.g. spacecraft, satellite)**
When someone says "Body Frame", that refers to the frame glued to your satellite, space shuttle, UFO, rover. Basically, the thing you're trying to keep track of.

**INERTIAL: Refers to reference (e.g. Earth)**
When someone says inertial frame, they always mean reference. In most situations, the Earth and its variety of frames is what you'll refer to as inertial. However, you can also set the inertial frame to something that's not the Earth, like the ISS.

## 2.3 It's All Relative

I must emphasize that *attitude is all relative.* Without knowing and specifying a reference frame, trying to describe your attitude is basically useless.

## 2.4 Why Worry About Attitude Anyways?

Asking why attitude is important is similar to asking "Why do planes have to be headed in the right direction?". Depending on the project/mission, the importance of making sure the body is pointed in the right direction ranges anywhere from marginal (super basic cubesats) or critical (space telescopes, comms satellites, space shuttle). But 9 times out of 10, attitude is pretty high on the priority list.

The math in this topic also has plenty of appearances outside of the aero sphere, such as in video games, robotics, computer vision, and biomechanics.

## 2.5 4 Basic Truths

1. **You need a minimum of 3 coordinates to describe relative angular displacement in 3D.**

   There's likely some in-depth proof showing why this is the case, but I'm not qualified enough to comment on it. The most popular example of a 3 coordinate set is Euler Angles.

2. **A 3 coordinate set will have at least 1 geometric orientation where the coordinates are singular.**

   When an attitude description becomes **singular**, it's best described as acting ... weird. As in, the math doesn't behave very nicely (divisions by 0, non-uniqueness, etc.). As a real life example, think gimbal lock. The gimbal, when turned a certain way, loses degrees of freedom.

3. **At/near the singularities, the kinematic differential equations are also singular.**

When you get close to a singularity in your attitude description, the kinematic differential equations of your attitude description also turn feral. These won't be too heavily covered in this text so don't worry too much about them.

4. **There exists redundant sets of 4 or more coordinates that are universally valid.**

   While all 3 coordinate descriptions go singular somewhere, there are 4 coordinate descriptions that won't go singular anywhere. Everybody's favorite example of this is the quaternion.

# Chapter 3

# DCMs

It doesn't stand for dichloromethane. And please, don't call it a DCM matrix.

**DIRECTION COSINE MATRIX: A transformation matrix that converts from one reference frame to another.**

We consider the DCM the "mother of all attitude parameterizations", quoting Professor Schaub. Every DCM maps to an attitude description of your choice and every description maps back to a DCM. You could even say it's the center of the great web of descriptions. With DCMs, we directly interact with the basis vectors of the frames themselves. Before we move any further, let's get some *formalisms* out of the way.

## 3.1 Funky Formalisms

When describing the body frame, I'll use the letter $b$. Describing the basis vectors that make up the body frame, I'll say $\hat{b_1}, \hat{b_2}, \hat{b_3}$. Likewise, describing the inertial frame basis vectors uses the letter $n$, like so: $\hat{n_1}, \hat{n_2}, \hat{n_3}$. Constantly writing out $\hat{b_1}, \hat{b_2}, \hat{b_3}$ is gonna get old real fast, so lets save our hands some work and assemble them into something called a *vectrix.*

$$\{\hat{b}\} = \begin{bmatrix} \hat{b}_1 \\ \hat{b}_2 \\ \hat{b}_3 \end{bmatrix} \quad \{\hat{n}\} = \begin{bmatrix} \hat{n}_1 \\ \hat{n}_2 \\ \hat{n}_3 \end{bmatrix}$$

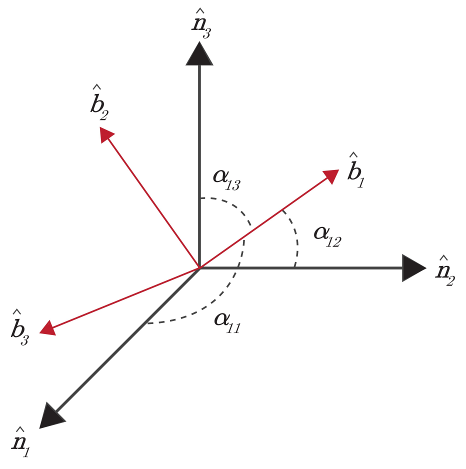
In addition to saving ink, vectrices are also easier to do linear algebra with, since they're (for all intents and purposes) matrices. When the time comes, I'll refer to the components of a matrix, like so:

$$[C] = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix}$$

Pretty standard matrix stuff. Now, we can get into the math.

## 3.2 The Math: A Tale of 2 Frames

Here's a diagram of the 2 frames. $\hat{b}_1$ forms angles $\alpha_{11}, \alpha_{12}, \alpha_{13}$ with the 3 inertial basis vectors $\hat{n}_1, \hat{n}_2, \hat{n}_3$. This applies for the other body vectors as well, but that's just not shown on the diagram.



A good way to relate a body frame vectors to an inertial frame vectors is via the following:

$$\hat{b_1} = \cos(\alpha_{11})\hat{n_1} + \cos(\alpha_{12})\hat{n_2} + \cos(\alpha_{13})\hat{n_3}$$
$$\hat{b_2} = \cos(\alpha_{21})\hat{n_1} + \cos(\alpha_{22})\hat{n_2} + \cos(\alpha_{23})\hat{n_3}$$
$$\hat{b_3} = \cos(\alpha_{31})\hat{n_1} + \cos(\alpha_{32})\hat{n_2} + \cos(\alpha_{33})\hat{n_3}$$

The cosines find the projection of each $\hat{b}$ against each $\hat{n}$. Being masters of linear algebra, we can immediately organize this whole thing into the following, to convert from inertial to body frame:

$$\{\hat{b}\} = \begin{bmatrix} \cos(\alpha_{11}) & \cos(\alpha_{12}) & \cos(\alpha_{13}) \\ \cos(\alpha_{21}) & \cos(\alpha_{22}) & \cos(\alpha_{23}) \\ \cos(\alpha_{31}) & \cos(\alpha_{32}) & \cos(\alpha_{33}) \end{bmatrix} \{\hat{n}\} = [BN]\{\hat{n}\}$$

And from the definition of a cosine, we can also say:

$$\{\hat{b}\} = \begin{bmatrix} b_1 \cdot n_1 & b_1 \cdot n_2 & b_1 \cdot n_3 \\ b_2 \cdot n_1 & b_2 \cdot n_2 & b_2 \cdot n_3 \\ b_3 \cdot n_1 & b_3 \cdot n_2 & b_3 \cdot n_3 \end{bmatrix} \{\hat{n}\} = [BN]\{\hat{n}\}$$

We'll call this matrix [BN]. The BN matrix we have here is the *Direction Cosine Matrix*. It doesn't take a rocket scientiest to see that the "Direction Cosine" part of the name comes from all these cosines.

### 3.2.1 Well, Whats The Point?

Glad you asked. Here's a situation: You are on Earth and your friend is on a spaceship in orbit. You just saw a really cool star and want to radio up to your friend where to look, with respect to their frame. You know the DCM that converts from your frame to theirs. How would you achieve this?

We can easily do this with DCMs. Lets say that v is the vector to the star. We can describe it in both the inertial frame and body frame, like so:

$$v = v_{b1}\hat{b_1} + v_{b2}\hat{b_2} + v_{b3}\hat{b_3} = [v_b]^T\{\hat{b}\} \text{ (1)}$$

$$v = v_{b1}\hat{n_1} + v_{n2}\hat{n_2} + v_{n3}\hat{n_3} = [v_n]^T\{\hat{n}\}\ (2)$$

The $[v_b]$ and $[v_n]$ just indicate the coefficients that you multiply your basis vectors by. That T in the right side of the equation just indicates a transpose. Pretty standard coordinate frame stuff.

Well, we have that {b} vectrix in the equation 1 and we know that $\{\hat{b}\} = [BN]\{\hat{n}\}$. So lets get to substituting.

$$[v_b]^T\{\hat{b}\} = [v_n]^T\{\hat{n}\}$$

$$[v_b]^T[BN]\{\hat{n}\} = [v_n]^T\{\hat{n}\}$$

Doing some funky transposing and linear algebra things, we'll discover that:

$$\boldsymbol{v_b = [BN]v_n}$$

and likewise,

$$\boldsymbol{v_n = [BN]^T v_b}$$

As you can see, DCMs not only *help describe how one frame is rotated relative to another*, but it can also help *translate what one vector looks like to an observer in a different frame.*

## 3.3 Adding and Subtracting DCMs

Despite all we've been doing so far, we are allowed to have more than 2 frames. For example, let's say your ship is floating outside the ISS. You know the DCM that converts the ISS frame to your ship's frame. You also know the DCM that converts some inertial frame to ISS frame. What if you want to know how to convert from inertial straight to your ship's frame?

Lets represent the ship frame = U, the ISS frame = I, and the inertial frame = N. We want to find the DCM [UN], but we only know [UI] and [IN]. The solution is super simple:

$$\boldsymbol{[UN] = [UI][IN]}$$

This is what we call *DCM Addition.* But what about *DCM Subtraction*?

Let's say we already know [UN] and we know [IN]. How would we find the ship frame relative to the ISS frame [UI]? Following our addition we can say:

$$[UI] = [UN][NI]$$

But wait a second, we don't know [NI]? We actually do. Transposing an orthogonal matrix gives the inverse.

$$[IN]^T = [NI]$$

Thus we can say:

$$\boldsymbol{[UI] = [UN][IN]^T}$$

Doing this to [UN] subtracts out the part where I is related to N.

## 3.4 Properties

Well, what makes a DCM a DCM?

1. **DCMs are 3x3.**
2. **The magnitude of each row and column is 1.**

   Ensuring the rows and columns are unit length makes the math less chaotic.
3. **Every DCM element is $\leq |1|$.**

   A consequence of the above. Not exactly hard to see why.
4. **DCMs are orthogonal.**

   Using DCMs with orthogonal vectors is convention. Making sure DCMs are orthogonal is also why we can invert them by transposing.

5. **The determinant of a DCM is +1 (Unit, Orthogonal, Right).**

   This property is essentially a result of the previous properties compounded together. A unit orthogonal matrix has a determinant of $\pm 1$ and the sign of the determinant indicates the handedness of the DCM. A +1 indicates a right handed frame and -1 is a left handed frame. Left handed frames technically aren't wrong, but right handed frames have been convention for forever. This is why right handed frames are referred to as "proper" and left, "improper".

6. **DCMs are non-singular!**

   There's no divisions by zero, no weird functions, and since we're working at the lowest level, not many ambiguities can arise.

### 3.4.1 Wait a Minute...

If DCMs are non-singular, why don't we exclusively use them to describe attitude? While DCMs are lovely and non-singular, they're not always the easiest/intuitive paramaterization to use.

*The sheer amount of elements in the DCM is not very convenient.* There are 9 elements and 6 constraints to keep track of attitude. When we tack on angular velocity $\omega$ and multiple frames, the amount of numbers flying around can quickly get out of hand.

*DCMs aren't super easy to visualize.* If I hand you a list of 9 numbers, would you be able to immediately tell how the body frame is rotated? Probably not.

However, the DCM is super useful for translating between attitude descriptions. We often use the DCM as a common "hub" to translate from description to description.

## 3.5 Test Your Knowledge

1. Identify the illegal DCMs:

$$a. \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} b. \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} c. \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.866 & -0.5 \\ 0 & 0.5 & 0.866 \end{bmatrix} d. \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2. Given DCMs [KB], [KJ], [NJ], how would you find DCM [BN]?

# Chapter 4

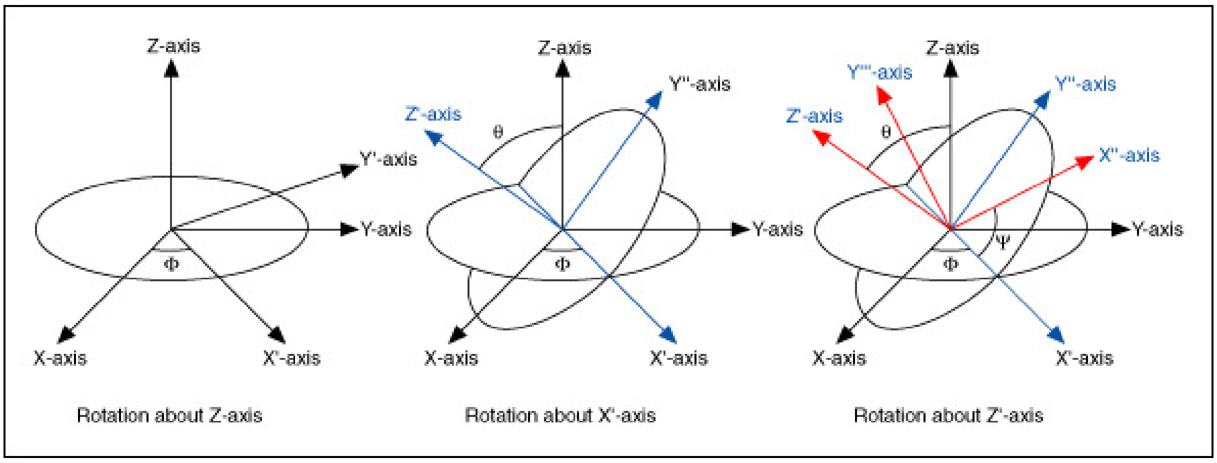
# Euler Angles

If you've heard of yaw pitch or roll, you've most certainly heard of Euler Angles.

**EULER ANGLES: A parameterization that describes attitude in 3 sequential rotations.**

To describe an attitude in Euler angles, you need to specify your type of Euler angle, referred to as a *set*. Then, you specify the degrees of the angles that belong to this set. Each of the angles describe how far you rotate the frame around that specified axis.

Here's a visual example if it ain't too clear:



This is what's described as a 3-1-3 Euler Angle, since the rotations operate on the Z, X, and Z body axes. The rotation sequence does the following:

1. Rotate the frame around the Z-axis by angle $\phi$. This forms a new frame with $X'$ and $Y'$ axes.
2. Rotate the frame around the X'-axis by angle $\theta$. This forms a frame with the $Z'$ and $Y''$ axes.
3. Finally, rotate the frame around the Z' axis by angle $\psi$. This forms a frame with the $X''$ and $Y'''$ axes, and the rotation is finished.

Of course, this is not the only set out there.

## 4.1 Types of Euler Angles

Euler angles fall into two categories.

1. **Symmetric:**

   A Symmetric Euler set implies that the first and last rotation axes are repeated. The best example of this is the 3-1-3 Euler angle shown above.

2. **Asymmetric:**

   On the other hand, Asymmetric Euler sets don't have repeating rotation axes. You're probably most familiar with the Yaw-Pitch-Roll or 3-2-1 set.

In total, there are 12 possible sets.

## 4.2 Euler Angles to DCMs

Euler Angles relate to DCMs pretty seamlessly. We can represent each individual rotation around an axis as a DCM. These individual rotations just move us from intermediate frame to intermediate frame. Once you have all these DCMs together, you can just multiply them together to find the overall DCM.

But how do we represent rotations around axes as DCMs?:

Rotation matrix around axis 1 (X):

$$[R_1(\theta)] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}$$

Rotation matrix around axis 2 (Y):

$$[R_2(\theta)] = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}$$

Rotation matrix around axis 3 (Z):

$$[R_3(\theta)] = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

To get the full DCM of a $\alpha\beta\gamma$ Euler set:

$$[BN(\theta_1, \theta_2, \theta_3)] = [R_\gamma(\theta_3)][R_\beta(\theta_2)][R_\alpha(\theta_1)]$$

(Depending who you work with, $\theta_1, \theta_2, \theta_3$ is the same as $\phi, \theta, \psi$, respectively. It'll depend on who you ask, just make sure you clarify.)

## 4.3 DCMs to Euler Angles

Converting from DCMs to Euler angles is a more involved process, due to the large variety of sets.

## 4.4 Singularities

As you've probably heard, Euler angles are prone to throwing singularities when rotated in certain ways.

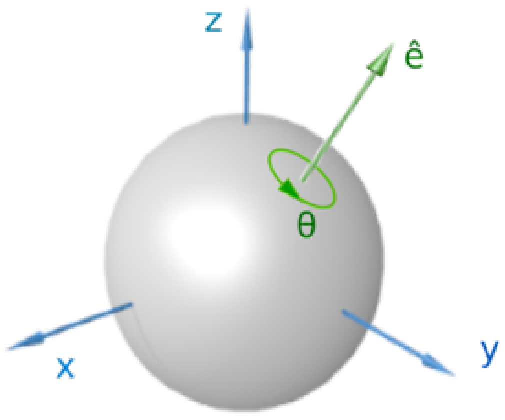
## 4.5 Test Your Knowledge

# Chapter 5

# Principal Rotation Vectors

Euler Angles are great and easy to visualize, but you really don't wanna use them for body frames that can tumble head over heels. Let's try something new, something called a Principal Rotation Vector. First, let's internalize *Euler's Principal Rotation Theorem.* it states:

**THEOREM: A rigid body can be brought from a start to an end orientation by a single rigid rotation ($\phi$ rad/deg) around some principal axis ($\hat{e}$). This principal axis is fixed in both the start and end orientation.**



(I know it says $\theta$, just pretend it's $\phi$)

We can take advantage of this theorem and derive the **Principal Rotation Vector** attitude parameterization, also known as *PRVs* for short.

PRVs consist of 2 parts:

1. **ê, The Principal Rotation Vector**

   As the name may have revealed, you indeed need to define a principal rotation vector to spin your frame around. The components of the principal rotation vector are labeled as $< \mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3} >$

2. $\phi$, **How Far to Rotate**

   You also need to define how far to spin the frame around the PRV.

## 5.1 PRVs to DCM

This section isn't too complex, the formula is just thick.

$$[BN] = \begin{bmatrix} e_1^2\Sigma + c_\phi & e_1e_2\Sigma + e_3s_\phi & e_1e_3\Sigma - e_2s_\phi \\ e_2e_1\Sigma - e_3s_\phi & e_2^2\Sigma + c_\phi & e_2e_3\Sigma + e_1s_\phi \\ e_3e_1\Sigma + e_2s_\phi & e_3e_2\Sigma - e_1s_\phi & e_3^2\Sigma + c_\phi \end{bmatrix}$$

$$\Sigma = 1 - c_\phi$$

Note that the $c_\phi, s_\phi$ are just shorthand for $\cos\phi, \sin\phi$

## 5.2 DCM to PRVs

Thankfully, this process is a little slimmer.

$$cos(\phi) = \frac{1}{2}(C_{11} + C_{22} + C_{33} - 1)$$

$$\hat{e} = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \frac{1}{2\sin\phi}\begin{bmatrix} C_{23} - C_{32} \\ C_{31} - C_{13} \\ C_{12} - C_{21} \end{bmatrix}$$

Chapter 6

# Quaternions/EPs

Quaternions are usually the first thing that comes to mind when someone says "non-singular attitude description". If you've googled "quaternion math", you might've been overwhelmed by the mathematical mumbo jumbo, so let's leave all that stuff for the math majors. Thankfully, you don't have to be a master of quaternion math to be able to use quaternions.

### 6.0.1 A Bit of Definition

Just to get this out of the way, you'll probably hear the term "**Euler Parameters**" being used (EPs for short) and you might wonder what the difference is. Quaternions are the mathematical construct and Euler Parameters are the *coefficients* of the quaternion.

## 6.1 Well, What is a Quaternion?

Quaternions form a number system that extends the complex numbers. They're commonly represented with 4 terms:

$$a + bi + cj + dk$$

Where $a, b, c, d$ are real coefficients and $i, j, k$ are the hypercomplex terms. $a$ is referred to as the scalar/real term. I'm certainly not qualified to say more so I'll leave it at that.

## 6.2 Defining EPs from PRVs

As you probably know, quaternions involve 4 terms. We'll refer to them as:

$$< \beta_0, \beta_1, \beta_2, \beta_3 >$$

Where $\beta_0$ is the real term. Please be aware that the nomenclature will change according to who you ask.

Referring back to PRVs, we can define the EPs as:

$$\beta_0 = cos(\phi/2)$$
$$\beta_1 = e_1 sin(\phi/2)$$
$$\beta_2 = e_2 sin(\phi/2)$$
$$\beta_3 = e_3 sin(\phi/2)$$

### 6.2.1 But... Why?

You might be thinking: "This seems completely useless! You're just making a PRV way more complex!". Thinking back to PRVs, you'll remember that the $\hat{e}$ vector went ambiguous when at a $\phi$ of 0. With quaternions, you'll never have to deal with the $\hat{e}$ vector at 0, since the sines make it disappear. You're now free to point in whatever direction you choose, without fearing a singularity.

## 6.3 Constraints

EPs have only 1 constraint, which is the following:

$$\beta_0^2 + \beta_1^2 + \beta_2^2 + \beta_3^2 = 1$$

In math words, this tells us that EPs live on a *unit hypersphere.* Quaternions aren't actually limited to being unit length, but that's a whole nother can of worms that most engineers don't bother opening.

## 6.4 EPs to DCM

Converting an EP set to a DCM involves the following:

$$[C] = \begin{bmatrix} \beta_0^2 + \beta_1^2 - \beta_2^2 - \beta_3^2 & 2(\beta_1\beta_2 + \beta_0\beta_3) & 2(\beta_1\beta_3 - \beta_0\beta_2 \\ 2(\beta_1\beta_2 - \beta_0\beta_3) & \beta_0^2 - \beta_1^2 + \beta_2^2 - \beta_3^2 & 2(\beta_2\beta_3 + \beta_0\beta_1) \\ 2(\beta_1\beta_3 + \beta_0\beta_2) & 2(\beta_2\beta_3 - \beta_0\beta_1) & \beta_0^2 - \beta_1^2 - \beta_2^2 + \beta_3^2 \end{bmatrix}$$

If you want a cleaner looking formula,

$$[BN] = (\beta_0^2 - \epsilon^T\epsilon)[I_{3x3}] + 2\epsilon\epsilon^T - 2\beta_0[\tilde{\epsilon}]$$

$$\epsilon = (\beta_1, \beta_2, \beta_3)$$

(The tilde indicates the skew symmetric operator.)

## 6.5 DCMS to EP

Moving from DCMs to EPs is a little more complex. In order to avoid bumping into a division by zero, people use *Sheppard's Method.* Here's how it goes:

1. Find the largest value out of the following:

$$\beta_0^2 = \frac{1}{4}(1 + trace([C]))$$

$$\beta_1^2 = \frac{1}{4}(1 + 2C_{11} - trace([C]))$$

$$\beta_2^2 = \frac{1}{4}(1 + 2C_{22} - trace([C]))$$

$$\beta_3^2 = \frac{1}{4}(1 + 2C_{33} - trace([C]))$$

2. Find the **positive square root of this largest value**, and find the rest of the values using the following set of equations:

$$\begin{aligned}
\beta_0\beta_1 &= (C_{23} - C_{32})/4, \ \beta_1\beta_2 = (C_{12} + C_{21})/4 \\
\beta_0\beta_2 &= (C_{31} - C_{13})/4, \ \beta_3\beta_1 = (C_{31} + C_{13})/4 \\
\beta_0\beta_3 &= (C_{12} - C_{21})/4, \ \beta_2\beta_3 = (C_{23} + C_{32})/4
\end{aligned}$$

### 6.5.1 Elaborate?

This might appear vague, so lets break it down a bit in an example.

Let's say that the largest value of the bunch is $\beta_2^2$. The positive square root of this value is just: $\beta_2$. Now that we have this value, we can find the rest of the $\beta$s by dividing all the step 2 products that include $\beta_2$. For example, to find $\beta_0$:

$$\beta_0 = \frac{\beta_0\beta_2}{\beta_2}$$

And so on so forth.

## 6.6 EP Addition

While you can add the DCMs derived from an EP set, you could also do it a more direct way. Rotating a quaternion $A =< A_0, A_1, A_2, A_3 >$ by $B =< B_0, B_1, B_2, B_3 >$ into $C$:

$$[C] = \begin{bmatrix} B_0 & -B_1 & -B_2 & B_3 \\ B_1 & B_0 & B_3 & -B_2 \\ B_2 & -B_3 & B_0 & B_1 \\ B_3 & B_2 & -B_1 & B_0 \end{bmatrix} [A]^T$$

Chapter 7

# Classical Rodriquez Parameters

Classical Rodriguez Parameters, or CRPs for short, are a 3 coordinate parameterization. These will come in handy when we discuss the QUEST method. In the context of the other parameterizations, they're honestly not all that complex. Since this is a 3 coordinate description, there is no constraint.

## 7.1 EPs to CRPs

In the context of EPs, CRPs are described as:

$$q_i = \frac{\beta_i}{\beta_0}$$

for $i = 1, 2, 3$

### 7.1.1 A Singularity?

Your keen eye might have noticed a zero appearing if your angle hits 180. And it's true, **CRPs hit a singularity when the rotation**

**around the axis is at 180 degrees**, like so:

$$q_i = \frac{\beta_i}{\cos(180^\circ/2)} = \frac{\beta_i}{0}$$

## 7.2 CRPs to EPs

$$\beta_0 = \frac{1}{\sqrt{1 + q^T q}}$$
$$\beta_i = \frac{q_i}{\sqrt{1 + q^T q}}$$

for $i = 1, 2, 3$

## 7.3 PRVs to CRPS

The relation between CRPs and PRVs is very simple:

$$\bar{q} = \tan{(\frac{\phi}{2})}\hat{e}$$

For small angles, we can even approximate it to where it's even simpler:

$$\bar{q} \approx \frac{\phi}{2}\hat{e}$$

# Chapter 8

# Attitude Determination?

Now that you got a good chunk of the preliminaries out of the way, let's get into how it's applied. The whole topic of *Attitude Determination* is broken up into 2 types of problems:

1. **Static Attitude Determination:** You take all of your measurements at the same time, and try to solve for the optimal geometry of the measurements, figuring out your attitude at that instant of time. We'll be focusing on this one.

2. **Dynamic Attitude Determination:** You take your measurements over time and try to figure out how your body is spinning around. This is a far harder problem and one that we won't cover.

### 8.0.1 Definition

Assume that we have $n > 1$ measurement vectors in body frame $({}^B\hat{v}_k)$ and their corresponding measurement vectors in inertial frame $({}^N\hat{v}_k)$. These measurement vectors include information such as the direction to the sun, the direction of the magnetic field, direction to the center of the earth, etc. Attitude determination is figuring out how our body frame was spun around (relative to inertial) such that the inertial measurement vectors line up with the body measurements. We can codify this rotation into the DCM [BN], formalizing the problem we'll tackle as:

$$^{B}\hat{v}_k = [BN]\,^{N}\hat{v}_k$$

$$\text{for k = 1,2, ... n}$$

## 8.1 How Many Vectors?

The section prior raises the question, "What's the least amount of measurement vectors we need to figure out our attitude?".

### 8.1.1 A 2D Example

### 8.1.2 Now, in 3D

### 8.1.3 Mathematical Formalism

Attitude is a 3 DoF problem. Most measurement vectors have 2 DoF. Since you need to know 2 measurement vectors, you've constrained 4 DoF. Therefore, attitude is always an overconstrained problem. This is just a mathematical fact and, for our purposes, there aren't any major implications from this.

## 8.2 Common Sensors

Attitude sensors fall into 2 categories. *Absolute attitude sensors* observe phenomena (like fields or celestial objects) outside the craft. *Relative attitude sensors* detect changes in the attitude state of a craft, unlike absolute sensors. Relative sensors accumulate error over time and often need to be double checked against absolute sensors.

### 8.2.1 Sun Sensor

Sun sensors detect the sun vector relative to your craft. They usually consist of an array of photodiodes or phototransistors. By doing some voltage detection, you can calculate where the sun is. They do have a limited FOV and can be interfered by erroneous reflections off of other spacecraft. Also, they obviously don't work when something is eclipsing the sun. Due to the relatively low cost of silicon and its simplicity, sun sensors are a popular choice for cost-conscious missions.

### 8.2.2 Magnetometer

Magnetometers detect the magnetic field vector of Earth/whatever body you happen to be orbiting. There are a variety of ways that magnetometers are implemented (more than I can confidently say). They don't have a limited FOV, and work on the dark side of Earth, but are not the most precise sensors on your craft. The Earth's magnetic field changes over time, necessitating regular updates, and EMI can slightly vary the readings on the magnetometer.

### 8.2.3 Star Tracker

Star trackers are sensors you'd probably use when you've got a bigger budget, like for large scale probe missions or the SR-71[1]. A star tracker looks at a small section of the celestial sphere and references an extensive database of all the stars in the sky. In most cases, a star tracker can actually give you your full attitude description! Of course, they're not perfect. They're still restricted by FOV and can be interfered with by the sun, shiny space debris, random celestial phenomena etc.

### 8.2.4 Inertial Measurement Unit (IMU)

Inertial measurement units typically consist of a gyroscope (a sensor that detects changes in angular rates) and an accelerometer (a sen-

[1]Yes, the SR-71 used a star tracker.

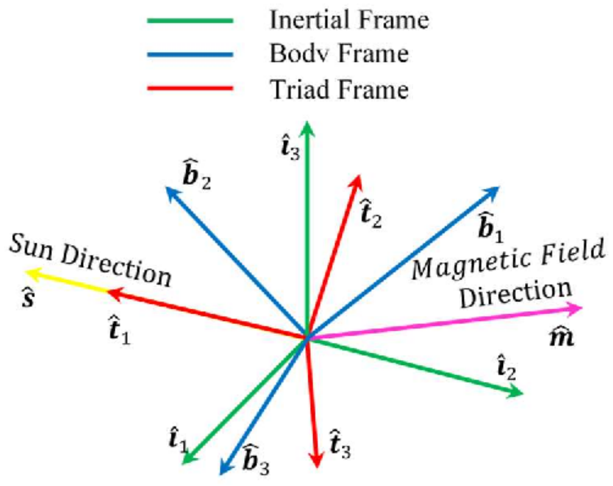
sor that detects acceleration). You could purchase one or the other but they usually come in a single, ready-to-go package. This is an example of a relative attitude sensor, which is subject to accumulated error. The rate at which error accumulates is highly dependent on the quality/price of the sensor.

# Chapter 9

# TRIAD Method

Published in 1964, TRIAD (or specifically, TRIAD-1) is one of the oldest algorithms out there. Despite its age, it is still a viable method for simpler satellites. There exists other variations on TRIAD, but I won't be covering those.

## 9.1 How it Works



We want a DCM that gets us from inertial frame to a body frame, but jumping straight from inertial to body seems pretty hard. Let's use an intermediate frame, more commonly referred to as *triad frame*, to hop from

Inertial $\rightarrow$ Triad $\rightarrow$ Body

### 9.1.1 How Do We Assemble the Triad Frame?

You'll need to assemble the triad frame using the measurement vectors in both the body and inertial context. Assuming we have a sun sensor that gives us a measurement vector $\hat{s}$ and a magnetometer that gives us a vector $\hat{m}$:

For body:

$$^B\hat{t}_1 = {}^B\hat{s},\ \ ^B\hat{t}_2 = \frac{^B\hat{s} \times {}^B\hat{m}}{|^B\hat{s} \times {}^B\hat{m}|},\ \ ^B\hat{t}_3 = {}^B\hat{t}_1 \times {}^B\hat{t}_2$$

Likewise, for inertial:

$$^N\hat{t}_1 = {}^N\hat{s},\ \ ^N\hat{t}_2 = \frac{^N\hat{s} \times {}^N\hat{m}}{|^N\hat{s} \times {}^N\hat{m}|},\ \ ^N\hat{t}_3 = {}^N\hat{t}_1 \times {}^N\hat{t}_2$$

We can now say:

$$[BT] = [^B\hat{t}_1\ {}^B\hat{t}_2\ {}^B\hat{t}_3]$$

$$[NT] = [^N\hat{t}_1\ {}^N\hat{t}_2\ {}^N\hat{t}_3]$$

Recalling DCM addition and subtraction, we can now easily conclude:

$$[BN] = [BT][NT]^T$$

Neat and simple.

## 9.2 Downsides

TRIAD does have its rough edges.

1. **Only 2 measurement vectors can be used.**
   TRIAD-1 only uses 2 measurement vectors. This is problematic for many reasons. If even one sensor becomes inoperable, TRIAD doesn't work. Also, you won't be able to utilize more than 2 types of sensors on your craft.

2. **When measurement vectors align, the triad frame falls apart.**
   This is just a side-effect of the math. The cross product of 2 coincident vectors is just 0, thus you won't be able to form the triad frame.

# Chapter 10

# What is Wahba's Problem?

## 10.1 Definitions and Things

TRIAD is great and all, but what if we want to make use of all the sensors we have on our satellite? After all, your sponsor would probably be pissed if they paid for a GPS receiver and 4 star trackers but you're only using 1 star tracker and a sun sensor.

*Wahba's Problem* was a question posed by the venerable Grace Wahba back in 1965. It goes a little like this:

### 10.1.1 Defining Wahba's Problem

Assume that we have $n > 1$ measurement vectors in body frame (${}^B\hat{v_k}$) and their corresponding measurement vectors in inertial frame (${}^N\hat{v_k}$). Attitude determination is basically a question of finding a DCM $[BN]$ that converts the inertial measurements to the body measurements like so:

$${}^B\hat{v_k} = [\tilde{B}N]\,{}^N\hat{v_k}$$

for k = 1,2, ... n

Nothing we don't know. However, the real world is mathematically cruel. You will never find a $[\tilde{B}N]$ matrix that perfectly maps from inertial to body, hence the little tilde on the B. This is usually due to imperfect sensors giving imperfect measurement vectors.

Well, what do we do now? If we can't acquire a perfect [BN], then we should at least try finding a [BN] that maps our inertial measurements as closely as possible to our body measurements.

So Wahba proposed, "Hey, why don't we try to minimize my handy dandy loss function to get the best possible [BN] matrix?"[1]:

$$\mathbf{J([\tilde{B}N]) = \frac{1}{2}\sum_{k=1}^{n} w_k|\,{}^{\mathit{B}}\hat{v}_k - [\tilde{B}N]\,{}^{\mathit{N}}\hat{v}_k|^2}$$

Let's break this down.

- $J([\tilde{B}N])$: This describes the overall "wrongness" of the [BN] matrix we're looking at. Smaller J = Better [BN].
- $w_k$: This is a *weighting value* assigned to that kth measurement vector. You assign this to each sensor yourself. Higher weights are assigned to more reliable sensors. For example, you would assign a higher weight to a sun sensor or star tracker vs a magnetometer. Note that only relative sizes of the weight values matter, the actual values themselves don't really matter.
- $\hat{v}_k^B - [\tilde{B}N]\hat{v}_k^N$: This describes the error between the kth body measurement and the corresponding kth inertial measurement, mapped to body.

When you boil it down, it's basically a fancier least squares fit. Finding a solution involves minimizing J as much as possible.

### 10.1.2 Well, How Do We Solve It?

Actually, there's no one definitive way to solve this. In addition, clever mathematicians and engineers are still coming up with solutions.

[1] I'm pretty sure she didn't actually say this, just roll with it.

# Chapter 11

# Davenport's Q

The first solution to Wahba's problem we're gonna look at is called Davenport's Q. Paul B. Davenport came up with this quaternion based method in 1968. It is an older method, but it is a precursor to other methods out there, like QUEST.

Beware, this chapter will get a bit math heavy. If you don't feel like looking at the derivation, just check out the "In a Nutshell" section and cherry pick the formulas.

### 11.0.1 Wahba's Problem in Another Light

Instead of outright trying to minimize J, we can twist Wahba's problem into another form, in which we can minimize a value called the "gain". Let's see how it's done.

This is the normal definition of Wahba's problem.

$$J([\tilde{B}N]) = \frac{1}{2}\sum_{k=1}^{n} w_k|\,{}^B\hat{v}_k - [\tilde{B}N]\,{}^N\hat{v}_k|^2$$

That squared part in the right hand side is the same as taking a dot product, so let's just replace it:

$$J([\tilde{B}N]) = \frac{1}{2}\sum_{k=1}^{n} w_k({}^B\hat{v}_k - [\tilde{B}N]\,{}^N\hat{v}_k)^T({}^B\hat{v}_k - [\tilde{B}N]\,{}^N\hat{v}_k)$$

Evaluating this dot product mish mash gives us:

$$J([\tilde{B}N]) = \frac{1}{2}\sum_{k=1}^{n} w_k({}^B\hat{v}_k{}^T\,{}^B\hat{v}_k + {}^N\hat{v}_k{}^T\,{}^N\hat{v}_k - 2\,{}^B\hat{v}_k{}^T[\tilde{B}N]\,{}^N\hat{v}_k)$$

Continuing to evaluate and simplify gives us:

$$J([\tilde{B}N]) = \sum_{k=1}^{n} w_k(1 - {}^B\hat{v}_k{}^T[\tilde{B}N]\,{}^N\hat{v}_k)$$

Don't worry, this is just Wahba's cost function, just in a different form. We still want to minimize J so how do we go about this? We'll have to maximize

$${}^B\hat{v}_k{}^T[\tilde{B}N]\,{}^N\hat{v}_k$$

Let's define this as the **"gain"**:

$$\mathbf{g} = \sum_{\mathbf{k=1}}^{\mathbf{n}} \mathbf{w_k}({}^B\hat{\mathbf{v}}_\mathbf{k}{}^\mathbf{T}[\tilde{\mathbf{B}}\mathbf{N}]\,{}^N\hat{\mathbf{v}}_\mathbf{k})$$

*In order to minimize J, we have to maximize the gain, g.*

## 11.1 Davenport's Method

Davenport developed the following way to get a gain function in the context of quaternions. Bear with me, this is quite the batch of equations.

$$[B] = \sum_{k=1}^{N} w_k\,{}^B\hat{v}_k\,{}^N\hat{v}_k{}^T$$

$$[Z] = [(B_{23} - B_{32}), (B_{31} - B_{13}), (B_{12} - B_{21})]^T$$

$$\sigma = trace([B])$$

$$[S] = [B] + [B]^T$$

$$[K] = \begin{bmatrix} \sigma & Z^T \\ Z & S - \sigma[I_{3x3}] \end{bmatrix}$$

Finally,

$$g(\tilde{\beta}) = \bar{\beta}^T [K] \bar{\beta}$$

Where

$$\bar{\beta} =< \beta_0, \beta_1, \beta_2, \beta_3 >$$

### 11.1.1 Maximizing the Gain?

We are still trying to maximize gain, so we could simply go "Let's just boost $\bar{\beta}$ to infinity". While a good first thought, this won't work since we have to honor the *constraint* that EPs must be unit length. This has turned into a constrained optimization problem. A good way to solve constrained optimization problems is via Lagrange Multipliers, as you may (or may not) remember from Calc 3. Let's represent our Lagrangified gain formula as $g'$.

$$g'(\bar{\beta}) = \underbrace{\bar{\beta}^T [K] \bar{\beta}}_{g} - \underbrace{\lambda(\bar{\beta}^T \bar{\beta} - 1)}_{constraint}$$

We'll need to differentiate $g'$ in order to find the extremum:

$$\frac{d(g'(\bar{\beta}))}{d\bar{\beta}} = 2[K]\bar{\beta} - 2\lambda\bar{\beta} = 0$$

As you can clearly see, what comes out is an eigenvector problem:

$$[K]\bar{\beta} = \lambda\bar{\beta}$$

Well, what does this mean for us? The gain formula just turns into:

$$g(\bar{\beta}) = \bar{\beta}^T [K] \bar{\beta} = \bar{\beta}^T \lambda \bar{\beta} = \lambda$$

Therefore, the optimal EP set is the eigenvector associated with the largest eigenvalue of K.

## 11.2 In a Nutshell

To summarize, Davenport's Q boils down to the following:

1. Compute the [K] matrix.
2. Find the eigenvalues and eigenvectors of said [K] matrix.
3. Choose the largest eigenvalue & corresponding eigenvector.
4. This eigenvector is your set of EPs.

# Chapter 12

# QUEST

Davenport's Q is lovely, but solving that eigenvalue problem is computationally expensive. This makes it fairly unpopular for realtime applications. In order to gain a bit of speed, how about we solve that eigenvalue problem a different way? Although a little less straightforward, the extra speed offered by the QUEST method (which stands for QUaternion ESTimation) is much appreciated. Like the Davenport's Q chapter, quite math heavy.

## 12.1 A Nice Approximation

Recall from the previous chapter the loss ($J$) and gain ($g$) functions:

$$J([\tilde{B}N]) = \sum_{k=1}^{n} w_k(1 - {}^B\hat{v}_k{}^T[\tilde{B}N]\,{}^N\hat{v}_k)$$

$$g = \sum_{k=1}^{n} w_k({}^B\hat{v}_k{}^T[\tilde{B}N]\,{}^N\hat{v}_k)$$

Also recall that the gain function eventually comes out to be the K matrix's largest eigenvalue, let's call this $\lambda_{opt}$:

$$g(\bar{\beta}) = \lambda_{opt}$$

This must mean that, to find an optimal J,

$$J = \sum_{k=1}^{n} w_k - g = \sum_{k=1}^{n} w_k - \lambda_{opt}$$

Let's isolate the optimal eigenvalue:

$$\lambda_{opt} = \sum_{k=1}^{n} w_k - J$$

Solving the problem like this seems like a whole bunch of work that we don't want to do, so let's just assume J is small. Like really, really small. So small, in fact, that we can treat it like 0. Thus, the equation turns into:

$$\boldsymbol{\lambda_{opt} \approx \sum_{k=1}^{n} w_k}$$

We now have a *really* good guess at the real optimal eigenvalue. (Well, unless your sensors are throwing laughably bad measurements that is)

## 12.2 Refining the Guess

We got a great guess, but it's still just a guess. In order to continue, we need to refine it and numerically find the actual maximum eigenvalue. We know that eigenvalues must satisfy the characteristic equation:

$$f(s) = det([K] - s[I_{3x3}]) = 0$$

If this seems unfamiliar, just think back on how you found the eigenvalues for a 2 x 2 matrix. For example:

$$det(\begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}) = 0$$

Which gives a characteristic equation of

$$\lambda^2 + 3\lambda + 2 = 0$$

Since we're trying to find the root of the characteristic equation, we can use *Newton-Raphson Iteration.* Newton-Raphson is just a numerical method to find the roots of polynomials. Don't let the "numerical" scare you, it's actually pretty simple. We'll use the $f(s)$ (the characteristic equation) we defined above and its derivative in the Newton-Raphson:

$$\lambda_0 = \sum_{k=1}^{n} w_k$$

$$\lambda_1 = \lambda_0 - \frac{f(\lambda_0)}{f'(\lambda_0)}$$

Repeating this process a couple of times and checking whether $\lambda$ fulfills the characteristic equation,

$$\lambda_{max} = \lambda_i = \lambda_{i-1} - \frac{f(\lambda_{i-1})}{f'(\lambda_{i-1})}$$

Repeating this moves you closer and closer to the actual maximum eigenvalue. We're allowed to do this since our $\lambda_{opt}$ guess already places us close to the actual $\lambda_{max}$.

## 12.3 Searching for $\bar{q}$

QUEST is actually a bit of a misnomer, since we'll actually be first receiving the result in CRPs. A quick refresher on CRPs:

$$q = \hat{e}\tan\frac{\phi}{2} = \frac{1}{\beta_0}\begin{bmatrix}\beta_1\\ \beta_2\\ \beta_3\end{bmatrix}$$

And let's describe a convenient notational link between the EPs and CRPs, like so:

$$\frac{\bar{\beta}}{\beta_0} = \begin{bmatrix}1\\ \bar{q}\end{bmatrix}$$

First, our original eigenvector problem looks like:

$$[K]\bar{\beta} = \lambda_{opt}\bar{\beta}$$

Let's get this whole formula into CRPs and expand the K matrix:

$$[K]\frac{\bar{\beta}}{\beta_0} = \lambda_{opt}\frac{\bar{\beta}}{\beta_0}$$

$$\begin{bmatrix} \sigma & Z^T \\ Z & S - \sigma[I_{3x3}] \end{bmatrix} \begin{bmatrix} 1 \\ \bar{q} \end{bmatrix} = \lambda_{opt} \begin{bmatrix} 1 \\ \bar{q} \end{bmatrix}$$

If we do the whole matrix multiplication and just trim out the unnecessary top row, we'll get:

$$[Z] + ([S] - \sigma[I_{3x3}])\bar{q} = \lambda_{opt}\bar{q}$$

Superb, we're getting close now. We can now isolate $\bar{q}$, thus:

$$\bar{\mathbf{q}} = ((\lambda_{\mathbf{opt}} + \sigma)[\mathbf{I_{3x3}}] - [\mathbf{S}])^{-\mathbf{1}}[\mathbf{Z}]$$

Well, you came for EPs after all, so you'll need to do the following to convert back to EPs.

$$\bar{\beta} = \frac{1}{\sqrt{1 + \bar{q}^T\bar{q}}} \begin{bmatrix} 1 \\ \bar{q} \end{bmatrix}$$

## 12.4 Aren't You Forgetting Something?

If alarm bells are going off in your head since CRPs go singular at 180 degrees, then you are absolutely right. The way this is resolved is by considering an *alternate body frame*, determining your attitude from that, and adjusting afterwards to your true body frame. Having these 2 body frames will ensure that both will not bump into that singularity at the same time.

This technique is valid since defining a body frame is completely arbitrary. After all, what's stopping you from defining your $b_1$ vector towards say, the left wing of your spaceship?[1]

[1]The answer is nothing, if you're actually wondering.

## 12.5 In a Nutshell

1. Estimate $\lambda_{opt}$ by summing the weight values.
2. Refine your $\lambda_{opt}$ guess by doing Newton-Raphson with the characteristic equation.
3. Acquire your optimal attitude in CRPs. Convert to quaternions.
4. If need arise, handle the singularity by calculating from an alternate body frame.

# Chapter 13

# Answer Key

## 13.1 Chapter 1 DCM

1. a & b
2. $[KB]^T[KJ][NJ]^T$