Universidad de La Laguna

Robótica Computacional

Cuaterniones

Javier Duque Melguizo
Miguel Angel Ordoñez Morales

Índice

Definición

Los cuaterniones se definen como **un sistema numérico que extienden a los números complejos** con el objetivo de utilizarlos en la mecánica para cálculos y representaciones del espacio $\Re^3$ o tridimensional, de forma análoga a como se pueden utilizar los números complejos para representar el plano $\Re^2$ o espacio bidimensional. Los cuaterniones, como extensión de los números complejos, poseen una parte imaginaria y otra real y vienen representados de la siguiente manera:

$$\underbrace{a +}_{\text{Parte Real}} \underbrace{b i + c j + d k}_{\text{x3 Parte Imaginaria}}$$

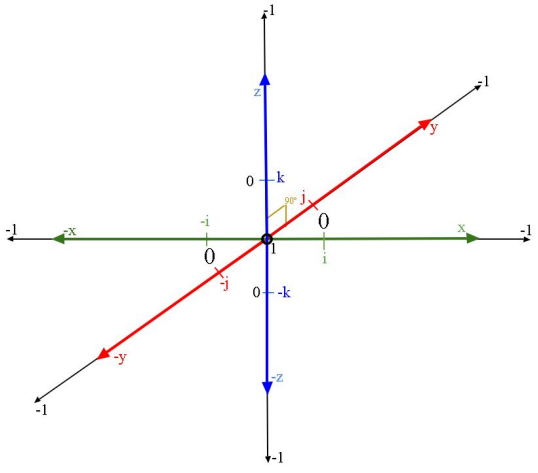
Donde:

$$i^2 = j^2 = k^2 = i \cdot j \cdot k = -1$$

$$\{a,\ b,\ c,\ d\} \in \Re$$

Conviene destacar que, originalmente , a las partes imaginarias, Hamilton, creador de los cuaterniones, los denomina **“parte vectorial”** y la parte real, **“parte escalar”**.

Representación Gráfica



Un sistema de coordenadas de 4 dimensiones representando en uno de 3 mediante **proyección estereográfica** del hipercubo de radio igual a $\sqrt{-1}$.

De forma que a cada parte imaginaria se le asigna un eje del sistema de coordenadas cartesianas espaciales (i al eje X, j al eje Y, k al eje Z) y a la parte real se le asigna la hipotética 4° dimensión (**W**), **que es representado sobre los otros 3.** de forma que el origen de coordenadas (0,0,0) se localiza en el punto w = 1 de esta 4° dimensión y W = -1 se localiza en un punto infinito donde los 3 ejes cartesianos en cualquiera de sus direcciones, positivas o negativas, convergen, es decir, tomando cualquier dirección desde el origen de coordenadas (0,0,0,1), todos los ejes de coordenadas cartesianas espaciales convergerán en el infinito en el punto donde W= -1.

Historia

Introducidos por William Rowan Hamilton en 1843. Aunque Carl Friedrich Gauss también los descubrió en 1819, su trabajo al respecto no se publicó hasta 1900, 45 años después de su fallecimiento.

Hamilton observó que los números complejos servían para representar puntos en el plano $\mathfrak{R}^2$, y de manera análoga, intento hacer lo mismo para puntos en el espacio $\mathfrak{R}^3$ añadiendo una parte imaginaría al número complejo:

$$a + b_i + c_j$$

Hamilton por muchos años intentó definir el álgebra de estas 3-tuplas , y aunque por muchos años sabía cómo sumarlos y restarlos, nunco logró resolver la problemática que resultaba de intentar multiplicarlos o dividirlos, cosa que se demostraría imposible bastantes años después, gracias a Ferdinand Georg Frobenius , en 1877, quien probó específicamente que el álgebra de división para ser finito y asociativo, sólo podía ejecutarse sobre secuencias o tuplas de números con dimensión N, donde N debe ser distinto a 3.

Por suerte para Hamilton, este no tuvo que esperar tanto y dió con la clave un Lunes de Octubre de 1843, en Dublín, de camino a la **R**eal **A**cademia **I**rlandesa para presidir la reunión del consejo. Mientras él paseaba con su mujer de camino a la RAI, empezó a dar vueltas a su problemática y dió con la solución, que consiste en añadir una parte imaginaria más y anotó en la piedra del puente de Brougham, por el que caminaba en ese instante, la fórmula que le dio la clave para crear lo que él, más tarde, denominaría, **“Quaternions”** :

$$i^2 = j^2 = k^2 = i \cdot j \cdot k = -1$$

Aunque Hamilton intentó promover el uso de sus cuaterniones y estos fueron usados en las famosas ecuaciones de Maxwell que describen el fenómeno electromagnético, en 1880, los cuaterniones se vieron desplazado por una herramienta que servía para lo mismo y que para la comunidad científica resultaba más intuitiva, **el cálculo vectorial**.

Importancia

Los cuaterniones no solo existen en el campo de las matemáticas puras. Estos han trascendido de forma que su importancia a día de hoy reside en su posibilidad de uso para facilitar o incluso resolver diversos problemas que de otra forma serían difíciles o imposibles de resolver. Por ejemplo, los cuaterniones son usados en:

En el campo de las matemáticas aplicadas son especial útiles para:

- Gráficas 3D por computadora, en donde son usados como el sistema de referencia por excelencia para representar rotaciones en el espacio. Aunque se habla cada vez más de reemplazar los cuaterniones por una representación del espacio más intuitiva y representable en el espacio (recordemos que los cuaterniones no lo son) llamada “Rotores”.
- Visión por computador (por ejemplo, los colores RGB de una imagen se representan como un cuaternión con su parte real o escalar igual al valor nulo).
- Textura cristalográfica.

En el campo de la física son muy usados para:

- Resolución de problemas de electromagnetismo (ecuaciones de Maxwell).
- Describir la teoría de la relatividad general planteada por Einstein.
- Resolución de problemas de mecánica cuántica (especialmente, para representación espines de electrones)

En el campo de la robótica son especialmente utilizados para simplificar y reducir, notablemente, el número de cálculos computacionales a realizar para resolver **los problemas de cinemática directa**, que, por lo normal, incluyen un gran número de multiplicaciones al tratarse de un problema de cálculo matricial, lo cual se explicará más adelante en el informe.

Álgebra

A continuación se explica el álgebra necesaria para el desarrollo de la cinemática directa.

A partir de una operación entre cuaterniones, la idea es obtener como resultado otro cuaternión, es decir, se define como una *"regla"* donde se debe operar entre elementos con la **misma estructura**. Sin embargo, en algunos tipos de números, esto se vuelve complejo o no se cumple.

En las matrices, la propiedad de **adición** cumple con esta *"regla"*, ya que se realiza componente a componente y a partir de matrices del mismo tamaño, y da como resultado otra matriz del mismo tamaño. Mientras que la propiedad de **producto** no cumple con la *"regla"*, ya que no se necesitan matrices que tengan la misma estructura, solo importa que coincidan en una dimensión, y da como resultado una matriz con las dimensiones que pueden variar.

$$A_{m\times n} + B_{m\times n} = C_{m\times n} \qquad A_{m\times n}.B_{n\times p} = C_{m\times p}$$

En la fórmula de adición se puede ver que las 3 matrices tienen las misma estructura (dimensiones), pero en el producto, se utilizan matrices de distinta dimensión, por lo que no se mantiene esa consistencia y no se cumple la "regla".

Para establecer el álgebra con los cuaterniones, se define su aritmética mediante la aritmética usual de los números complejos. Para definir las operaciones vamos a partir de dos cuaterniones, definidos como:

$$q_1 = a_1 + b_1 i + c_1 j + d_1 k$$

$$q_2 = a_2 + b_2 i + c_2 j + d_2 k$$

Conjugado

Es una operación que consiste en invertir todos los componentes imaginarios.

$$\sim q_1 = a_1 - b_1 i - c_1 j - d_1 k$$

Ejemplo # 1

$$q_1 = -3 + 6i + 7j - 8k$$
$$q_1 = -3 - 6i - 7j + 8k$$

Suma

Al igual que con las matrices, se realiza término a término. Esta operación es **conmutativa** y **asociativa**.

$$q_1 + q_2 = (a_1 + a_2) + (b_1 + b_2)\, i + (c_1 + c_2)\, j + (d_1 + d_2)\, k$$

Ejemplo # 2

$$q_1 = -3 + 6i + 7j - 8k$$
$$q_2 = 1 + 12i - 7j - 11k$$
$$q_1 + q_2 = (-3 + 1) + (6 + 12)\, i + (7 - 7)\, j + (-8 - 11)\, k$$
$$q_1 + q_2 = -2 + 18i - 19k$$

Producto

Esta operación se comporta distinta, ya que si se hace término a término, como en el caso anterior, se obtiene como resultado un escalar en vez de un cuaternión, esto es lo que se conoce como el **producto escalar**, lo cual no cumple con la *"regla"* mencionada al principio.

Para ello, se define otra manera de hacer el producto, llamado **producto vectorial**. Esta operación es **asociativa**, pero no conmutativa.

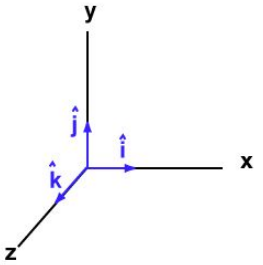
$$q_1 \cdot q_2 = \textit{Propiedad Distributiva}$$

Se utiliza la propiedad distributiva, generando 16 términos:

$$(a_1a_2) + (a_1b_2)\,i + (a_1c_2)\,j + (a_1d_2)\,k$$
$$(b_1a_2)\,i + (b_1b_2)\,i^2 + (b_1c_2)\,ij + (b_1d_2)\,ik$$
$$(c_1a_2)\,j + (c_1b_2)\,ji + (c_1c_2)\,j^2 + (c_1d_2)\,jk$$
$$(d_1a_2)\,k + (d_1b_2)\,ki + (d_1c_2)\,kj + (d_1d_2)\,k^2$$

En este punto, el problema es la multiplicación de partes imaginarias de distinto tipo, ya que las del mismo tipo quedan elevadas al cuadrado y son iguales a **-1**.

Para resolver este problema, se traslada a un sistemas de coordenadas. Cada parte imaginaria se asocia a un eje de coordenadas en el espacio de 3 dimensiones, como se ve en la imagen a continuación



En este punto, como el producto no es conmutativo, hay que tener **muy en cuenta** que el producto de **ij** es distinto a **ji**. Para saber su valor, se utiliza la técnica de la mano derecha, que cuenta con los siguientes pasos:

1. Se coloca la mano en dirección de la primera componente.
2. Se coloca la palma de la mano en sentido de la segunda componente.
3. Cerramos la mano en sentido desde la primera para la segunda componente.
4. Nos fijamos en el pulgar, si está en sentido de la componente que falta, el signo es positivo, de lo contrario es negativo.

Con este proceso, se puede construir la tabla con todas las operaciones:

	i	**j**	**k**
i	**i . i = -1**	**i . j = k**	**i . k = -j**
j	**j . i = -k**	**j . j = -1**	**j . k = i**
k	**k . i = j**	**k . j = -i**	**k . k = -1**

Una vez que se sustituye cada valor, quedan 4 términos que representan a cada componente:

$$(a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2)$$
$$(a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2)\,i$$
$$(a_1c_2 - b_1d_2 + c_1a_2 + d_1b_2)\,j$$
$$(a_1d_2 + b_1c_2 - c_1b_2 + d_1a_2)\,k$$

Ejemplo # 3

$$q_1 = -3 + 6i + 7j - 8k$$
$$q_2 = 1 + 12i - 7j - 11k$$

Las componentes son:

$$(-3 - 72 + 49 - 88)$$
$$(-36 + 6 - 77 - 56)\,i$$
$$(21 + 66 + 7 - 96)\,j$$
$$(33 - 42 - 84 - 8)\,k$$

Se hacen las sumas y restas:

$$q_1 \cdot q_2 = -144 - 163i - 2j - 101k$$

Rotación

Los cuaterniones permiten representar las orientaciones y las rotaciones de objetos en tres dimensiones. Sin embargo, no son intuitivos, lo que hace que las personas eviten utilizarlos. Se usan principalmente 2 fórmulas, las cuales son:

$$Q = cos\left(\frac{\theta}{2}\right) + sen\left(\frac{\theta}{2}\right) \cdot \vec{n}$$

$$O = Q \cdot (0, \vec{r}) \cdot \overline{Q_i}$$

La primera es la representación de un cuaternión, a partir de la rotación con θ grados, de un punto alrededor de un vector. Mientras, la segunda permite calcular las coordenadas de un punto, una vez rotado, a partir del producto de cuaterniones.

¿Por qué se usa $\theta/2$ en vez de θ? ¿Por qué se multiplica por una cuaternión y su conjugación? ¿Cómo se relaciona el coseno y el seno con las coordenadas del cuaternión?

Esas son algunas preguntas a las que nos enfrentamos la primera vez que vemos esas fórmulas. Pero, pensar en una rotación en 2 dimensiones, es trivial y un buen comienzo para entender el problema en 3 dimensiones. Dado un plano con un sistema de coordenadas cartesianas y un punto **P**, con un componente real *'x'* y uno imaginario *'y'*, se puede transformar al sistema polar, de la siguiente manera:

$$P(x_1, iy_1)$$
$$x = r \cdot \cos(\theta)$$
$$y = r \cdot i \cdot sen(\theta)$$

El símbolo **r** representa la distancia del origen de coordenadas al punto en línea recta. Pero, también podemos verlo como un vector con un extremo en el origen de coordenadas y otro en el punto **P**, con magnitud **r**. En este punto, como lo que importa es el ángulo y no el punto, asumimos que es un vector unitario, el cual tiene magnitud 1, para simplificar la fórmula.

$$x = \cos(\theta)$$
$$y = i \cdot sen(\theta)$$

Dado otro punto **Q**, lo que nos interesa es saber que la multiplicación de un número complejo por otro número complejo **unitario**, es equivalente a rotar en el espacio 2D.

$$P(\cos(\theta), i \cdot sen(\theta))$$
$$Q(x_2, iy_2)$$

Por la izquierda se multiplica una rotación con θ y por la derecha un punto en un plano de dos dimensiones.

$$(\cos(\theta) + i \cdot sen(\theta))(x_2 + iy_2) = (x_2 \cdot \cos(\theta) - y_2 \cdot \sin(\theta)) + i(x_2 \cdot sen(\theta) + y_2 \cdot \cos(\theta))$$

Para un ángulo de 90 grados quedaría:

$$(i)\,(x_2 + iy_2) = -y_2 + ix_2$$

Aunque los cuaterniones representan 4 dimensiones, se pueden hacer restringidos al subespacio 3D, al utilizar cuaterniones puros, que son aquellos que su componente real es cero.

Dado un cuaternión **S**, se aplica el mismo proceso anterior utilizando una rotación de 90 grados.

$$S = (w + ix + jy + kz)$$
$$(i)\,(w + ix + jy + kz) = -x + iw - jz + ky$$

Donde los componentes (w, x) se convierten en (-x, w), que significa una rotación de 90 grados. Simultáneamente, los componentes (y, z) se convierten en (-z, y), que es otra rotación de 90 grados. Por lo tanto, resulta que multiplicar por un cuaternión siempre gira en dos planos independientes a la vez.

Como la multiplicación de cuaterniones no es conmutativa, ¿Qué ocurre si se multiplica por la derecha?

$$(w + ix + jy + kz)\,(i) = -x + iw + jz - ky$$

Los componentes (w, x) se convierten en (-x, w) como antes, pero los componentes (y ,z) se convierten en (z ,- y), que es diferente, ya que aunque gira 90 grados, lo hace en una dirección opuesta.

El problema es que las rotaciones ocurren en dos planos, pero si se multiplica por ambos lados, las rotaciones opuestas se cancelan y la otra rotación se hará 2 veces.

$$(i)\,(w + ix + jy + kz)\,(i) = -w - ix + jy + kz$$

Los componentes (w, x) se convierten en (-w, -x), que significa una rotación de 180 grados, mientras los otros componentes no tienen cambios.

Como no se quiere rotar el plano que contiene la componente real, se invierte el cuaternión de la derecha, tal que:

$$(i)\,(w + ix + jy + kz)\,(\bar{i}) = (i)\,(w + ix + jy + kz)\,(-i) = w + ix - jy - kz$$

Ahora, la rotación del plano (w, x) se cancela y la del plano (y, z) se realiza dos veces. Entonces, es cuando se entiende que se debe utilizar el ángulo dividido entre 2, para compensarlo al rotarlo 2 veces.

Cinemática directa

El objetivo de la cinemática directa es encontrar los valores que deben tener las variables **cartesianas** a partir de las variables **articulares**. Para su resolución, se utiliza la matriz Denavit-Hartenberg y se calculan las matrices de transformación, para multiplicarlas entre sí y obtener las variables cartesianas.

Sin embargo, a medida que aumenta el número de articulaciones, aumenta el número de multiplicaciones de matrices, lo que hace que aumente el tiempo de cómputo considerablemente. Por ello, surge otro método, que utiliza **cuaterniones** y que busca optimizar el cálculo de la cinemática directa.

Lo primero que hacemos es definir la estructura de un cuaternión:

```
class Quaternion:
  def __init__(self, w, x, y, z):
    self.w = w
    self.x = x
    self.y = y
    self.z = z
```

Cada parámetro representa una componente del cuaternión, tal que:

$$q = w + xi + yj + zk$$

Luego, en esta clase se agrupan las operaciones explicadas en el álgebra, ya que son las que vamos a utilizar para resolver la cinemática directa, las cuales son:

- **Suma**

```
def __add__(self, other):
  return Quaternion(
    self.w + other.w,
    self.x + other.x,
    self.y + other.y,
    self.z + other.z
  )
```

- **Producto**

```
def __mul__(self, other):
  return Quaternion(
    self.w*other.w - self.x*other.x - self.y*other.y - self.z*other.z,
    self.w*other.x + self.x*other.w + self.y*other.z - self.z*other.y,
    self.w*other.y - self.x*other.z + self.y*other.w + self.z*other.x,
    self.w*other.z + self.x*other.y - self.y*other.x + self.z*other.w
  )
```

Nota: Se puede ver que cada componente es el resultado de la propiedad distributiva.

- **Conjugado**

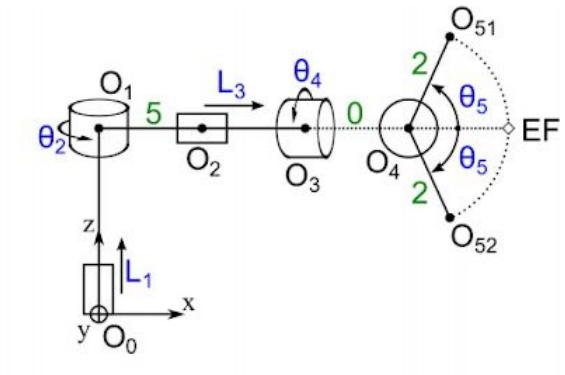
```
def __invert__(self):
  return Quaternion(
    self.w,
    -self.x,
    -self.y,
    -self.z
  )
```

Además de las operaciones, se implementa la representación de un cuaternión como la rotación de un punto alrededor de un vector, que corresponde a la siguiente función, que se explicó en el apartado anterior.

$$Q = cos\left(\frac{\theta}{2}\right) + sen\left(\frac{\theta}{2}\right) \cdot \vec{n}$$

```
@staticmethod
def rotationalVector(vector, theta):
  halfThetha = theta / 2
  sin_halfTheta = return Quaternion(
    math.cos(halfThetha),
    vector[0] * math.sin(halfThetha),
    vector[1] * math.sin(halfThetha),
    vector[2] * math.sin(halfThetha)
  )
```

Para aplicar la cinemática directa, se necesita de un manipulador.



Se puede ver que tiene 5 articulaciones, tanto de rotación como de desplazamiento, que dependen de valores variables, para calcular las coordenadas de los puntos se sigue un procedimiento que consiste en:

- **O0**

```
o0 = Quaternion(0, 0, 0, 0)
```

Para la primera articulación no hace falta hacer ningún cálculo. Lo más relevante es el eje de coordenadas definido en esa posición, ya que será utilizado más adelante.

- **O1**

```
l1 = float(input('Valor de longitud1 en metros: '))
t1 = 0

r1 = Quaternion(0, 0, 0, l1)

n1 = [0, 0, 1]
q1 = Quaternion.rotationalVector(n1, t1)

q1c = ~q1

o1 = q1 * r1 * q1c
```

Para asimilarlo con la teoría de rotación de cuaterniones antes explicada, separaremos en partes de código más pequeñas.

```
l1 = float(input('Valor de longitud1 en metros: '))
```

```
t1 = 0
```

Se solicita el valor del desplazamiento.

```
r1 = Quaternion(0, 0, 0, l1)
```

Es un vector puro, que indica en qué sentido se desplaza el siguiente punto, donde las últimas tres componentes representan los ejes **x**, **y**, **z** respectivamente. En este caso, como la dirección de crecimiento de la articulación coincide con el eje positivo **z** del sistema coordenadas, se añade la variable en la posición **z** del vector **r1** y con signo positivo.

```
n1 = [0, 0, 1]
q1 = Quaternion.rotationalVector(n1, t1)
```

Como es una articulación de desplazamiento, no hay rotación. Por lo tanto, no importa el valor de **n1**, mientras que el ángulo **t1** es cero, pero por convenio colocamos el vector unitario en la dirección que crece la articulación.

```
q1c = ~q1
```

Se calcula el cuaternión inverso.

```
o1 = q1 * r1 * q1c
```

Corresponde a la función:

$$O = Q \cdot (0, \vec{r}) \cdot \overline{Q_i}$$

Calcula la posición del punto **O1**, en función de la rotación de los 2 cuaterniones, para contrarrestar las rotaciones en otros planos.

- **O2**

```
t2 = float(input('Valor de theta2 en grados: '))
t2 = np.radians(t2)
a2 = 5

r2 = Quaternion(0, a2, 0, 0)

n2 = [0, 0, 1]
q2 = Quaternion.rotationalVector(n2, t2)

q2c = ~q2
```

```
o2 = (q1 * q2) * r2 * (q2c * q1c) + o1
```

Es similar al anterior, a excepción que es una articulación de rotación.

```
t2 = float(input('Valor de theta2 en grados: '))
t2 = np.radians(t2)
a2 = 5
```

Se solicita el valor de rotación, que se transforma en radianes, porque es la unidad que utiliza python, y se establece la longitud del brazo, ya que no varía.

```
r2 = Quaternion(0, a2, 0, 0)
```

Ahora la traslación ocurre en el eje x, y como coincide con el sistema de coordenadas, tiene signo positivo.

```
n2 = [0, 0, 1]
q2 = Quaternion.rotationalVector(n2, t2)
```

Ahora sí tiene utilidad esta parte del código, al ser una articulación de rotación. El eje de rotación es el eje **z** del sistema de coordenadas, que coincide con el sentido de la rotación que se quiere dar, según el sistema de la mano derecha, por lo tanto **n2** es de signo positivo.

```
q2c = ~q2
```

Se calcula el cuaternión inverso.

```
o2 = (q1 * q2) * r2 * (q2c * q1c) + o1
```

Hay más valores para multiplicar que con '**O1**', porque es un proceso acumulativo, donde las rotaciones y traslaciones, dependen de las articulaciones anteriores. Por lo tanto, la fórmula se modifica, tal que

$$O_n = \prod_{i=1}^{n} Q_i \cdot (0, \vec{r}) \cdot \prod_{i=n}^{1} \overline{Q_i} + O_{n-1}$$

- **O3 y O4**

El proceso es similar, **O3** como una articulación de desplazamiento y **O4** como una de rotación. Solo cambian los valores de los cuaterniones **'r'** y vectores **'n'**.

```
# ------ 3º articulación: Desplazamiento --------
r3 = Quaternion(0, l3, 0, 0)
```

```
n3 = [1, 0, 0]
# ------ 4º articulación: Rotación --------
r4 = Quaternion(0, a4, 0, 0)
n4 = [1, 0, 0]
```

- **O51, O52 y OEF**

```
t5 = float(input('Valor de theta5 en grados: '))
t5 = np.radians(t5)
a5 = 2

r5 = Quaternion(0, a5, 0, 0)

# 5A
n5A = [0, -1, 0]
q5A = Quaternion.rotationalVector(n5A, t5)

q5Ac = ~q5A

# 5B
n5B = [0, 1, 0]
q5B = Quaternion.rotationalVector(n5B, t5)

q5Bc = ~q5B

# EF
nEF = [0, 1, 0]
qEF = Quaternion.rotationalVector(nEF, 0)

qEFc = ~qEF
```

Para este conjunto de puntos, es un poco distinto, ya que se forman a partir de una bifurcación.

```
t5 = float(input('Valor de theta5 en grados: '))
t5 = np.radians(t5)
r5 = Quaternion(0, 2, 0, 0)
```

Se solicita la variable, que es común para los dos brazos, así como definir su longitud, la cual crece en el eje **x** del sistema de coordenadas.

```
n5A = [0, -1, 0]
n5B = [0, 1, 0]
```

Nuevamente, estos puntos continúan la extensión de la bifurcación previa definida:

```
n5A = [0, -1, 0]
q5A = Quaternion.rotationalVector(n5A, t5)

n5B = [0, 1, 0]
q5B = Quaternion.rotationalVector(n5B, t5)
```

La única diferencia es el signo de la rotación, esto se debe a que es una pinza. Para **n5A**, se asigna un valor negativo, porque el ángulo crece en sentido contrario al sistema de coordenadas, para **n5B** pasa lo contrario.

```
nEF = [0, 1, 0]
qEF = Quaternion.rotationalVector(nEF, 0)
```

Para **nEF** es solo una proyección, su valor es indistinto, ya que el ángulo es cero. Además, como anteriormente se dijo que la operación de rotación es acumulativa, por todas las anteriores, este último punto queda:

```
oEF = (q1 * q2 * q3 * q4 * qEF) * r5 * (qEFc * q4c * q3c * q2c * q1c)
+ o4
```

Comparación

En el siguiente link se puede observar 2 gráficas, donde se compara la cinemática directa con cuaterniones y matrices de Denavit-Hartenberg.

https://docs.google.com/spreadsheets/d/1eDFwmhCD3_inccH95Qt5EAPlYW5OA02TLOlamG25i0g/edit#gid=0

Las gráficas están en función del tiempo (eje y) y el tamaño de la entrada (eje x), que corresponde al número de articulaciones. Se representan 2 maneras de comparación, tales que:

- **Con optimización**

Cuando se ejecuta la última fórmula de la cinemática directa, la cual tiene 2 productores, se pueden reutilizar multiplicaciones. Por ejemplo:

$$O_1 = Q_1 \cdot \left(0, \vec{r_1}\right) \cdot \overline{Q_1} + O_0$$

$$O_2 = Q_1 \cdot Q_2 \cdot \left(0, \vec{r_2}\right) \cdot \overline{Q_2} \cdot \overline{Q_1} + O_1$$

$$O_3 = Q_1 \cdot Q_2 \cdot Q_3 \cdot \left(0, \vec{r_3}\right) \cdot \overline{Q_3} \cdot \overline{Q_2} \cdot \overline{Q_1} + O_2$$

Como se puede ver, a medida que el número de articulaciones aumentan, hay más productos y sobre todo **repetidos**. En este apartado, se optimizan los cálculos, ya que son definidos previamente para reutilizarlos, tal que:

$$Q_{12} = Q_1 \cdot Q_2$$

$$\overline{Q_{21}} = \overline{Q_2} \cdot \overline{Q_1}$$

$$Q_{13} = Q_{12} \cdot Q_3$$

$$\overline{Q_{31}} = \overline{Q_3} \cdot \overline{Q_{21}}$$

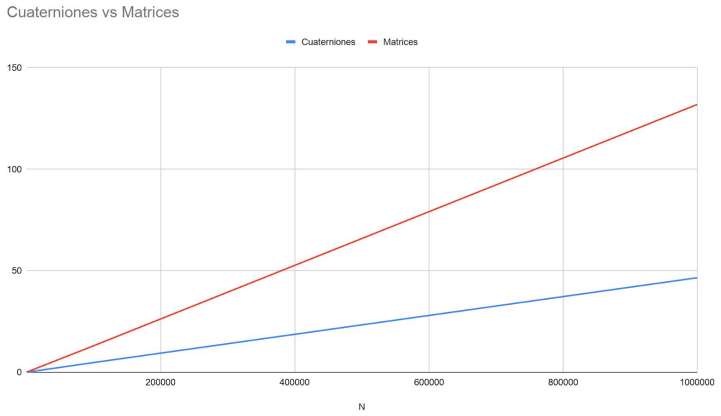
Luego, se reutilizan en la fórmula.

$$O_1 = Q_1 \cdot \left(0, \vec{r_1}\right) \cdot \overline{Q_1} + O_0$$

$$O_2 = Q_{12} \cdot \left(0, \vec{r_2}\right) \cdot \overline{Q_{21}} + O_1$$

$$O_3 = Q_{13} \cdot \left(0, \vec{r_3}\right) \cdot \overline{Q_{31}} + O_2$$

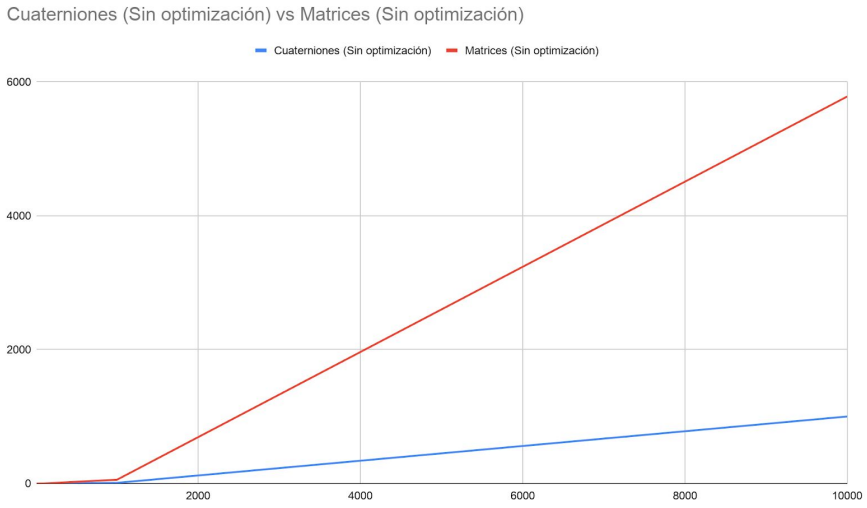
Es un ahorro significativo de operaciones, Además esta operación de optimización también se hace con las matrices para tener las mismas condiciones, simplificando el número de operaciones con matrices en su productorio particular. Los algoritmos que se obtienen son 2 algoritmos de complejidad **O(n)** .



En la gráfica se puede observar que los tiempos en los que el algoritmo que resuelve la cinemática directa con cuaterniones mejora significativamente a medida que crece el tamaño de la entrada respecto al algoritmo que lo resuelve con matrices de transformación D-H.

- **Sin optimización**

Se refiere a hacer la comparación si la optimización antes explicada. Tampoco se aplica a las matrices, para tener nuevamente las mismas condiciones. De esta forma obtenemos 2 algoritmos con complejidad **O(n^2)** .



En este caso, se ve que la diferencia se mantiene notoria entre el algoritmo que resuelve la cinemática directa con cuaterniones y la que lo resuelve con matrices.

Conclusiones

Los cuaterniones son más difíciles de entender y justificar matemáticamente, posiblemente dada su naturaleza en 4D.

Sin embargo, se da la casualidad que son más fáciles de utilizar para implementar un algoritmo de resolución de cinemática directa.

Por otro lado, las matrices son más intuitivas, su explicación y justificación matemática es más sencilla, pero a la hora de escalar el problema de cinemática directa se comienza a complicar y ser menos eficiente.

Por ello, durante el desarrollo del informe, hicimos una explicación intuitiva de los cuaterniones, donde se puede ver su increíble origen, álgebra, propiedades que permiten representarlos como una rotación y en consecuencia, resolver el modelo de la cinemática directa.

Cuando son optimizados logran mejorar el proceso de la cinemática directa con una diferencia evidente con respecto a las matrices utilizadas con el método Denavit-Hartenberg, reduciendo significativamente el tiempo de cómputo y logrando ser entre 3 y 5 veces más rápido.

Por lo tanto, es normal que sean utilizados en diversos campos el desarrollo diseño de videojuegos, movimiento de robots, entre otros, ya que permite hacer más operaciones en el mismo tiempo y movimientos más rápidos y en consecuencia dan margen a cómputos más precisos.

Repositorio

En siguiente link contiene el repositorio de GitHub, donde se encuentra el código fuente de cada ejercicio de cinemática directa y las comparaciones entre cuaterniones y matrices.

https://github.com/JDM-ULL-93/CD_Cuaternios

También en él se encuentra un README.md, donde explica las instrucciones para su funcionamiento.