# Rotaciones en MatLab mediante Matrices de Rotación y Cuaterniones

Carlos Alberto Edo Solera

ÍNDICE:

1.- Rotaciones mediante cuaterniones

2.- Álgebra de cuaterniones.

3.- Cuaterniones con MatLab.

## 1.- Rotaciones mediante cuaterniones.

Los cuaterniones fueron descubiertos en 1843 por William Rowan Hamilton. Son una extensión de los números complejos, en los que se ha añadido dos dimensiones adicionales, teniendo un total de 4. Una real y 3 imaginarias.

Teniendo en cuenta que en 4 dimensiones hay 3 soluciones diferentes a la raíz de (-1), cada una de las dimensiones imaginarias tiene un valor múltiplo de la raíz de (-1), conocidos como *i, j, k*.

Cada una de estas unidades imaginarias representará un giro de 180º, siendo su cuadrado un giro de 360º, de forma que tanto "+1", como "-1", representarán el mismo giro, aunque por caminos diferentes.

El cuaternión *i* representa una rotación de 180º alrededor del eje X, el *j* sobre el eje Y, y el *k* sobre el Z. De esta manera, $i*i=-1$, representa una rotación de 360º alrededor del eje X.

Los cuaterniones pueden representar rotaciones 3D, escalados y reflexiones, sin embargo, no pueden representar traslaciones.

Para poder representar una rotación mediante cuaterniones hemos de tener el punto a rotar en forma de cuaternión, lo que se consigue generando un cuaternión con la parte imaginaria a cero, y como valores de *i, j, k* las coordenadas del vector. Así el vector v={1,2,3} quedaría como el cuaternión $q=0+i+2j+3k$.

Para rotar un punto mediante un cuaternión la operación que debemos hacer es premultiplicar el punto (en forma también de cuaternión) por el cuaternión de la rotación, y postmultiplicarlo por el conjugado del cuaternión de la rotación.

Prot = q * P * conj(q)

El conjugado de un cuaternión se obtiene cambiando el signo de todos los elementos de su parte imaginaria.

Si queremos hacer varias rotaciones sucesivas, hemos de ir premultiplicando cada cuaternión por el correspondiente a la siguiente rotación. Así, si q1 y q2 representan 2 rotaciones sucesivas, q3 = q2 * q1. Para rotar usando q3 hemos de tener en cuenta una propiedad del conjugado de cuaterniones, que es la siguiente: Conj(q2 * q1) = Conj(q1) * Conj(q2). Y tener siempre presente que la multiplicación de cuaterniones, por lo general, **no** es conmutativa. Así, Prot = q3 * P * conj(q3) = q2 * q1 * P * conj(q1) * conj(q2).

## 2.- Álgebra de cuaterniones.

Adición de cuaterniones:

Para sumar dos cuaterniones hemos de sumarlos elemento a elemento, de esta forma: a + *i* b + *j* c + *k* d) + (e + *i* f + *j* g + *k* h) = (a+e) + *i* ( b+f) + *j* ( c+g) + *k* ( d+h).

Resta de cuaterniones:

La resta de cuaterniones se realiza de formaanáloga a la suma:

(a + i b + j c + k d)-(e + i f + j g + k h) = (a-e) + i ( b-f) + j ( c-g) + k ( d-h)

Multiplicación de cuaterniones:

Para multiplicar cuaterniones hemos de tener en cuenta el álgebra que hay detrás de las 3 unidades imaginarias, usando la fórmula de Hamilton:

$i2 = j2 = k2 = ijk = -1$

De donde se extraerán el resto de identidades para poder operar la multiplicación, que son:

$i * i = j * j = k * k = -1$

$i * j = k,$

$j * i = -k$

$j * k = i,$

$k * j = -i$

$k * i = j,$

$i * k = -j$

Como puede verse, resulta evidente que la multiplicación no deba ser conmutativa.

*Código que multiplica 2 cuaterniones:*

```
function [q]=multiplica_cuaternion(q1,q2)

x =  q1(2) * q2(1) + q1(3) * q2(4) - q1(4) * q2(3) + q1(1) * q2(2);
y = -q1(2) * q2(4) + q1(3) * q2(1) + q1(4) * q2(2) + q1(1) * q2(3);
z =  q1(2) * q2(3) - q1(3) * q2(2) + q1(4) * q2(1) + q1(1) * q2(4);
w = -q1(2) * q2(2) - q1(3) * q2(3) - q1(4) * q2(4) + q1(1) * q2(1);

q=[w;x;y;z];
```

División de cuaterniones:

La división se realiza mediante el conjugado, aprovechando el hecho de que, cuando un cuaternión está *normalizado*, su conjugado es igual a su inversa, con lo cual podemos dividir multiplicando por el conjugado.

Normalización de un cuaternión:

Se considera la *Norma* de un cuaternión, a la raíz cuadrada de la suma de sus cuatro componentes al cuadrado. Cuando ésta es 1, se dice que el cuaternión está normalizado. Si no lo es, el cuaternión puede normalizarse dividiendo cada una de sus 4 componentes por dicha norma.

## 3.- Cuaterniones con Matlab:

Para poder trabajar en MatLab con los cuaterniones se han diseñado diferentes algoritmos que permiten intercambiar las representaciones de una rotación mediante *Matrices de rotación, Cuaterniones y Ángulos de Euler*.

*p.Ej. – Paso de Matriz de rotación a cuaternión:*

```
%Esta funcion transforma una matriz de rotacion en un cuaternion
%que realiza la misma rotacion.
function [q]=matriz2cuat(matriz)
%matriz - es la matriz de rotacion.
%q - es el cuaternion.

T = matriz(1,1)+matriz(2,2)+matriz(3,3)+ 1;
if T > 0
    S = 0.5 / sqrt(T);
    qw = 0.25 / S;
    qx = ( matriz(3,2) - matriz(2,3) ) * S;
    qy = ( matriz(1,3) - matriz(3,1) ) * S;
    qz = ( matriz(2,1) - matriz(1,2) ) * S;
else if T <= 0
        if ((matriz(1,1) > matriz(2,2))&(matriz(1,1) > matriz(3,3)))
            S = sqrt( 1.0 + matriz(1,1) - matriz(2,2)- matriz(3,3))*2;
            qw = (matriz(2,3) - matriz(3,2)) / S;
            qx = 0.25 * S;
            qy = (matriz(1,2) + matriz(2,1)) / S;
            qz = (m02 + m20) / S;
        else if (matriz(1,1) > matriz(2,2))
                S = sqrt( 1.0 + matriz(2,2) - matriz(1,1) -
matriz(3,3) ) * 2;
                qw = (matriz(1,3) - matriz(3,1)) / S;
                qx = (matriz(1,2) + matriz(2,1)) / S;
                qy = 0.25 * S;
                qz = (matriz(2,3) + matriz(3,2)) / S;
            else
                S = sqrt( 1.0 + matriz(3,3) - matriz(1,1) -
matriz(2,2) ) * 2;
                qw = (matriz(1,2) - matriz(2,1)) / S;
                qx = (matriz(1,3) + matriz(3,1)) / S;
                qy = (matriz(2,3) + matriz(3,2)) / S;
                qz = 0.25 * S;
```

```
            end;
        end;
    end;
end;

q=[qw;qx;qy;qz];
```

Una vez introducida una rotación 3D mediante cualquiera de los 3 métodos, se obtiene la representación de esa rotación en los otros 2.

Se ha implementado una interfaz gráfica MatLab (GUIDE) que agrupa dichas funcionalidades, además de disponer de una función para ver el resultado de la rotación introducida, aplicada sobre un objeto dibujado en unos ejes.

Se puede elegir entre usar el algoritmo de rotación por matrices de rotación o usar el algoritmo con cuaterniones, pudiéndose ver que ambos llegan a la misma figura.

Para el desarrollo de estos algoritmos se ha tenido que implementar a su vez un abanico de subfunciones como son la multiplicación de cuaterniones, cambio de representación mediante a vector de un punto a representación mediante cuaternión, y viceversa, normalización de un cuaternión, conjugado, ... Y otras funciones que han sido adaptadas de las funciones desarrolladas durante las prácticas de la asignatura, como extraer los puntos inicial y final de una línea dado su manejador, aplicar una rotación a un objeto, aplicar una rotación a una línea,... siempre en versión para ser usadas con matrices de rotación y en versión para cuaterniones.