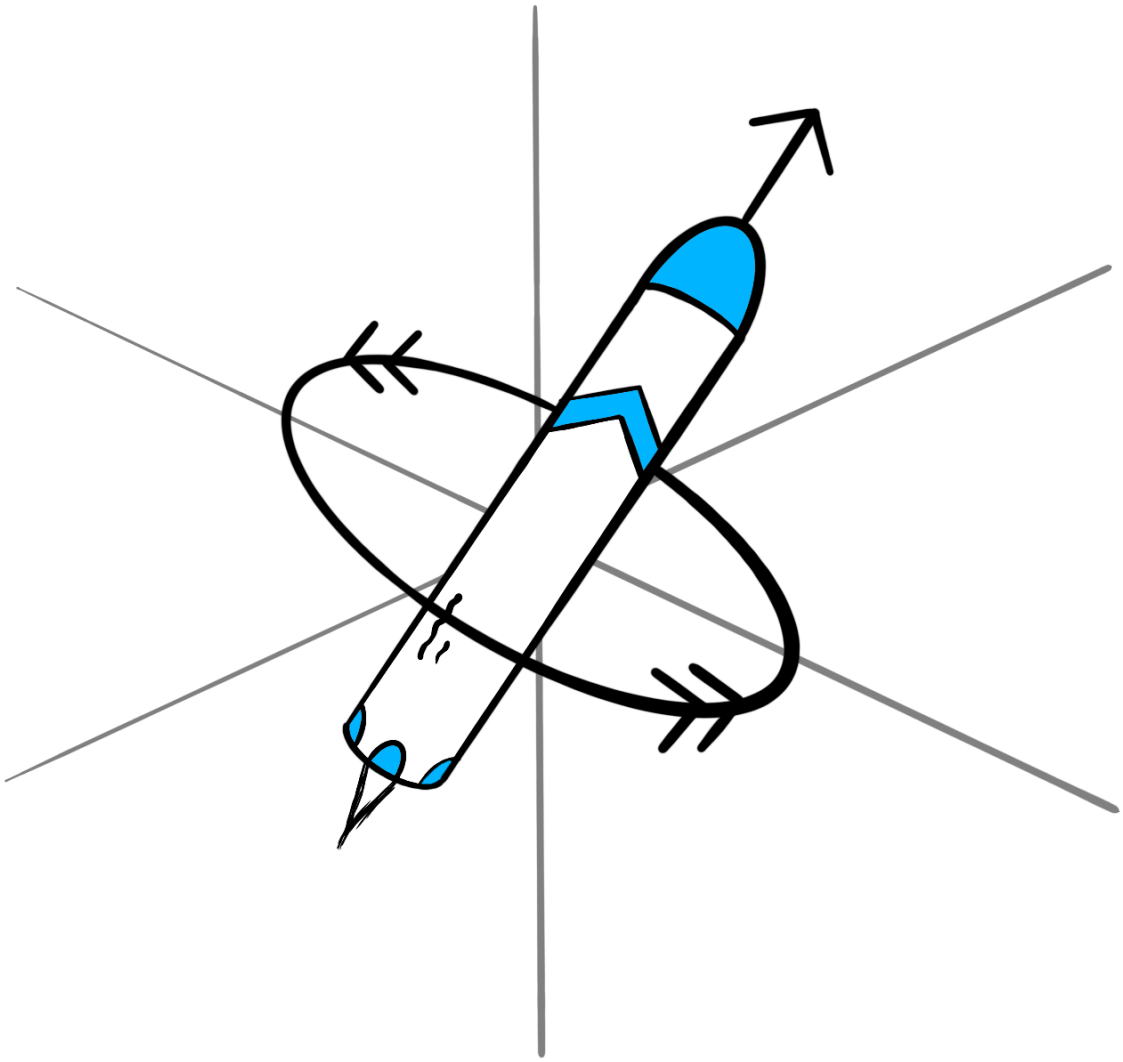
Developing a Quaternion-based PID Guidance Software



Jorge Díaz Chao

Index

Introduction **1**

Part I: Quaternions **2**

Introduction to hypercomplex numbers 2

Stereographic projections 3

Quaternions applied to rotations 9

Part II: Guidance software **14**

PID control theory 14

Quaternion-based PID control architecture 17

Quaternion kinematics 18

Simulation 19

Conclusion **20**

Appendix A: non-commutativity proof 21

Appendix B: PID tuning 21

Appendix C: quaternion calculus 23

Appendix D: simulation details 25

Appendix E: quaternion-based PID code 26

Bibliography 29

Introduction

This work aims to explore the means through which sets of four-dimensional hypercomplex numbers, namely quaternions, can express three-dimensional rotations, and develop a quaternion-based PID guidance architecture to control a rocket's attitude. My interest in this topic originates from the development of an advanced model rocket I have been working on for over a year, which uses motor gimbaling and avionics to re-orient itself. Moreover, quaternions are of great significance in advanced model rocketry and engineering due to the advantages they offer in contrast to other rotation systems, such as computational efficiency or problem minimization. Even more excitingly, this is an outstanding opportunity to apply complex numbers to real life problems. Additionally, the topic is not very well documented and often overlooked, setting a fun mathematical challenge.

Developing a quaternion-based guidance software requires a deep understanding of quaternions, hence, they will be studied through stereographic projections, scaling up from lower dimensions, to understand how these express three-dimensional rotations. This knowledge will be subsequently applied in combination with PID control theory to create a reliable attitude guidance software, tested and tuned using coded flight simulations.

Part I: Quaternions

Introduction to hypercomplex numbers

Quaternions are an absolutely fascinating and often under-appreciated number system from math, regularly defined as a four-dimensional extension of complex numbers, just as complex numbers are a two-dimensional extension of the real numbers. Quaternions play a role in describing three-dimensional rotation and even in quantum physics, while often appearing in mathematics as a particular non-commutative algebraic system. Similar to the often used expression for complex numbers $a + bi$, quaternions are generally presented in the form of:

$$q = q_0 + q_1 i + q_2 j + q_3 k \tag{1}$$

Where $q_0, q_1, q_2, q_3 \in \mathbf{R}$, i, j, k are the fundamental units of quaternions and $q \in \mathbf{H}$.

The product of two quaternions satisfies the fundamental rules introduced by Hamilton:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{2.1}$$

$$ij = -ji = k \qquad jk = -kj = i \qquad ki = -ik = j \tag{2.2) (2.3) (2.4}$$

Note that the multiplication of any two fundamental quaternion units does not commute, but instead one is the negation of the other, as proved by contradiction in Appendix A. These rules of quaternions are defined to be true because the algebraic properties that derive from them. A multiplication table for quaternions can be derived from the basic rules:

	1	i	j	k
1	1	i	j	k
i	i	-1	k	-j
j	j	-k	-1	i
k	k	j	-i	-1

Fig. 1

The product of two quaternions, q and p, is the result of the Hamilton product, determined by the multiplication of the fundamental units and the distributive law such that:

$$q \cdot p = \begin{matrix} q_0p_0 - q_1p_1 - q_2p_2 - q_3p_3 \\ (q_0p_1 + q_1p_0 + q_2p_3 - q_3p_2)i \\ (q_0p_2 - q_1p_3 + q_2p_0 + q_3p_1)j \\ (q_0p_3 + q_1p_2 - q_2p_1 + q_3p_0)k \end{matrix} \tag{3}$$

The norm, conjugate and inverse of a quaternion are defined by eq. (4), (5) and (6) respectively:

$$q = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \tag{4}$$

$$q^* = (q_0 + q_1 + q_2 + q_3)^* = q_0 - q_1 - q_2 - q_3 \tag{5}$$

$$q^{-1} = \frac{q^*}{q} \tag{6}$$

These equations provide a solid foundation for this paper; however, an extension of quaternions' arithmetic may be found in ref. [1].

Stereographic Projections

At first glance, quaternions might result difficult to understand, and perhaps impossible to visualize, due to its natural four dimensional appearance. Thus, tools and techniques such as stereographic

projections are often used to depict these four-dimensional shapes into three dimensional spaces. However, this technique is not limited to four-dimensional shapes, but can also be used for lower dimensions. For instance, stereographic projections can be used to map the surface of the our three-dimensional globe on a two-dimensional plane.

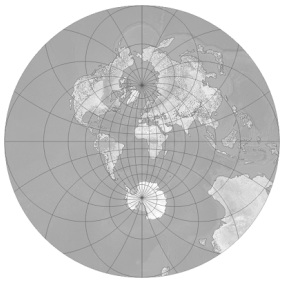


Fig. 2

As common to every non-trivial math problem, simplifying things might be useful to build up intuition. Therefore, stereographic projections will be explained and demonstrated from lower dimensions to the fourth dimension, exploring how they may express rotation along with imaginary numbers, to be analogous with quaternions later on.

Let our two-dimensional plane be defined by the complex plane, and have the unit circle displayed on it containing all unit complex numbers, those with $|z| = 1$, where z is a complex number. For the purpose of this work, note that complex numbers may be seen as a rotation of the unit circle rather than a point on it, where the unit circle will rotate in such a way so that the point originally at 1 moves to the desired unit complex number as shown by fig. 3:

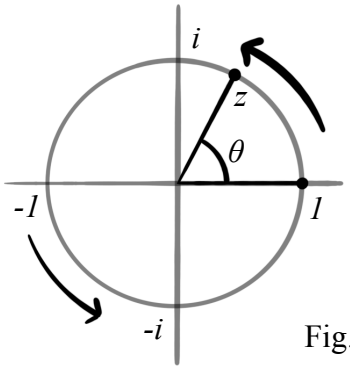


Fig. 3

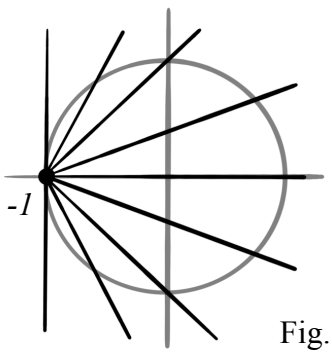


Fig. 4

To perform a stereographic projection it is required to establish a somewhat arbitrary point on the complex plane, let this point be located at *-1* and emit infinite rays which will only intersect the unit circle and the vertical axis once, as depicted by fig. 4. Therefore, that point from the unit circle is said to be projected in the associated and unique intersecting point of the vertical axis. As a result the unit circle is projected on a one-dimensional line rather than a two-dimensional circumference. Every rotation of the unit circle will be perceived as the movement of a reference point on the projected line, as demonstrated by fig. 5, where the two-dimensional rotation from the point z_1 to z_2 is seen as an equivalent vertical translation on the projected line.

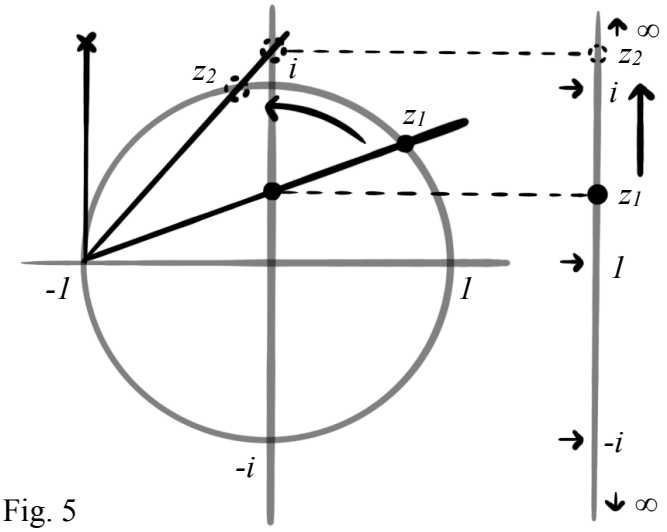


Fig. 5

Take into consideration that stereographic projections have depth limitations, as the projected line only shows those points in the complex plane with $|z| = 1$. Using only unit complex numbers, or an equivalent, ensures the complex coordinate is contained by the unit circle and will therefore be projected on the line.

Through experimentation using an an interactive stereographic projection simulation retrieved from ref. [2], observations can be made. Changing directly the real component of the complex number, $a + bi$, expressing a rotation, will determine the distance of the reference point from the centre, on the projected line. While modifications of the imaginary part will translate the point within a certain range, from i to $-i$, if the real part is positive, for instance. From these patterns the following deduction can be made: different coefficients correspond to particular rotations and they are combinable to achieve a desired orientation.

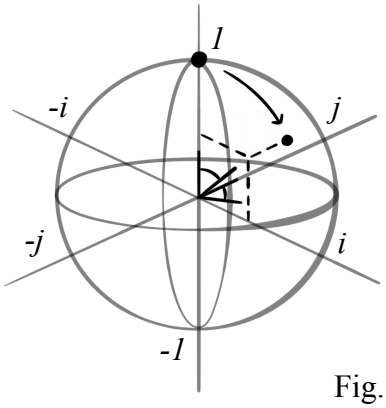


Fig. 6

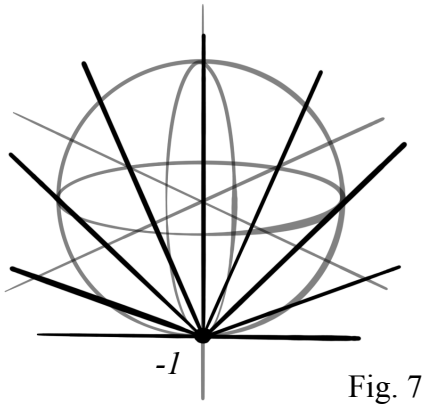


Fig. 7

The same technique may be applied to a three-dimensional space, with a vertical axis denoted by the real number line, and the horizontal plane spanned by the imaginary number i and j axes, such that any point on this field is defined by a vector-like group of non-existing numbers, $a + bi + cj$, as depicted by fig. 6.

Let the projection point, common to all stereographic projections, be located at the point -1, and emit infinite rays in all direction, as shown in fig. 7, so that they intersect with the surface of the unit sphere and the horizontal plane once each, as noted by fig. 8 and 9. The surface of the three-dimensions sphere will be projected on a two-dimensional plane just like the unit circle got projected into a line. Any translation of the point originally at 1, determined by the vector $[a \quad b \quad c]$

of unit length so it is contained within the sphere of radius 1, will be seen as a rotation of the sphere as it adapts to neglect any relative motion between the point and the sphere.

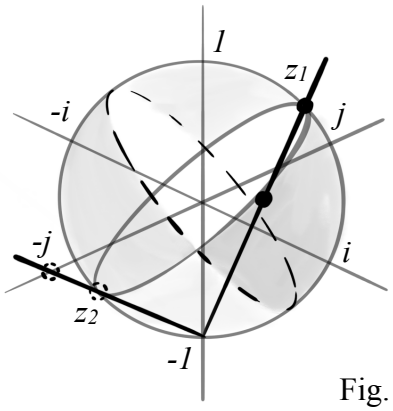


Fig. 8

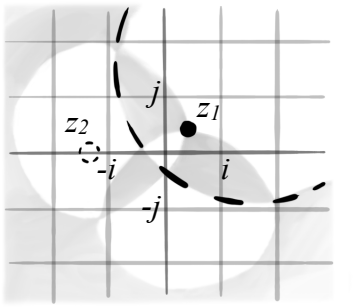


Fig. 9

Through interactive simulations, it can be observed that modifications to the imaginary components will determine the direction of the reference point in the projected plane, while changes in the real part will determine how far the point is from the centre. Note the difference between points $z_1 = a_1 + b_1 i + c_1 j$ and $z_2 = a_2 + b_2 i + c_2 j$ in fig. 9, where $a_1 > a_2$, $b_1 < b_2$ and $c_1 < c_2$. Not only coefficient modifications are combinable to achieve a desired orientation, a two-dimensional rotation may be retrieved from the stereographically projected three-dimensional rotating sphere, which is the concept behind quaternions.

Three-dimensional stereographic projections, however, fail at describing three-dimensional rotation. Only 2 degrees of freedom may be obtained from $a + bi + cj$, given that its modulus is constrained to be 1 so that the point is within the unit sphere's surface. The equator's rotation of the sphere in its original three-dimensional field can only be modified through an external factor, instead, quaternions might be used as they provide this additional degree of freedom with their additional dimension of imaginary numbers, $a + bi + cj + dk$.

Four-dimensional stereographic projections are harder to depict as the original appearance is beyond our perception, unlike a three-dimensional sphere or a two-dimensional circle, a four-dimensional hypersphere can't be visualised previous to its projection onto a lower dimension. However, trends in lower dimension stereographic projections allow to understand, visualize and compute these four-dimensional stereographic projections, whose axes and reference points are predicted by fig. 10, such that:

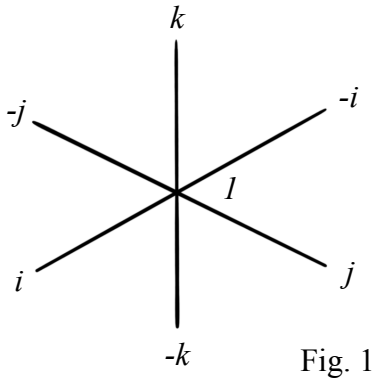


Fig. 10

- The unit four-dimensional hypersphere defined by unit quaternions with $q = 1$ is projected in a three-dimensional field, initially as a unit sphere.
- Originally, the axes appearing in the three-dimensional space are the i, j and k axes, while the real number line is invisible to the projection, but perpendicular to every other axis.
- The projection point is located at -1, projected at infinity and therefore neglected in the projection, while the point at 1 gets projected at the centre of the space.
- Unit quaternions with no real part, otherwise known as unit vectors, get projected on a unit sphere passing though points i, j, k and their opposites shown in fig 10 as reference points.
- The line passing through 1 and i and the unit sphere crossing i, j and k are analogous to the projected circle from the two-dimensional stereographic projection and the projected plane in the three-dimensional stereographic projection respectively.
- The reference points will translate when multiplied by rotation quaternions according to eq. (2).

Quaternions Applied to Rotations

Visualizing quaternions through stereographic projections makes possible understanding their application to rotations. Two-dimensional rotations can be expressed as the multiplication of various unit complex numbers, whose modulus equals *1*, resulting in another unit complex number, such that $z_1 \cdot z_2 = z_3$, where $z_1 = z_2 = z_3 = 1$. This process involves adding the arguments of the complex numbers, while the magnitude remains constant, producing a rotation appreciable in the complex plane, as seen in fig. 11:

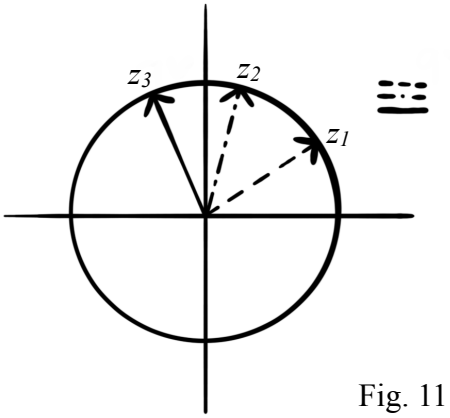


Fig. 11

Deductively, this logic extends to quaternions, given that the multiplication of two unit quaternion results in another unit quaternion. The multiplication of two rotation quaternions $q \cdot p$ will be equivalent to the rotation p followed by the rotation q. However, this results in another rotation quaternion and not any real point in a three-dimensional field undergoing a rotation of any kind. An approach to apply rotation quaternions to our three-dimensional world will be presented hereunder.

From the fundamental rules of quaternions the translation for every point projected on a stereographic projection, represented by *v,* can be predicted, as seen in fig. 12.1 and 12.2. When

these reference points are multiplied by a rotation quaternion, their translation may be perceived as a set of rotation on the different axes of the stereographic projection as depicted by the fig. 12.3:

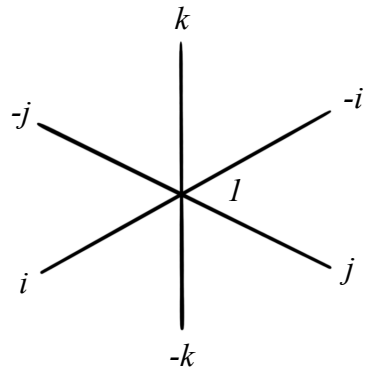


Fig. 12.1

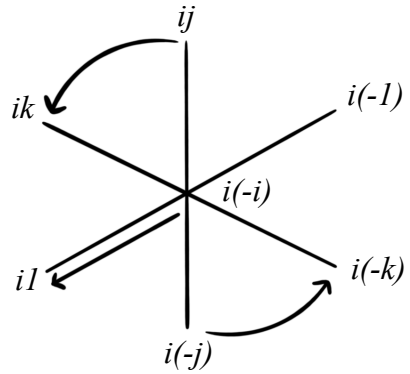


Fig. 12.2

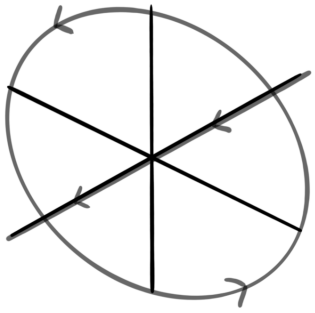
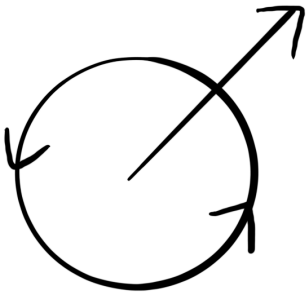


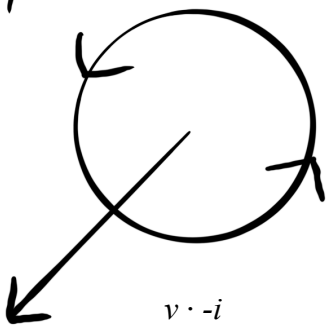
Fig. 12.3

When multiplied by the unit quaternion i, or $0 + 1i + 0j + 0k$, the reference points 1, i, j, k, and their opposites on the four-dimensional stereographic projection move to their corresponding new values, determined by fig. 1. For example, the point 1 moves to i as $i \cdot 1 = i$, and j moves to k as $i \cdot j = k$. From these translations two separate rotations can be noticed, the circle passing through j and k rotating anti-clockwise, and the line passing through 1 and i rotating linearly, similar to the projected rotations seen in three-dimensional and two-dimensional stereographic projections respectively. However, quaternion multiplication is non-commutative, meaning that the order of operations may influence the product as shown by the comparison between fig. 13.1 and 13.2:



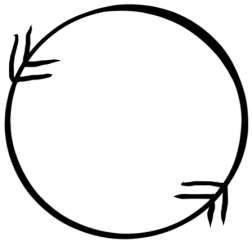
$i \cdot v$

Fig. 13.1



$v \cdot -i$

Fig. 13.2



$i \cdot v \cdot -i$

Fig. 13.3

If the reference points were to be left multiplied by $-i$ instead, the same rotations would occur but on opposite directions. However, if right multiplied by $-i$ only the line passing through 1 and i would rotate on an opposite direction, as it is clearly depicted by fig. 13.2. Arranging this set of operations appropriately, such that $i \cdot v \cdot i^*$, where i^* is $-i$, allows to cancel one of the rotations resulting on a doubled rotation around a desired axis, as demonstrated by fig. 13.3.

This idea may be generalized such that:

$$v_{n+1} = q \cdot v_n \cdot q^* \tag{7}$$

Where v is a the three-dimensional vector $\begin{bmatrix} v_1^n & v_2^n & v_3^n \end{bmatrix}$ retrieved from $v_n = v_0^n + v_1^n i + v_2^n j + v_3^n k$, v_{n+1} is the rotated vector, and q is a unit quaternion with $\|q\| = 1$ that determines the angle and axis of rotation.

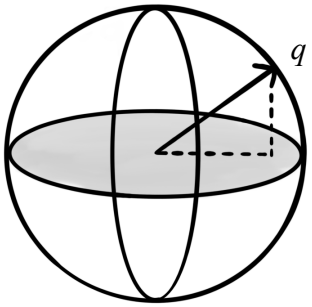


Fig. 14

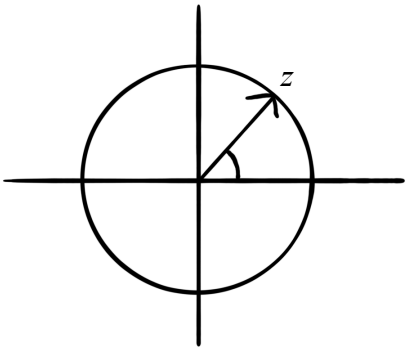


Fig. 15

Where the shaded plane is the three-dimensional ijk space, and the axis perpendicular to that is the real axis.

These properties lead to the alternative expression of unit quaternions:

$$\begin{gathered} q = cos(\frac{\alpha}{2}) + usin(\frac{\alpha}{2}) \\ q = cos(\frac{\alpha}{2}) + sin(\frac{\alpha}{2})(xi + yj + zk) \end{gathered} \tag{8}$$

Where u is a unit vector defining the rotation axis and α is the angle of rotation. Similar to complex numbers in modulus-argument form, defined as $z = cos(\theta) + isin(\theta)$, and graphically appearing as in fig. 15. The unit vector u ensures that $|q| = 1$, since $cos^2(\theta) + sin^2(\theta) = 1$ is always true. This alternative way to visualize quaternions can be represented graphically as:

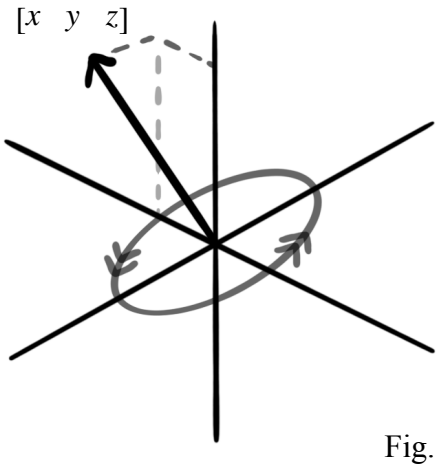


Fig. 16

Since the rotation resulting from $q \cdot v \cdot q^*$ is doubled in the desired rotation axis and cancelled in the other, as seen in fig. 13, there are two ways to specify any orientation in a three-dimensional space using quaternions. This ends up being useful for interpolation, as it provides flexibility by allowing to take shorter or longer paths to a new orientation. If q rotates v to a particular orientation, $-q$ will also rotate v to that same orientation, notice that:

$$-q \cdot v \cdot (-q)^* = -(-q) \cdot v \cdot q^{(*)} = q \cdot v \cdot q^{(*)} \tag{9}$$

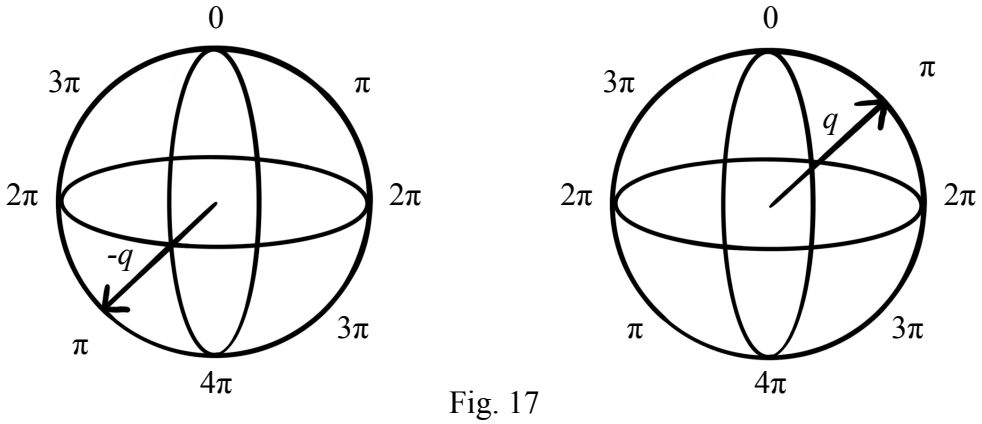


Fig. 17

And that is how quaternions may be used to describe three-dimensional rotations in a more computationally efficient way, allowing for enhanced interpolations between rotations and avoid gimbal lock, the loss of one degree of freedom common to Euler angles, which makes them unsuitable for some practical applications.

Euler angles refer to the common set of three angles that describe orientation around three different axes, with respect to a fixed coordinate system. However, conversions between Euler angles and quaternions are very useful for attitude interpretation and setting up the control system. Hence:

Euler to quaternion:

$$q_0 = cos(\frac{e_x}{2}) \cdot cos(\frac{e_y}{2}) \cdot cos(\frac{e_z}{2}) + sin(\frac{e_x}{2}) \cdot sin(\frac{e_y}{2}) \cdot sin(\frac{e_z}{2}) \quad (10.1)$$

$$q_1 = sin(\frac{e_x}{2}) \cdot cos(\frac{e_y}{2}) \cdot cos(\frac{e_z}{2}) - cos(\frac{e_x}{2}) \cdot sin(\frac{e_y}{2}) \cdot sin(\frac{e_z}{2}) \quad (10.2)$$

$$q_2 = cos(\frac{e_x}{2}) \cdot sin(\frac{e_y}{2}) \cdot cos(\frac{e_z}{2}) + sin(\frac{e_x}{2}) \cdot cos(\frac{e_y}{2}) \cdot sin(\frac{e_z}{2}) \quad (10.3)$$

$$q_3 = cos(\frac{e_x}{2}) \cdot cos(\frac{e_y}{2}) \cdot sin(\frac{e_z}{2}) - sin(\frac{e_x}{2}) \cdot sin(\frac{e_y}{2}) \cdot cos(\frac{e_z}{2}) \quad (10.4)$$

Quaternion to Euler:

$$e_x = atan2(2 \cdot (q_0 \cdot q_1 + q_2 \cdot q_3), q_0^2 - q_1^2 - q_2^2 + q_3^2) \quad (11.1)$$

$$e_y = asin(2 \cdot (q_0 \cdot q_1 - q_2 \cdot q_3)) \quad (11.2)$$

$$e_z = atan2(2 \cdot (q_0 \cdot q_1 + q_2 \cdot q_3), q_0^2 + q_1^2 - q_2^2 - q_3^2) \quad (11.3)$$

Part II: Guidance Software

Directing a rocket's thrust to control its angular velocity, known as thrust vectoring, requires a feedback control system. Namely, a proportional-integral-derivative (PID) controller that returns a corrective output based on an error input, attempting to reach a null error. As the rocket's attitude deviates from the desired orientation the error is monitored and the rocket's thrust is directed accordingly to minimize the error. Hence, a quaternion-based PID control architecture for rockets' attitude will be developed.

PID control theory

Virtually, PID controllers calculate an error value and modify a system correspondingly in order to minimize it. Error values e, are obtained by calculating the deviation between a desired d, and actual value a, of the controlled system. When dealing with Euler angles, the error may be calculated through simple arithmetic. For instance, if the rocket's desired attitude is set to be 0^o, but it is actually 5^o, then the error value would be the difference.

$$e(t) = a(t) - d(t) \tag{12}$$

Note that values may be updated, derived and integrated over time, thus, are presented as a function of time. The PID controller, eq. (13.1), will take the error to compute a corrective value, c, to modify the system. Calculations use three control terms, proportional, integral, and derivative, eq. (13.2), (13.3) and (13.4) respectively, making optimal control possible.

$$c(t) = c_p(t) + c_i(t) + c_d(t)$$
$$c(t) = k_p \cdot e(t) + k_i \cdot \int_0^t e(t)dt + k_d \cdot \frac{\mathrm{d}}{\mathrm{d}t}e(t) \tag{13.1}$$

These, however, are dependant on a set of coefficients correspondent to each term, k_p, k_i and k_d, that require to be tuned either experimentally or mathematically, as discussed in Appendix B.

$$c_p(t) = k_p \cdot e(t) \tag{13.2}$$

As the name suggests, the proportional term, eq. (13.2), offers a proportional corrective contribution c_p, to a present error value. Therefore, as the error increases the outputted corrective value also will. Similarly, as the error approaches *0*, the corrective value will decrease until it stops contributing to the correction. For example, if the rocket's thrust requires to be angled at 0.5^o to compensate a 1^o deviation error, the proportional gain k_p, may be *0.5*. If this is the case, whenever the error is 5^o, the proportional term will angle the thrust to 2.5^o.

$$c_i(t) = k_i \cdot \int_0^t e(t)dt \tag{13.3}$$

However, an only proportional controller won't be able to account for cumulative error, making the system very sensible to imperfections, or even unfunctional for systems that require a constant corrective response, that won't be provided by the proportional term if the error is null. Solving this requires recording past error values, which the integral term, eq. (13.3), sums up over time keeping a running total, allowing a corrective contribution c_i, not only related to the error, but also the time for which it has persisted. In other words, the integral term accounts for steady state error encountered in all imperfect systems, defined as the difference between the desired and actual value

when the proportional output is steady, providing a corrective response to unexpected errors. In rocketry, the integral term can be seen in action whenever there is a misalignment of the rocket's motor. If this is the case, despite the error being null the thrust will continue exerting an angular moment on the rocket due to its misaligned angle, increasing the error. By recording this error and integrating it, the integral term could account for the misalignment. The integral gain k_i, will determine how sensitive the PID controller is to this accumulated error as they are multiplied.

$$c_d(t) = k_d \cdot \frac{\mathrm{d}}{\mathrm{d}t} e(t) \tag{13.4}$$

Similarly to how the integral term looks at the past, the derivative term, eq. (13.4), estimates the future error value by calculating its recent rate of change. Hence, the derivative term seeks to anticipate future errors. If the absolute error is increasing rapidly, the corrective response c_d, will be greater in order to decrease the error's incremental rate avoiding further errors. Likewise, if the absolute error is decreasing swiftly, the corrective contribution will negatively, decreasing the overall correction and avoiding any overshoot. A greater difference between time adjacent rotations translates to higher angular velocities, thus, the error will vary more rapidly and require a greater corrective value. Again, the derivative gain k_d, will define how sensitive the PID controller is to the error's rate of change.

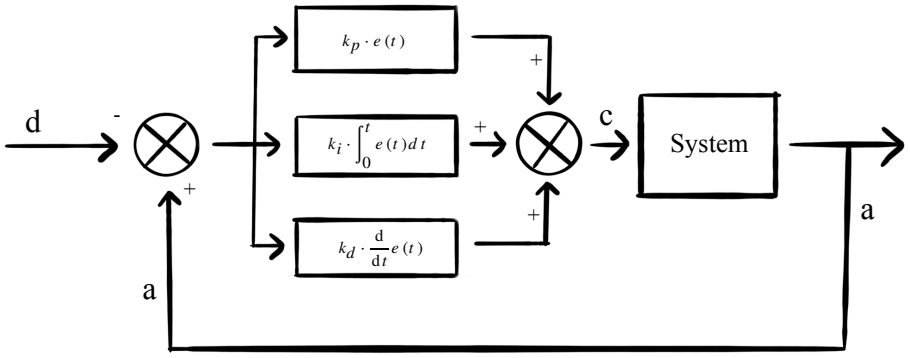


Fig. 18

The resultant corrective value c, obtained by adding c_p, c_i and c_d is subsequently outputted by the PID algorithm and applied to the controlled system, changing the actual value, which will be fed back again into the control system summarised by the following diagram:

Quaternion-based PID control architecture

The combination of the proportional, integral and derivative terms offer great reliability. Nonetheless, the integration of quaternions to a PID controlled non-linear system does increase the mathematical complexity.

Due to the nature of quaternions, the deviation q_e between the rocket's current q_a, and desired q_d rotations, denoted by quaternions relative to a common frame, cannot be calculated through simple arithmetic. The product of two rotation quaternions, the error value q_e and q_a, will be equivalent to the rotation q_a followed by the rotation q_e, resulting in q_d. Thus, the error is to be determined by:

$$q_e(t) \cdot q_a(t) = q_d(t)$$
$$q_e(t) = q_a^{-1}(t) \cdot q_d(t)$$

Given that q_a is a rotation quaternion, and therefore unit, $q_a^{-1} = \frac{q_a^*}{q_a} = \frac{q_a^*}{1} = q_a^*$, hence:

$$q_e(t) = q_a^*(t) \cdot q_d(t) \tag{14}$$

The error is null when $q_e = 1 + 0i + 0j + 0k$, which translates to a rotation of 0^o on all axes. Raw error values in the form of quaternions may require a transformation to axis error as a three-dimensional vector, obtained from the alternative definition of a quaternion presented by eq. (8),

which relies on the angle and axis of rotation, denoted by unit a vector, from which the magnitude of the rotation on each axis can be retrieved.

$$x_e(q_e) = \begin{bmatrix} q_1^e \\ q_2^e \\ q_3^e \end{bmatrix} \tag{15}$$

This new three-dimensional vector containing the axis error values is inputted PID controller, eq. (13.1), to define the corrective angular velocity ω_c that the thrust must exert on the rocket, based on a set of previously tuned proportional, integral and derivative terms, such that:

$$\omega_c(t) = (k_p + k_i\frac{1}{s} + k_d\frac{1}{1+N\frac{1}{2}}) \begin{bmatrix} q_1^e(t) \\ q_2^e(t) \\ q_3^e(t) \end{bmatrix} \tag{16}$$

Where $\frac{1}{s}$ and $\frac{1}{1+N\frac{1}{2}}$ integrate and derive the axis error with respect to time respectively.

Quaternion kinematics

Further kinematic calculations may be performed using the corrective or recorded angular velocities and the quaternion kinematic equations, retrieved through the time derivatives and integrals of quaternions, whose obtainment is further developed in Appendix C.

$$\frac{\mathrm{d}q(t)}{\mathrm{d}x} = \lim_{\Delta t \to 0} \frac{q(t+\Delta t) - q(t)}{\Delta t}$$

$$\frac{\mathrm{d}q(t)}{\mathrm{d}x} = \frac{1}{2}\omega(t)q_a(t) \tag{17}$$

Where ω is the corrective or recorder angular velocity, and q_a is the actual orientation. Rotation over time may be accumulated by integrating the differential equation appropriate to the rotation rate definition, from which the future rotation q_{a+1} can be calculated by:

$$q_{a+1} = [cos\frac{\omega \cdot \Delta t}{2}, sin\frac{\omega \cdot \Delta t}{2} \cdot \frac{\omega}{\omega}] \cdot q_a \tag{18}$$

This new attitude would then be fed back into the control loop outlined by the following figure:

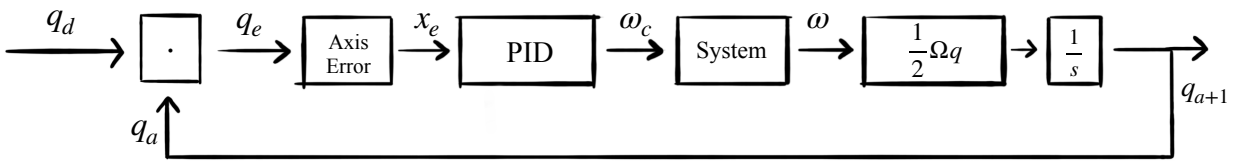


Fig. 19

This control scheme for a rocket's attitude provides better computational performance and increased reliability over guidance software based on Euler angles. The developed control architecture utilizes quaternions for all calculations and measurement, with no Euler angles transformations, maintaining simplicity and preventing singularity issues common to this alternative form of rotation expression.

Simulation

The designed quaternion-based PID guidance software has been tested with a simulation running on Swift. The code developed may be found in Appendix E. The graph of fig. 20 demonstrates the effectiveness of this more efficient and reliable control architecture, although its performance may vary upon its tuning. More details of the simulation are discussed in Appendix D.

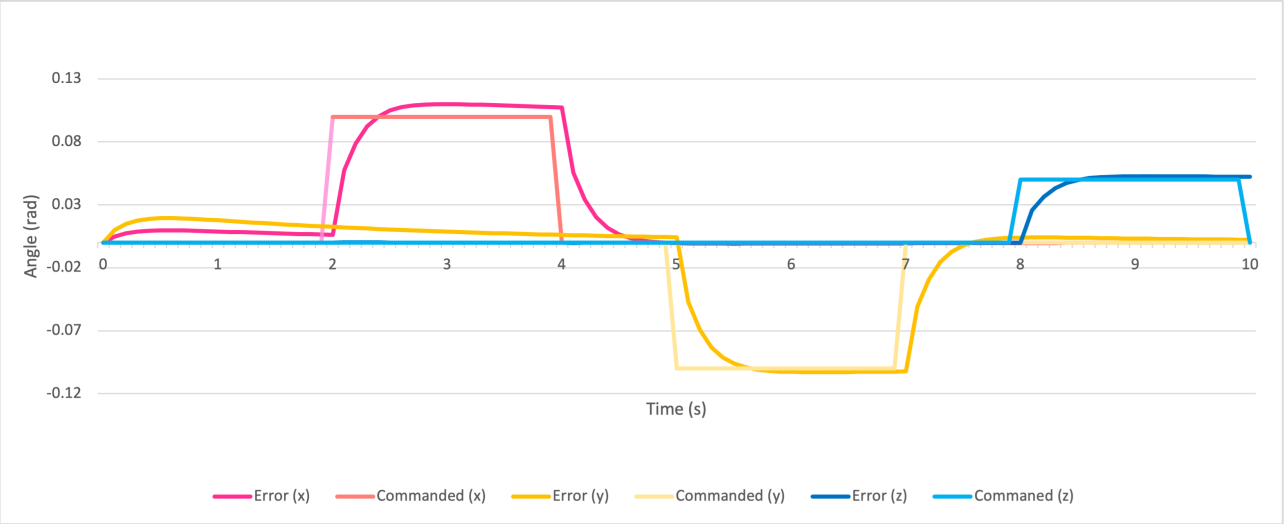


Fig. 20

Conclusion

Wrapping up, quaternions are a fascinating tool worth of its complexity as they allow for great reliability and computationally efficiency, very useful for engineering. This paper studied in detail the nature and visualization of four-dimensional hypercomplex numbers, initially seen as an impossible task. Later, this study was used to develop a quaternion-based PID control architecture combining the application of quaternions to rotation and PID control theory. The guidance software demonstrated to be effective and will be featured in future advanced model rocket flights as well as in flight simulation software. Given that changes in mass or thrust during a rocket flight may require different PID gains for optimal compensation, the control architecture could be improved by making it adaptive to changes in the environment, rather that constraining its PID gains to a set of previously tuned values. Overall, this investigation brought me a step closer to the flight of the advanced model rocket I am currently developing and only dreamt of to this date, allowing me to design its guidance softwares, while setting a very complicated and fun mathematical experience.

Appendix A

Non-commutativity proof

The non-commutative properties of quaternions may be proved by contradiction assuming quaternion multiplication was commutative:

$$ijk \cdot = -1 \cdot i \rightarrow i^2jk = -i \rightarrow -jk = -i \rightarrow jk = i \qquad (A.1)$$

$$ijk \cdot jk = -1 \cdot jk \rightarrow ij^2k^2 = -jk \rightarrow i = -jk \rightarrow jk = -i \qquad (A.2)$$

$$jk = i \not\equiv jk = -i$$

Hence, by contradiction it can be proven that the commutative rule $ab = ba$ does not apply:

$$jk \neq kj \rightarrow i \neq -i$$

Appendix B

PID tuning

Despite their reliable architecture, PID controllers will only be effective under the right set of proportional, integral and derivative gains used to calculate the correction. Therefore, these values need to be finely tuned, which might be done experimentally, or mathematically through the use of simulations. From the nature of PID controllers, the following manual tuning method can be derived for the effects of increasing the gains independently:

	Rise	Overshoot	Settling time	Steady-state error	Stability
$\mathbf{k_p}$	Decrease	Increase	Minor change	Decrease	Degrade
$\mathbf{k_i}$	Decrease	Increase	Increase	Decrease	Degrade
$\mathbf{k_d}$	Minor change	Decrease	Decrease	No effect	Improve

Alternatively or supplementarily, PID gains may be adjusted to the following constrains:

1. $$\frac{\mathrm{d}}{\mathrm{d}x}_{t=t_d} = \lim_{t \to t_d} \omega(t) = 0 \qquad (B.1)$$

2. $$\int_0^{t_d} \tau_c(t)dt = \int_0^{t_d} \tau_a(t)dt \qquad (B.2)$$

3. Lowest possible value for t_d

Where ω is the angular velocity, t_d is the time t at which the error rotation $q_e = 1 + 0i + 0j + 0k$, and τ_a is the set of angular forces acting on the rocket, and τ_c is the corrective angular force exerted by the motor on the rocket and controlled by the quaternion-based PID algorithm. As this force is proportional to some variable factors such as the rocket's dimensions, mass and the motor's distance from the centre of gravity, the PID gains may adjust differently. For instance, higher rotational moments of inertia will translate to higher proportional gains as it requires a greater force to create a corrective moment, and lower derivative gains as error will increase at lower rates.

Alternative proven mathematical approaches to tune PID controllers are discussed in ref. [3].

Appendix C

Quaternion calculus

Time derivatives of quaternion rotations may be developed through the local variations in a vector space. The definition of the derivative is applied:

$$\frac{\mathrm{d}q(t)}{\mathrm{d}x} = \lim_{\Delta t \to 0} \frac{q(t+\Delta t) - q(t)}{\Delta t} \tag{C.1}$$

Where the original orientation $q_a = q(t)$, the rotated attitude $q_{a+1} = q(t+\Delta t) = q_a \cdot \Delta q$, and Δq is the local rotation quaternion, easily obtained from its local angular change $\Delta\phi$ corresponding to the angular rate vector:

$$\tilde{q} = \Delta q \cdot e^{\Delta\phi} \tag{C.2}$$

$$\Delta\phi = log(\Delta q^* \cdot \tilde{q}) \tag{C.3}$$

The local angular change $\Delta\phi$ can be approximated by the Taylor expansion in eq. (C.4):

$$q = e^{\phi u/2} = cos(\frac{\phi}{2}) + u \cdot sin(\frac{\phi}{2}) = \begin{bmatrix} cos(\frac{\phi}{2}) \\ u \cdot sin(\frac{\phi}{2}) \end{bmatrix} \tag{C.4}$$

$$\Delta q \approx \begin{bmatrix} 1 \\ \Delta\phi/2 \end{bmatrix} \tag{C.5}$$

Since, $\Delta\phi/2$ is equivalent to the angular velocity ω, the quaternion derivative can be simplified:

$$\begin{aligned}\frac{\mathrm{d}q(t)}{\mathrm{d}x} &= \frac{q_a \cdot \Delta q - q_a}{\Delta t} \\ &= \frac{q_a \cdot \left(\begin{bmatrix} 1 \\ \Delta\phi/2 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right)}{\Delta t} \\ &= \frac{q_a \cdot \begin{bmatrix} 0 \\ \Delta\phi/2 \end{bmatrix}}{\Delta t} \\ &= \frac{1}{2} \cdot q_a \cdot \omega\end{aligned} \tag{C.6}$$

There are various integration methods for unit rotation quaternions. The designed quaternion-based PID control architecture, however, involved the direct multiplication method, which may be seen in more detail in ref. [4]. Given the eq. (C.6), the rotation R an exponential map is introduced:

$$R(\omega, \Delta t) = e^{\Omega \Delta t} \tag{C.7}$$

Where:

$$\Omega = \begin{bmatrix} 0 & -\frac{\omega_x}{2} & -\frac{\omega_y}{2} & \frac{\omega_z}{2} \\ \frac{\omega_x}{2} & 0 & \frac{\omega_z}{2} & -\frac{\omega_y}{2} \\ \frac{\omega_y}{2} & -\frac{\omega_z}{2} & 0 & \frac{\omega_x}{2} \\ \frac{\omega_z}{2} & \frac{\omega_y}{2} & -\frac{\omega_x}{2} & 0 \end{bmatrix} \tag{C.8}$$

The exponential term in eq. (C.7) is expanded in a Maclaurin series and is simplified to:

$$R(\omega, \Delta t) = \left(cos(\frac{\omega}{2})I + \frac{2}{\omega} \cdot sin(\frac{\omega}{2})\Omega \right) \Delta t \tag{C.9}$$

Where I is a forth-order identity matrix, and the vector ω is the angular velocity. The matrix R can also be represented as a unit quaternion, and multiplied by the actual orientation to calculate the following attitude as a quaternion:

$$q_{a+1} = [cos\frac{\omega \cdot \Delta t}{2}, sin\frac{\omega \cdot \Delta t}{2} \cdot \frac{\omega}{\omega}] \cdot q_a \qquad (C.10)$$

Appendix D

Simulation details

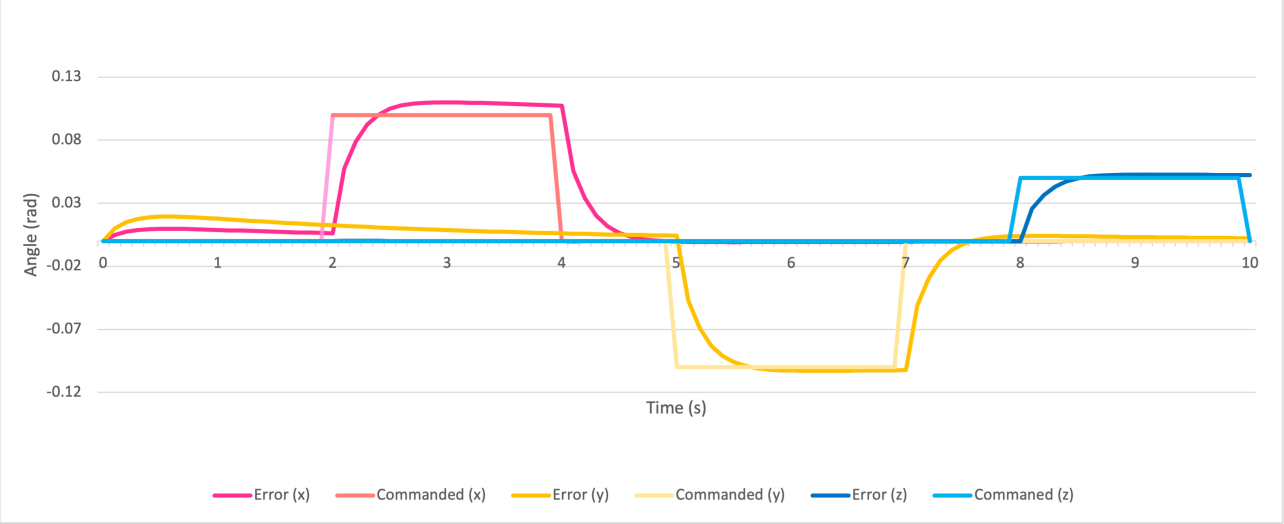


Fig. D.1

The simulation runs on the Swift code presented in the following appendix, and follows the control architecture simplified in fig. 19. Note that the results are presented as Euler angles in radians across the different axes of our three-dimensional world x, y and z; however, the control system only relies on quaternions without compromising reliability nor performance. The effectiveness of the developed quaternion-based PID control architecture is proven by the error minimization seen in fig. D.1. Its performance, however, depends on the set of PID gains previously tuned.

Different orientations for the axes x, y and z are commanded at seconds 2, 5 and 8 respectively. The system calculates the deviation between the actual and desired attitude, represented by quaternions, and is then transformed into a three-dimensional vector that accounts for the deviation in each axis individually through eq. (15). The axis error is inputted into the PID algorithm, eq. (16), which returns the commanded angular rate in order to minimize the error. The angular velocity was constrained within a range of values accounting for the limitation that the system may offer, like the thrust of a rocket, whose angular force is limited by a series of factors. Disturbances like wind were also simulated by adding the angular velocity exerted on the system to the total angular rate, then used in eq. (18) to calculate the new orientation of the system.

Appendix E

Quaternion-based PID code

```
import Foundation
import simd

extension Comparable {
    func clamped(to limits: ClosedRange<Self>) -> Self {
        return min(max(self, limits.lowerBound), limits.upperBound)
    }
}

#if swift(<5.1)
extension Strideable where Stride: SignedInteger {
    func clamped(to limits: CountableClosedRange<Self>) -> Self {
        return min(max(self, limits.lowerBound), limits.upperBound)
    }
}
#endif

func quaternionToEuler(_ q: simd_quatd) -> simd_double3 {

    var e = simd_double3()

    e.x = atan2(2 * (q.real * q.imag.x + q.imag.y * q.imag.z), pow(q.real, 2) - pow(q.imag.x, 2) - pow(q.imag.y,
2) + pow(q.imag.z, 2))

    e.y = asin(2 * (q.real * q.imag.y - q.imag.x * q.imag.z))

    e.z = atan2(2 * (q.real * q.imag.z + q.imag.x * q.imag.y), pow(q.real, 2) + pow(q.imag.x, 2) - pow(q.imag.y,
2) - pow(q.imag.z, 2))
```

```
    if abs(e.x) == .pi {
        e.x = 0
        e.y = .pi - e.y
        e.z = -1 * (.pi - e.z)
    }

    return e
}

func eulerToQuaternion(_ e: simd_double3) -> simd_quatd {

    var q = simd_quatd()

    q.real = cos(e.x / 2) * cos(e.y / 2) * cos(e.z / 2) + sin(e.x / 2) * sin(e.y / 2) * sin(e.z / 2)

    q.imag.x = sin(e.x / 2) * cos(e.y / 2) * cos(e.z / 2) - cos(e.x / 2) * sin(e.y / 2) * sin(e.z / 2)

    q.imag.y = cos(e.x / 2) * sin(e.y / 2) * cos(e.z / 2) + sin(e.x / 2) * cos(e.y / 2) * sin(e.z / 2)

    q.imag.z = cos(e.x / 2) * cos(e.y / 2) * sin(e.z / 2) - sin(e.x / 2) * sin(e.y / 2) * cos(e.z / 2)

    return q
}

func error(a: simd_quatd, d: simd_quatd) -> simd_quatd {
    return d * simd_conjugate(a)
}

func errorToAxis(e: simd_quatd) -> simd_double3 {
    return simd_double3(x: e.imag.x, y: e.imag.y, z: e.imag.z)
}

let timeInterval = 0.1
var previousError = simd_double3()
var accumulatedError = simd_double3()

func PID(e: simd_double3, kp: Double, ki: Double, kd: Double) -> simd_double3 {

    let p = kp * e

    accumulatedError += (e * timeInterval)
    let i = ki * accumulatedError

    let d = kd * (e - previousError)
    previousError = e

    return p + i + d
}

func rotate(a: simd_quatd, w: simd_double3) -> simd_quatd {
    let rotation = simd_quatd(real: cos((length(w) * timeInterval) / 2), imag: sin((length(w) * timeInterval) / 2)
* (w / length(w)))
    return rotation * a
}

var actualAttitude = [eulerToQuaternion(simd_double3(x: 0.0, y: 0.0, z: 0))]
var desiredAttitude = [eulerToQuaternion(simd_double3(x: 0, y: 0, z: 0)),
                       eulerToQuaternion(simd_double3(x: 0.1, y: 0, z: 0)),
                       eulerToQuaternion(simd_double3(x: 0, y: -0.1, z: 0)),
                       eulerToQuaternion(simd_double3(x: 0, y: 0, z: 0.05))]
```

```
var disturbance = simd_double3(x: 0.05, y: 0.1, z: 0)
var actualError = [simd_quatd]()
var time = [Double]()

let kp = 9.0
let ki = 3.0
let kd = 1.0

for i in 0...100 {
    time.append(timeInterval * Double(i))

    switch time[i] {
    case 0..<2:
        actualError.append(error(a: actualAttitude[i], d: desiredAttitude[0]))
    case 2..<4:
        actualError.append(error(a: actualAttitude[i], d: desiredAttitude[1]))
    case 5..<7:
        actualError.append(error(a: actualAttitude[i], d: desiredAttitude[2]))
    case 8..<10:`2
        actualError.append(error(a: actualAttitude[i], d: desiredAttitude[3]))
    default:
        actualError.append(error(a: actualAttitude[i], d: desiredAttitude[0]))
    }
    let axisError = errorToAxis(e: actualError[i])

    var w = PID(e: axisError, kp: kp, ki: ki, kd: kd)
    w = simd_double3(x: w.x.clamped(to: -1...1),
                     y: w.y.clamped(to: -1...1),
                     z: w.z.clamped(to: -0.5...0.5))
    actualAttitude.append(rotate(a: actualAttitude[i], w: (w + disturbance)))

    let actualAngles = quaternionToEuler(actualAttitude[i])
}
```

Bibliography

[1] - Eberly, D. (2010, August 18). Quaternion Algebra and Calculus. Redmond WA 98052; Geometric Tools. Retrieved from https://www.geometrictools.com/Documentation/Quaternions.pdf

[2] - Sanderson, G., & Eater, B. (2018, October 26). *Visualizing quaternions, an explorable video series*. Eater. Retrieved from https://eater.net/quaternions

[3] - Lequin, O., Gevers, M., Mossberg, M., Bosmans, E., & Triest, L. (2003, January 30). *Iterative feedback tuning of PID parameters: Comparison with classical tuning rules*. Control Engineering Practice. Retrieved from https://www.sciencedirect.com/science/article/pii/S0967066102003039

[4] - Whitmore, S. A. (2009, May 9). *Closed-form integrator for the quaternion (Euler angle) kinematics equations*. NASA Technical Reports Server. Retrieved from https://ntrs.nasa.gov/citations/20080004113

[5] - Zhang, F. (1998, May 19). *Quaternions and matrices of quaternions*. Linear Algebra and its Applications. Retrieved from https://www.sciencedirect.com/science/article/pii/0024379595005439

[6] - Zhao, F., & van Wachem, B. (2013, July 13). *A novel quaternion integration approach for describing the behaviour of non-spherical particles*. ResearchGate. Retrieved from https://www.researchgate.net/publication257451700_A_novel_Quaternion_integration_approach_for_describing_the_ behaviour_ of_non-spherical_particles

[7] - Bacon, B., Schuet, S., Lombaerts, T., & Belcastro, C. (2012, September 6). *Quaternion-based control architecture for determining controllability/maneuverability limits*. Aerospace Research Central. Retrieved February 1, 2022, from https://arc.aiaa.org/doi/10.2514/6.2012-5028

[8] - Wang, X., Wang, X., & Yu, C. (2011). Unit-Dual-Quaternion-Based PID Control Scheme for Rigid-Body Transformation. IFAC Proceedings Volumes, 44, 9296-9301. Retrieved from https://

www.semanticscholar.org/paper/Unit-Dual-Quaternion-Based-PID-Control-Scheme-for-Wang-Wang/5cd7f29f4c54b768ded1e5c6268a3332da1983ac

[9] - Balya, Darohini and M. F., Abas and Pebrianti, Dwi and Hamzah, Ahmad and M. H., Ariff and M. R., Arshad (2019) Optimization of quaternion based on hybrid PID and Pw control. In: The 5th International Conference on Electrical, Control and Computer Engineering (INECCE2019). Springer. Retrieved from https://books.google.es/books?id=gp_YDwAAQBAJ&lpg=PA152&lr&pg=PR2#v=onepage&q&f=false

[10] - Guentert, P. H. (2017, July 17). *A variable pitch quadrotor with quaternion based attitude controller*. OhioLINK ETD. Retrieved from https://etd.ohiolink.edu/apexprod/rws_olink/r/1501/10?clear=10&p10_accession_num=ucin1504882010631186

[11] - Castillo Frasquet, A. (2016, July 7). Development of a quaternion-based quadrotor control system based on disturbance rejection. Valencia; Universitat Politècnica de València. Retrieved from https://riunet.upv.es/bitstream/handle/10251/73016/29205468e_TFM_146789020271946170382093958781 73.pdf?sequence=3

[12] - Solà, Joan. (2015). Quaternion kinematics for the error-state KF. ResearchGate. Retrieved from https://www.researchgate.net/publication/278619675_Quaternion_kinematics_for_the_error-state_KF