Attitude with an Attitude
An Informal Intro to Spacecraft Attitude Determination

Brian Leung
University of Michigan, Class of 2022

November 21, 2020

Contents

Chapter 1

Preface

1.1 Prerequisites

You're welcome to dive in and learn as you go, but knowing some **Physics 1/Dynamics**, **Calculus 1-3, Diff Eq** 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 dedicated enough, there's also plenty of online classes that you can take.

1.2 Message to the Reader

At the time of writing this, I am not a seasoned spacecraft engineer nor a cutting edge researcher (well, fingers crossed). For now, I'm just a green 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

I want to emphasize that this text is <u>not</u> 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.

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

---

[1]TLDR: attitude determination speedrun any % glhf

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.**

A body just refers to something like a plane, spacecraft, or satellite. Angular orientation? 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? Of course, 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.*

2.1.1 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 your heart desires. Attaching frames to things helps ground them in a mathematical context, and that 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.
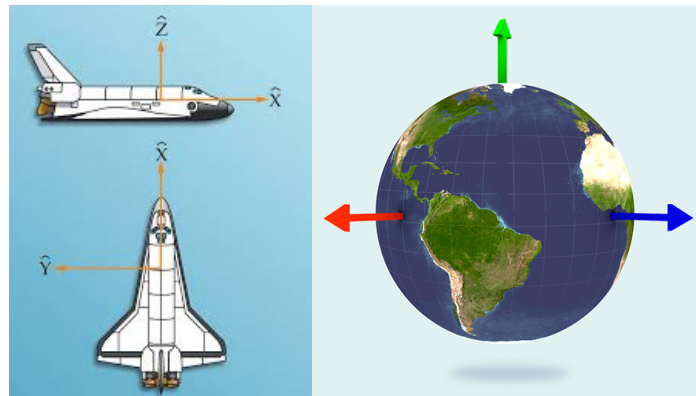
A Frame on the Space Shuttle (L)
ECEF (Earth Centered, Earth Fixed) Frame (R)

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 basic problems, 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.

2.1.2   It's All Relative

I must emphasize that *attitude is all relative*. Without knowing and specifying a reference frame, the description of your attitude is basically useless.

2.1.3   Why Worry About Attitude Anyways?

Trying to brush off attitude determination as unimportant 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 your object is pointed in a desired 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. This topic also has plenty of applications outside of the aero sphere, such as in video games, robotics, and biomechanics.

2.2   4 Basic Truths

1. **You need a minimum of 3 coordinates to describe relative angular displacement in 3D.** There's likely some indepth proof showing why this is the case, but I'm not qualified enough to comment on it. The best example of a 3 coordinate set is Euler Angles, covered in Ch 4.

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.**

Chapter 3

DCM

3.1 Definitions and Things

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. When I mean *attitude parameterization*, I mean descriptions of your attitude, such as quaternions, Euler angles, etc. Every DCM maps to a parameterization of your choice and every parameterization maps back to a DCM. You could say it's the center of the great web of parameterizations. 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.

3.1.1 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 alge-
bra 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.
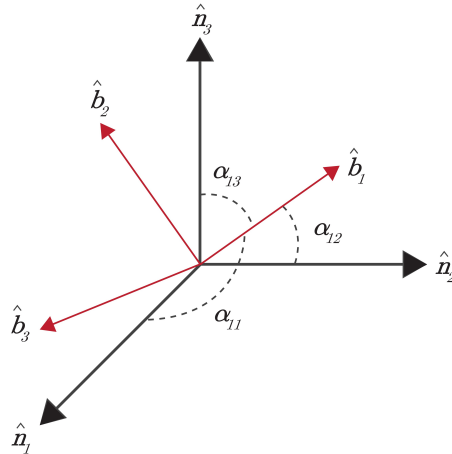
A good way to relate a body frame vectors to an inertial frame vectors
is via the following:

$$\hat{b}_1 = \cos(\alpha_{11})\hat{n}_1 + \cos(\alpha_{12})\hat{n}_2 + \cos(\alpha_{13})\hat{n}_3$$
$$\hat{b}_2 = \cos(\alpha_{21})\hat{n}_1 + \cos(\alpha_{22})\hat{n}_2 + \cos(\alpha_{23})\hat{n}_3$$
$$\hat{b}_3 = \cos(\alpha_{31})\hat{n}_1 + \cos(\alpha_{32})\hat{n}_2 + \cos(\alpha_{33})\hat{n}_3$$

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. DCMs are basically glorified rotation matrices.

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?

If it wasn't obvious, it's via that DCM you have. Lets say that v is the vector to the star. We can describe it in both the inertial frame and body frame, like so:

$$v = v_{b1}\hat{b}_1 + v_{b2}\hat{b}_2 + v_{b3}\hat{b}_3 = [v_b]^T\{\hat{b}\} \quad (1)$$

$$v = v_{b1}\hat{n}_1 + v_{n2}\hat{n}_2 + v_{n3}\hat{n}_3 = [v_n]^T\{\hat{n}\} \quad (2)$$

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. All in all, real 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 UFO is floating outside the ISS. You know the DCM that converts the ISS frame to your UFO 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 UFO frame?

Lets represent the UFO frame as U, the ISS frame as I, and the inertial frame to 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 UFO 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:

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

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. Let's say that some malevolent DCM doesn't maintain unit length. When you multiply some vector through this DCM, the vector that comes out on the other side isn't the same length as the original. And now, not only do you have to keep track of a ton of numbers, you have to scale that vector's components up or down. Since we want to make our lives as easy as possible, this ain't the move.

3. **Every DCM element is $\leq |1|$**

   A consequence of the above. Not exactly hard to see why.

11

4. **DCMs are orthogonal**

   Using vetrices with orthogonal vectors is convention because they're mathematically friendly. The fact that a DCM is orthogonal is the reason we can use the transpose to transform backwards from body to inertial. The inverse of an orthogonal matrix is just its transpose after all.

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

   This property is essentially a result of the previous properties piled together. A unit orthogonal matrix has a determinant of ± 1. Why ± 1 you ask? Well 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. A left handed frame technicallyyyyy isn't wrong, but right handed frames are convention. That's 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. 9 elements to keep track of attitude? With properties that we constantly have to enforce? Tacking on angular velocity ω 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 parameterizations.

3.5 Test Your Knowledge

1. Identify the illegal DCMs:

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

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

Chapter 4

Euler Angles

4.1   Definitions and Things

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.

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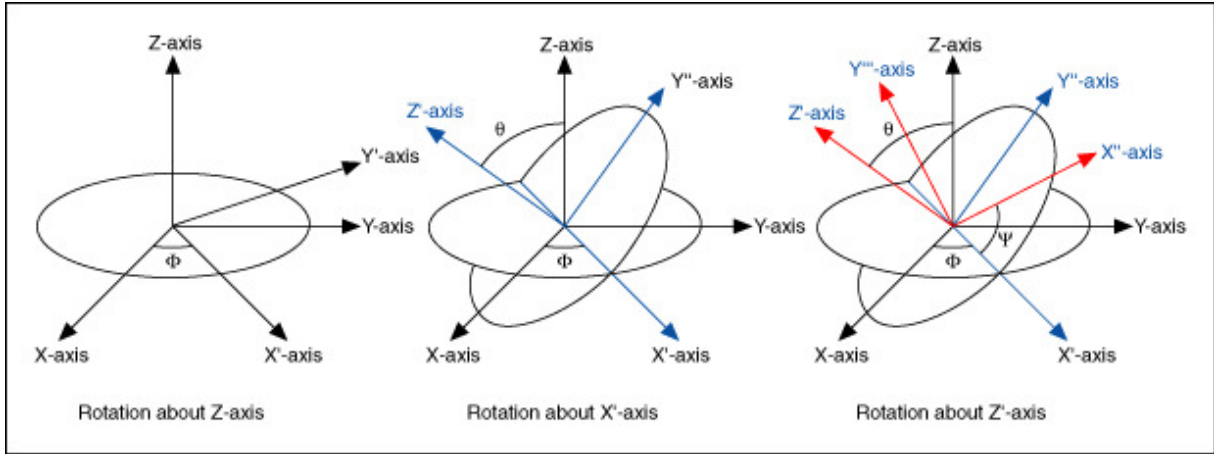
This is what's described as a 3-1-3 Euler Angle, since the rotations operate on the Z, X, and Z body axes. The rotation sequence does the following:

1. Rotate the frame around the Z-axis by angle ϕ. This forms a new frame with X' and Y' axes.

2. Rotate the frame around the X'-axis by angle θ. This forms a frame with the Z' and Y'' axes.

3. Finally, rotate the frame around the Z' axis by angle ψ. 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.

4.2   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.

4.3 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 ϕ, θ, ψ, respectively. It'll vary with who you ask, just make sure you clarify.)

4.4　DCMs to Euler Angles

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

4.5　Singularities

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

4.6　Test Your Knowledge

Chapter 5

Principal Rotation Vectors

5.1 Definitions and Things

Euler Angles are great, but you *really* don't wanna use them for spacecraft or other body frames that can tumble head over heels. Let's try something new and move away from the world of Euler angles.

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 (ϕ rad/deg) around some principal axis ($\hat{e}$). This principal axis is fixed in both the start and end orientation.
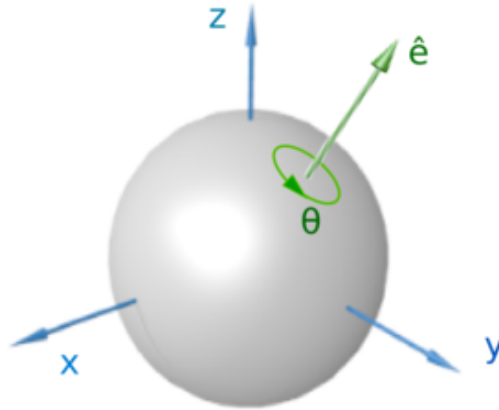
(I know it says θ, just pretend it's ϕ)

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. ϕ, **How Far to Rotate**

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

5.2   PRVs to DCM

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

$$[BN] = \begin{bmatrix} e_1^2 \Sigma + c_\phi & e_1 e_2 \Sigma + e_3 s_\phi & e_1 e_3 \Sigma - e_2 s_\phi \\ e_2 e_1 \Sigma - e_3 s_\phi & e_2^2 \Sigma + c_\phi & e_2 e_3 \Sigma + e_1 s_\phi \\ e_3 e_1 \Sigma + e_2 s_\phi & e_3 e_2 \Sigma - e_1 s_\phi & e_3^2 \Sigma + c_\phi \end{bmatrix}$$

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

Note that the c_ϕ, s_ϕ are just shorthand for $\cos \phi, \sin \phi$

5.3   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

6.1   Definitions and Things

Googling "quaternion math" will probably scare you, so let's leave all that fancy stuff for the math majors. Taken in the context of PRVs however, they're actually pretty straightforward.

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.

6.1.1   Well, what is a quaternion?

Quaternions form a number system that extends the complex number system. It was invented by William Rowan Hamilton back in 1843, after having an epiphany on a bridge near his home in Ireland. They're usually represented with 4 terms:

$$a + bi + cj + dk$$

Where a,b,c,d are the real coefficients and i,j,k are complex. I'm not gonna go too deep into this since I'm certainly not qualified enough to say more.

6.1.2 Defining EPs

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

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

Where β_0 is the real term. The nomenclature will change according to who you ask, so just make sure you clarify.

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

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

6.1.3 Constraints

EPs have only 1 constraint, which is the following:

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

Chapter 7

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 idea 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.

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 this text will not cover.

Chapter 8

TRIAD Method

Published in 1964, TRIAD is one of the oldest algorithms out there. Despite its age, it is still a viable method for simpler satellites.

Chapter 9

What is Wahba's Problem?

9.1 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:

9.1.1 Defining Wahba's Problem

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

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

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

Nothing we don't know (see 3.2.1). 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 this loss function to get the best possible [BN] matrix?"[1]:

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

Let's break this down.

- $J([\tilde{B}N])$: This describes the overall error 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 typically assign this 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 mapped inertial measurement.

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

9.1.2   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 10

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.

10.0.1 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

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

We remake this as the **"gain"**:

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

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

10.1   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.

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

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

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

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

28

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

Finally,

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

Where

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

10.1.1 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.

10.2   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 11

QUEST

Devenport's Q is lovely, but computationally, solving that eigenvector problem is pretty expensive. How about we bypass that eigenvector problem completely?

Chapter 12

Answer Key

12.1 Chapter 1 DCM

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