Guía de Trabajo: Matrices de Rotación y Coordenadas Exponenciales en Python

Mecánica de Robots:
Herramientas matemáticas para la localización espacial de cuerpos rígidos
(Física II - Grado en Robótica, EPSE-USC Lugo)

11 de marzo de 2025

1. Introducción

Esta guía tiene como objetivo que el alumno comprenda, implemente y visualice de forma incremental el comportamiento de las **matrices de rotación** y las **coordenadas exponenciales** mediante el desarrollo de funciones en Python. Se estudiará la construcción de la matriz de rotación a partir del eje unitario $\hat{\omega}$ y el ángulo θ utilizando dos métodos:

1. **Definición Explícita:** Se construye la matriz de 9 componentes a partir de la notación

$$R(\hat{\omega},\theta) = \begin{pmatrix} c_\theta + \hat{\omega}_1^2(1-c_\theta) & \hat{\omega}_1\hat{\omega}_2(1-c_\theta) - \hat{\omega}_3\, s_\theta & \hat{\omega}_1\hat{\omega}_3(1-c_\theta) + \hat{\omega}_2\, s_\theta \\ \hat{\omega}_1\hat{\omega}_2(1-c_\theta) + \hat{\omega}_3\, s_\theta & c_\theta + \hat{\omega}_2^2(1-c_\theta) & \hat{\omega}_2\hat{\omega}_3(1-c_\theta) - \hat{\omega}_1\, s_\theta \\ \hat{\omega}_1\hat{\omega}_3(1-c_\theta) - \hat{\omega}_2\, s_\theta & \hat{\omega}_2\hat{\omega}_3(1-c_\theta) + \hat{\omega}_1\, s_\theta & c_\theta + \hat{\omega}_3^2(1-c_\theta) \end{pmatrix},$$

   donde $c_\theta = \cos\theta$ y $s_\theta = \sin\theta$.

2. **Fórmula de Rodrigues:** Que utiliza la matriz antisimétrica asociada al eje unitario de rotación $\hat{\omega}$, definida como

$$[\hat{\omega}] = \begin{pmatrix} 0 & -\hat{\omega}_3 & \hat{\omega}_2 \\ \hat{\omega}_3 & 0 & -\hat{\omega}_1 \\ -\hat{\omega}_2 & \hat{\omega}_1 & 0 \end{pmatrix},$$

   para expresar la matriz de rotación de la siguiente forma:

$$R(\hat{\omega},\theta) = I + s_\theta\,[\hat{\omega}] + (1-c_\theta)\,[\hat{\omega}]^2.$$

Se explorarán propiedades fundamentales de las rotaciones, tales como la no conmutatividad, la cancelación de subíndices consecutivos, la matriz inversa y la propiedad asociativa, mediante ejemplos visuales y ejercicios de desarrollo de código en Python. Adicionalmente, se incluirá la implementación de la **matriz logaritmo** de una matriz de rotación, que permite obtener el eje y el ángulo de la rotación de forma robusta. Se considerarán condiciones numéricas basadas en la traza de la matriz (por ejemplo, si $\mathrm{tr}(R) \geq 3$ se asume que R es la identidad, y si $\mathrm{tr}(R) \leq -1$ se maneja el caso especial de $\theta \approx \pi$) para evitar errores de redondeo e indeterminaciones.

2. Objetivos

1. Comprender la estructura y propiedades de las matrices de rotación (elementos de $SO(3)$).
2. Implementar en Python funciones que generen la matriz de rotación a partir de la **definición explícita** y luego mediante la **fórmula de Rodrigues**.
3. Visualizar la rotación de vectores y sistemas de coordenadas, comprobando propiedades fundamentales (no conmutatividad, cancelación de subíndices, inversa y asociatividad).
4. Implementar una función robusta para calcular el **logaritmo de la matriz de rotación**, gestionando condiciones especiales en función de la traza para evitar valores complejos o indeterminaciones.

3. Desarrollo en Python

A continuación se presentan ejemplos de código. Seguiremos este orden: primero implementar la matriz de rotación mediante la **definición explícita** y, posteriormente, la **fórmula de Rodrigues**.

3.1. Función para Imprimir Matrices con 3 Decimales

La siguiente función imprime matrices 3x3 redondeando cada componente a 3 decimales.

```
def imprimir_matriz(M, nombre="Matriz"):
    """
    Imprime la matriz M redondeada a 3 decimales con un encabezado.
    """
    M_redondeada = np.round(M, 3)
    print(f"{nombre} =")
    print(M_redondeada)
    print()
```

3.2. Matriz de Rotación por Definición Explícita

Utilizando la notación de los apuntes, implementamos la matriz de rotación a partir de la definición explícita.

```
def rotacion_explicita(hat_omega, theta):
    """
    Calcula la matriz de rotación R a partir de la definición explícita.

    Parámetros:
      hat_omega: vector unitario (eje de rotación) de 3 componentes.
      theta: ángulo de rotación en radianes.

    Retorna:
      R: matriz de rotación 3x3.

    La fórmula utilizada es:
      R = [ c_theta + hat_omega_i^2*(1-c_theta) , hat_omega_i*hat_omega_j
         *(1-c_theta) - hat_omega_k*s_theta, ... ]
      donde c_theta = cos(theta) y s_theta = sin(theta).
    """
    hat_omega = np.array(hat_omega, dtype=float).flatten()
    # Normalizar el vector, en caso de que no sea unitario
```

```
    hat_omega = hat_omega / np.linalg.norm(hat_omega)
    w1, w2, w3 = hat_omega
    c_theta = np.cos(theta)
    s_theta = np.sin(theta)
    one_minus_c = 1 - c_theta

    R = np.array([
        [c_theta + w1**2 * one_minus_c,      w1*w2*one_minus_c - w3*s_theta,
            w1*w3*one_minus_c + w2*s_theta],
        [w1*w2*one_minus_c + w3*s_theta,       c_theta + w2**2 * one_minus_c
           ,   w2*w3*one_minus_c - w1*s_theta],
        [w1*w3*one_minus_c - w2*s_theta,       w2*w3*one_minus_c + w1*
           s_theta,  c_theta + w3**2 * one_minus_c]
    ])
    return R

# Ejemplo de uso:
import numpy as np

hat_omega = [0, 0, 1]    # eje de rotación unitario: \hat{\omega} = (0, 0, 1)
theta = np.pi / 4        # 45 grados

R_explicita = rotacion_explicita(hat_omega, theta)
imprimir_matriz(R_explicita, "R (Definición Explícita)")
```

3.3. Matriz de Rotación por la Fórmula de Rodrigues

A continuación se implementa la función utilizando la fórmula de Rodrigues. Se reutiliza la función para generar la matriz antisimétrica.

```
def matriz_antisimetrica(omega):
    """
    Genera la matriz antisimétrica [omega] a partir de un vector de 3
       componentes.

    Parámetros:
      omega: vector de 3 componentes.

    Retorna:
      A: matriz 3x3 antisimétrica.
    """
    w = np.array(omega, dtype=float).flatten()
    A = np.array([
        [0,    -w[2],   w[1]],
        [w[2],   0,    -w[0]],
        [-w[1], w[0],    0]
    ])
    return A

def rodrigues(hat_omega, theta):
    """
    Calcula la matriz de rotación R utilizando la fórmula de Rodrigues.

    Parámetros:
      hat_omega: vector unitario (eje de rotación) de 3 componentes.
      theta: ángulo de rotación en radianes.

    Retorna:
      R: matriz de rotación 3x3.
```

```
    La fórmula es:
      R = I + s_theta * [hat_omega] + (1-c_theta)*([hat_omega])^2,
    donde s_theta = sin(theta) y c_theta = cos(theta).
    """
    hat_omega = np.array(hat_omega, dtype=float).flatten()
    hat_omega = hat_omega / np.linalg.norm(hat_omega)
    A = matriz_antisimetrica(hat_omega)
    c_theta = np.cos(theta)
    s_theta = np.sin(theta)
    R = np.eye(3) + s_theta * A + (1 - c_theta) * np.dot(A, A)
    return R

# Ejemplo de uso:
R_rodrigues = rodrigues(hat_omega, theta)
imprimir_matriz(R_rodrigues, "R (Fórmula de Rodrigues)")
```

3.4. Comparación de Implementaciones

Compara ambas implementaciones y confirma que los resultados son muy similares (dentro de la tolerancia numérica).

```
def comparar_rotaciones(hat_omega, theta):
    R1 = rotacion_explicita(hat_omega, theta)
    R2 = rodrigues(hat_omega, theta)
    diferencia = np.linalg.norm(R1 - R2)
    return R1, R2, diferencia

R1, R2, diff = comparar_rotaciones(hat_omega, theta)
imprimir_matriz(R1, "R (Definición Explícita)")
imprimir_matriz(R2, "R (Rodrigues)")
print("Diferencia entre métodos:", round(diff, 4))
```

3.5. Rotación de Vectores y Sistemas de Coordenadas

Este ejercicio permite visualizar:

- La rotación de un vector en 3D.
- La no conmutatividad aplicando dos rotaciones en órdenes inversos.
- La verificación de propiedades como la inversa (ya que $R^{-1} = R^T$) y la propiedad asociativa.

Rotación de un Vector y Visualización 3D

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def rotar_vector(v, hat_omega, theta, metodo='rodrigues'):
    """
    Rota el vector v según el eje (hat_omega) y ángulo theta, utilizando el
       método especificado.

    Parámetros:
      v: vector a rotar.
      hat_omega: vector unitario (eje de rotación).
```

```
      theta: ángulo de rotación en radianes.
      metodo: 'rodrigues' o 'explicita' para elegir el método.

    Retorna:
      v_rotado: vector rotado.
    """
    if metodo == 'rodrigues':
        R = rodrigues(hat_omega, theta)
    else:
        R = rotacion_explicita(hat_omega, theta)
    return np.dot(R, v)

# Definir vector, eje y ángulo
v = np.array([1, 0, 0])
theta_visual = np.pi/3  # 60 grados

v_rotado = rotar_vector(v, hat_omega, theta_visual)

# Visualización del vector original y rotado
fig = plt.figure(figsize=(12,5))
ax1 = fig.add_subplot(121, projection='3d')
ax1.quiver(0, 0, 0, v[0], v[1], v[2], color='r', label='Original')
ax1.quiver(0, 0, 0, v_rotado[0], v_rotado[1], v_rotado[2], color='b', label=
    'Rotado')
ax1.set_title('Rotación de un vector')
ax1.set_xlim([-1.5, 1.5]); ax1.set_ylim([-1.5, 1.5]); ax1.set_zlim([-1.5,
    1.5])
ax1.legend()
```

Demostración de la No Conmutatividad

Se definen dos rotaciones sobre ejes distintos y se aplica el producto en órdenes diferentes.

```
def demo_no_conmutatividad(hat_omega1, theta1, hat_omega2, theta2, v):
    """
    Demuestra la no conmutatividad de las rotaciones aplicándolas en
       distinto orden.

    Parámetros:
      hat_omega1, hat_omega2: ejes unitarios de rotación.
      theta1, theta2: ángulos de rotación en radianes.
      v: vector a rotar.

    Retorna:
      v_orden1: resultado de aplicar R1 seguido de R2.
      v_orden2: resultado de aplicar R2 seguido de R1.
      R1, R2: matrices de rotación calculadas.
    """
    R1 = rodrigues(hat_omega1, theta1)
    R2 = rodrigues(hat_omega2, theta2)

    v_orden1 = np.dot(np.dot(R1, R2), v)
    v_orden2 = np.dot(np.dot(R2, R1), v)
    return v_orden1, v_orden2, R1, R2

hat_omega1 = [0, 0, 1]
theta1 = np.pi/4   # 45 grados
hat_omega2 = [0, 1, 0]
theta2 = np.pi/6   # 30 grados
v_no_con = np.array([1, 0, 0])
```

```
v_ord1, v_ord2, R1, R2 = demo_no_conmutatividad(hat_omega1, theta1,
    hat_omega2, theta2, v_no_con)

# Visualización comparativa
ax2 = fig.add_subplot(122, projection='3d')
ax2.quiver(0, 0, 0, v_ord1[0], v_ord1[1], v_ord1[2], color='g', label='R1*R2
    *v')
ax2.quiver(0, 0, 0, v_ord2[0], v_ord2[1], v_ord2[2], color='m', label='R2*R1
    *v')
ax2.set_title('No conmutatividad de rotaciones')
ax2.set_xlim([-1.5, 1.5]); ax2.set_ylim([-1.5, 1.5]); ax2.set_zlim([-1.5,
    1.5])
ax2.legend()
plt.show()
```

Verificación de Propiedades Adicionales

Comprueba:

- **Propiedad de Cancelación de Subíndices:** Si R_{ij} representa la rotación del sistema i respecto a j, se verifica que $R_{ij}R_{jk} = R_{ik}$.
- **Matriz Inversa:** Verificar que $R^{-1} = R^T$.
- **Propiedad Asociativa:** Comprobar que $(R_1R_2)R_3 = R_1(R_2R_3)$.

Ejemplo para la verificación de la inversa:

```
def verificar_inversa(R):
    """
    Verifica que la inversa de la matriz de rotación R es igual a su
        traspuesta.
    """
    R_inv = np.linalg.inv(R)
    return np.allclose(R_inv, R.T, atol=1e-3)

print("Verificación inversa (R_rodrigues):", verificar_inversa(R_rodrigues))
```

Ejercicio Visual

Implementa una visualización incremental del sistema de ejes (triedro) transformado por rotaciones sucesivas. Por ejemplo, se dibuja el sistema inicial y se actualiza tras aplicar una serie de rotaciones (por ejemplo, 36 pasos de 10° cada uno), demostrando gráficamente la propiedad asociativa y el efecto acumulativo de las rotaciones.

```
def dibujar_triedro(ax, R, origen=np.array([0,0,0]), escala=1.0):
    """
    Dibuja un sistema de ejes (triedro) transformado por la matriz de
        rotación R.
    """
    colores = ['r', 'g', 'b']  # ejes X, Y, Z
    etiquetas = ['X', 'Y', 'Z']
    for i in range(3):
        eje = R[:, i] * escala
        ax.quiver(origen[0], origen[1], origen[2],
                  eje[0], eje[1], eje[2],
                  color=colores[i], label=etiquetas[i])
```

```
# Visualización incremental del triedro
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_title('Transformación Incremental del Triedro')
ax.set_xlim([-2,2]); ax.set_ylim([-2,2]); ax.set_zlim([-2,2])
plt.ion()  # Modo interactivo

R_acumulada = np.eye(3)
for ang in np.linspace(0, 2*np.pi, 36):
    R_inc = rodrigues([0,0,1], ang)
    R_acumulada = np.dot(R_inc, R_acumulada)
    ax.cla()
    dibujar_triedro(ax, R_acumulada, escala=1.5)
    ax.set_xlim([-2,2]); ax.set_ylim([-2,2]); ax.set_zlim([-2,2])
    plt.pause(0.1)
plt.ioff()
plt.show()
```

4. Matriz Logaritmo de Rotación

En esta sección se implementa la función `matrix_log` que calcula el logaritmo de una matriz de rotación $R \in SO(3)$ según la definición de los apuntes de clase. Se tienen en cuenta condiciones numéricas robustas para evitar valores complejos o indeterminaciones:

- Si $\text{tr}(R) \geq 3$, se considera que R es la identidad (por errores de redondeo) y se define $\theta = 0$.
- Si $\text{tr}(R) \leq -1$, se maneja el caso especial en el que $\theta \approx \pi$.

```
def matrix_log(R):
    """
    Calcula el logaritmo de la matriz de rotación R.

    Parámetros:
      R : matriz de rotación 3x3 (R  SO(3))

    Retorna:
      theta : ángulo de rotación en radianes.
      log_R : matriz antisimétrica [hat_omega]*theta que representa el
         logaritmo de R.

    Se aplican las siguientes condiciones para robustez numérica:
      - Si trace(R) >= 3, se asume R  I y theta = 0.
      - Si trace(R) <= -1, se maneja el caso especial con theta = .
    """
    tr = np.trace(R)

    # Caso: R casi es la identidad
    if tr >= 3.0:
        theta = 0.0
        return theta, np.zeros((3,3))

    # Caso especial: theta cercano a
    elif tr <= -1.0:
        theta = np.pi
        # Se selecciona el índice con mayor valor diagonal para evitar
           indeterminaciones
```

```
        diag = np.diag(R)
        idx = np.argmax(diag)
        hat_omega = np.zeros(3)
        hat_omega[idx] = np.sqrt((R[idx, idx] - R[(idx+1)%3, (idx+1)%3] +
                                  R[(idx+2)%3, (idx+2)%3] + 1) / 2)
        # Evitar división por cero
        if np.abs(hat_omega[idx]) < 1e-6:
            hat_omega[idx] = 1e-6
        hat_omega = hat_omega / np.linalg.norm(hat_omega)
        # Usar la fórmula general; aunque en este caso, s_theta  0,
        # la expresión (R - R.T)/(2*sin(theta)) es válida para theta = .
        log_R = (R - R.T) / (2 * np.sin(theta))
        return theta, log_R

    # Caso general
    else:
        theta = np.arccos((tr - 1) / 2)
        s_theta = np.sin(theta)
        # Filtrar posibles divisiones por cero
        if np.abs(s_theta) < 1e-6:
            s_theta = 1e-6
        log_R = (R - R.T) / (2 * s_theta)
        return theta, log_R

# Ejemplo de uso:
R = rotacion_explicita([0, 0, 1], np.pi/3)
theta, log_R = matrix_log(R)
print("Theta:", round(theta, 3))
print("Matriz logaritmo:")
print(np.round(log_R, 3))
```

Esta función calcula el logaritmo de R de forma robusta, verificando la traza para identificar los casos especiales y filtrando valores que pudieran generar problemas numéricos.