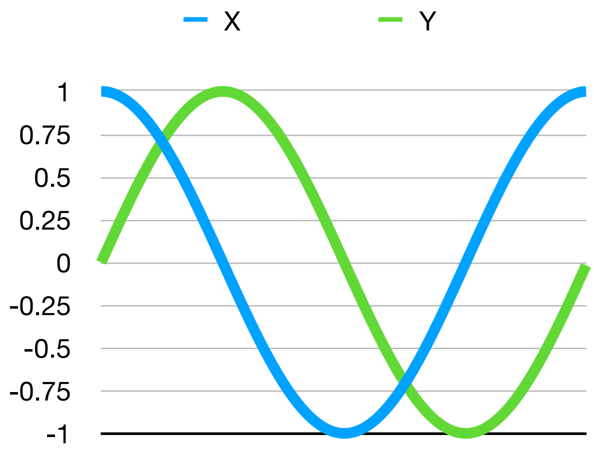


Vertical axis

$$x = cos(\theta)$$
$$y = sin(\theta)$$

Horizantal axis

1 *degree per tick*

Short Study on Quaternions

The study is focused on the math and properties of quaternions removing any obscurity it has received as being a difficult subject to grasp. The goal is to uncover the correctness in its use and its implementation when applied to rotations. The author admits the difficulty in its correct use as well. The free and open papers used for the study are few in number when discussing quaternions. The paper used in this study titled, *Quaternions, Interpolation and Animation* by Erik B. Dam, Martin Koch, and Martin Lillholm under the Technical Report DIKU-TR-98/5 dated July 17, 1998. In section 3.3, it covers quaternions and its properties. Many textbooks and other papers sadly covered little information on the properties. The authors may not fully grasp as well although speculatively, it might be why these papers were incompletely written without understanding the math and its properties. The selected paper, therefore, was chosen for the important information, the properties of quaternions. Furthermore this study also includes vectors and matrices in relation to quaternions and more so in relation to the active project for the development of the Algebra Library named Numerics kept at MageMCU Github.

Property-1: The *definition* of the quaternion. *The set of quaternions is denoted by Hamilton **H***.

Quaternions can be viewed as a vector with four-components.

$$\mathbf{q} = (w, x, y, z)$$

It can be visualized as a complex number in three dimensions where the w is the real part and the $x\mathbf{i}, y\mathbf{j}, z\mathbf{k}$ are the imaginary parts. The x, y, z are the scalar values and the $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are the orthonormal basis of the imaginary axes. *As a fourth dimensional concept, could not clarify the geometric representation*. The quaternion as seen as a complex number is not discussed any further than this representation .

$$\mathbf{q} = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$$

The quaternion components (w, x, y, z) can be further defined to be a scalar and a 3-component vector. Here both the quaternion and the vector uses bold type to distinguish from the scalar w without bold type.

$$\mathbf{q} = w + \mathbf{v}$$

The scalar is the ***w**-component* and the vector $\mathbf{v}$ contains the 3-components (***x, y, z***).

Property-2: The *norm-squared* of the quaternion. The vector dot product is $\mathbf{v} \bullet \mathbf{v} = \sqrt{x^2 + y^2 + z^2}$.

$$||q||^2 = w^2 + x^2 + y^2 + z^z = w^2 + \mathbf{v} \bullet \mathbf{v}$$

Property-2-1: The *norm* of the quaternion. *Could not clarify a quaternion length or magnitude.*

$$||q|| = \sqrt{q \bullet q} = \sqrt{w^2 + x^2 + y^2 + z^2}$$

Property-2-2: The *unit-length* (or *unit-quaternion*) of the quaternion. Are the unit-length of the quaternion and the norm of the quaternion a contradiction.

$$\hat{\mathbf{q}} = \frac{\mathbf{q}}{||q||} = \frac{w}{||q||} + \frac{\mathbf{v}}{||v||} = \left(\frac{w}{||q||}, \frac{x}{||q||}, \frac{y}{||q||}, \frac{z}{||q||} \right)$$

When the scalar $w \neq 0$, the vector $\mathbf{v}$ is *not* a unit-vector, otherwise when $w = 0$, then the vector $\mathbf{v}$ is a unit-vector if and only if $\mathbf{q}$ is an unit-quaternion. The unit-length notation for quaternions $\hat{\mathbf{q}}$ (q-hat) as shown here is similar to that of the unit-length for vectors $\hat{\mathbf{v}}$ (v-hat).

Property-3: The *identity* $\boldsymbol{I}$ of the quaternion.

$$\mathbf{q}\boldsymbol{I} = \boldsymbol{I}\mathbf{q} = \mathbf{q}$$

Property-3-1: Without defining the quaternion multiplication, the *conjugate* $\mathbf{q}^*$ of the quaternion

$$\mathbf{q}^* = (w, \mathbf{v})^* = (w, -\mathbf{v}) = (w, -x, -y, -z)$$

where $(q^*)^* = q$, $(pq)^* = p^*q^*$, $(p + q)^* = p^* + q^*$, and $qq^* = q^*q$.

Property-3-2: The *mapping* of the quaternion is a *norm* as defined in Property-2.

$$||q|| = \sqrt{qq^*}$$

The identity $I = 1 + \mathbf{0}$ is the unique neutral element under quaternion multiplication where I is the only neutral element in ***H.***

Property-3-3: The *inverse* of the quaternion.

$$\mathbf{q^{-1}} = \frac{\mathbf{q}*}{||q||^2}$$

Let $\mathbf{p} = \frac{\mathbf{q}*}{||q||^2}$, then $qp = q\frac{q*}{||q||^2} = \frac{qq*}{||q||^2} = \frac{||q||^2}{||q||^2} = 1 = I$. The unit-quaternion as defined in Property-2 is a subgroup of the group ***H***.

$$||q^{-1}|| = ||q*|| = ||q|| = 1$$

Property-4: The unit-quaternion can be created with an arbitrary *rotational-axis* of a unit-vector $\hat{\mathbf{v}}$ and with an arbitrary *angle* θ.

$$\hat{\mathbf{q}} = w + \hat{\mathbf{v}} = cos\left(\frac{\theta}{2}\right) + sin\left(\frac{\theta}{2}\right)\hat{\mathbf{v}}$$

Property-5: Quaternion multiplication. This involves the vector dot product and the vector cross product. The resultant is a rotational unit-quaternion. The proof is not given here.

$$\hat{\mathbf{p}}\hat{\mathbf{q}} = (p_w q_w - \hat{\mathbf{v}}_\mathbf{p} \bullet \hat{\mathbf{v}}_\mathbf{q}) + (p_w\hat{\mathbf{v}}_\mathbf{q} + q_w\hat{\mathbf{v}}_\mathbf{p} + \hat{\mathbf{v}}_\mathbf{p} \times \hat{\mathbf{v}}_\mathbf{q}) = \hat{\mathbf{r}}$$

Programmer's Note

As a mental frame of reference, the ***first plot*** above the title of this article is a spreadsheet line chart of a ***simple trigonometric cosine-sine plot*** programmed iteratively to the csv-file of the data giving about 720 data points (x, y). The quaternions have a similar analog in Property-4. ***Note the cosine-sine relationships between the plot and the equation in property-4.***

The following plots are similar for the quaternion multiplication, property-5. The equation used to plot the quaternion multiplication is $\hat{\mathbf{q}}^{'} = \hat{\mathbf{q}}\hat{\mathbf{c}}$ where $\hat{\mathbf{q}}$ is the unit-quaternion variable initially set to the identity I as $\hat{\mathbf{q}} = I$. Each iteration or after each quaternion multiplication, $\hat{\mathbf{q}}$ is re-assigned as $\hat{\mathbf{q}} = \hat{\mathbf{q}}^{'}$ which is actually the same variable. The unit-quaternion $\hat{\mathbf{c}}$ is a rotational constant assigned a ***rotational axis of (1, 1, 1***) then again with ***(1, 2, 3***) for two separate plots. The angle for each plot remains at a constant of 1-degree which in radian measure is 0.01745329….

In the quaternion class of the following *C-Pseudo-Code (actually C#) where* it lists the methods discussed for the quaternion class. The code snippets are part of a private library but is converted to C++ Library called Numerics at MageMCU at Github.

The Constructor:

```
public Quaternion4f(Vector3f axisVector, float angleRadians)
{
    // Private
    _size = 4;
    _tuples = new float[4];
    // NOT - Convert Degrees to Radians
    // WAS: float halfRadian = (angleDegrees * Mathf.Deg2Rad) / 2.0f;
    float halfRadian = angleRadians / 2.0f;
    float s = Mathf.Sin(halfRadian);
    Vector3f axisHat = axisVector.Normalize();

    _tuples[0] = Mathf.Cos(halfRadian);
    _tuples[1] = axisHat.x * s;
    _tuples[2] = axisHat.y * s;
    _tuples[3] = axisHat.z * s;
}
```

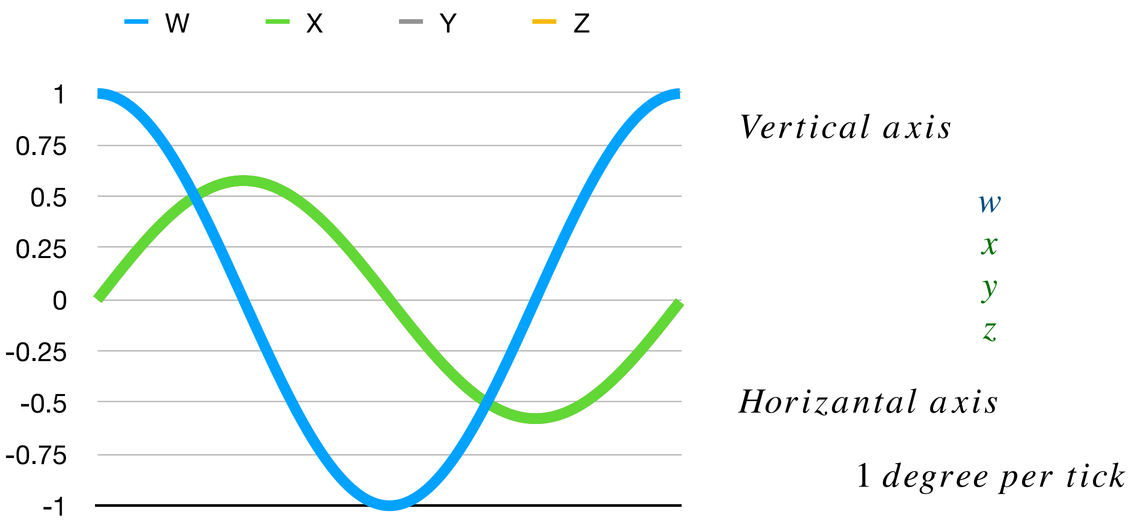
The Quaternion Multiplication:

```
public static Quaternion4f operator *(Quaternion4f q, Quaternion4f c)
{
    Vector3f qV = q.ToVector3f;
    Vector3f cV = c.ToVector3f;
    // multiplication (q.w * c.w) vector dot product (qV * cV)
    float qW = (q.w * c.w) - (qV * cV);
    // scalar vector multiplication (c.w * qV) & (pV * q.w) and cross product (pV ^ qV)
    Vector3f rV = (c.w * qV) + (cV * q.w) + (qV ^ cV);
    // Must normalize quaterion afterwards otherwise the components
    // converges to zero due to the floating point rounding errors
    // that diviates a unit-quaternion from its norm.
    return new Quaternion4f(qW, rV).UnitQuaternion();
}
```

The Test:

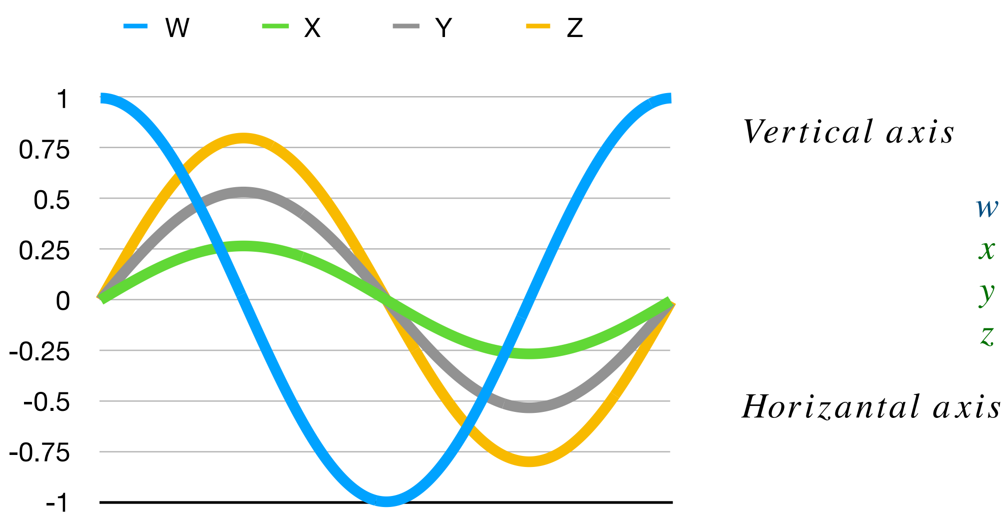
```
public class QuaternionTest02: MonoBehaviour
{
    // Quaternion Declarations
    Quaternion4f c;
    Quaternion4f q;

    float radian;
    Vector3f axis;

    // Initialization
    void Start()
    {
        // 1-degree to Radian Measure
        radian = 1.0f * Mathf.Deg2Rad;

        // Axis Vector
        axis = new Vector3f(1, 2, 3);

        // Quaternion Assignments
        c = new Quaternion4f(axis, radian);
        // Assigned the identity quaternion using
        // the constructor and its arguments.
        q = new Quaternion4f(axis, 0f);
    }

    // Game Loop
    void Update()
    {
        // Prints q(w, x, y, x) used for csv-file and plotting
        Debug.Log(q.ToString());

        // Quaternion Multiplication
        q = q*c;
    }
}
```

In this experiment, the unit-quaternion $\hat{\mathbf{c}}$ is set as the rotational constant assigned a ***rotational axis of (1, 1, 1***) and the angle is a constant of 1-degree which is converted to radian measure of 0.01745329….



The curves for Y and Z sit behind the green curve for X where W is the blue curve.

In this next experiment, the unit-quaternion $\hat{\mathbf{c}}$ is set as the rotational constant assigned a ***rotational axis of (1, 2, 3***) and the angle is a constant of 1-degree which is converted to radian measure of 0.01745329….



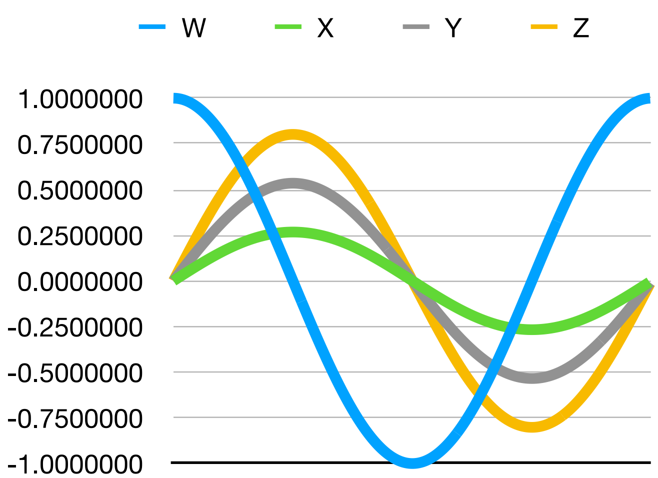
I had fun designing this experiment… I leave it to the reader to interpret all 3 plots especially their similarities and differences. Do the experiment yourself to verify whether you can reproduce the experiment. The Numerics Library can be located at MageMCU at Github. The csv-files can be located there as well. Match the curves along with the data as well. Enjoy!

Simulations

The experimental simulations were initially done using the Unity Game Engine. The reason for the simulations was to elucidate to me personally how quaternions would behave under different conditions and to demonstrate how to use them correctly. Computer simulations versus visually comprehending the math-proofs, for example, when plotting the data for the first plot, the angle θ whose data-type is an integer initially set to zero was incremented 1-degree giving about 720 data points. Simple algebraic trigonometry was enough. When using quaternions instead of addition, each iteration was multiplied by a constant to a quaternion variable also giving about 720 data points. The fact is not to add quaternions but to multiply quaternions producing similar results as in the trigonometric plot by simply adding angles. This simple difference was elusive. All I get from math-proofs, well, maybe what I cannot see. There will be further study when time permits. The quaternion class has the basic foundation to further develop other methods.

Arduino Experiment

Once satisfied with the simulations, the Numerics program for the Arduino Uno using VS-Code and PlatformIO IDE was tested with the updated quaternion class, a converted C# to C++ code. Arduino produced the exact results as the second quaternion experiment above…



Vertical axis

w
x
y
z

Horizantal axis

1 *degree per tick*

There was an issue when using Serial.print() and its counterpart Serial.println(). It would only print two decimal places. In the TESTS folder, is the Common.h file where it has the function *printQuaternion()* with a single parameter. There is discusses Arduino's float data-type and String object-type along with its sources. The code statement, *Serial.print(String(q.Element(i), 7)),* will print 7 decimal places as seen in the vertical axis of the chart. Review the code for the details and again, try to reproduce the data yourself.

Final Note:

Matrix multiplications and additions for a MCU like the Atmega328P is a little bit too much especially the memory where quaternions are a little faster than matrix math. Yet even quaternions can be studied further to reduce these calculations. For example, in some cases using the directional unit-vector is great for robotic rotations and the number of calculations is surprisingly at a bare-minimum. This technique will be discussed in up-coming articles.

Other Quaternion Properties
Quaternion multiplication is non-communicative $\hat{\mathbf{p}}\hat{\mathbf{q}} \neq \hat{\mathbf{q}}\hat{\mathbf{p}}$.
Vector addition was used instead of quaternion addition not incorporated.

Carpenter Software
Article 1005-Math-Quaternions-006 20221207.
Revision 1005-Math-Quaternions-006 20221208.
by Jesse Carpenter