1. Introducción y Conceptos Básicos

Este proyecto implementa una simulación por computadora de un robot humanoide (modelo **Unitree G1**). El objetivo es permitir que el robot camine utilizando una "Inteligencia Artificial" (Red Neuronal) y, al mismo tiempo, ejecutar movimientos manuales específicos, como saludar con la mano.

¿Qué herramientas usamos?

- **Python:** El lenguaje de programación principal.
- **MuJoCo:** Un motor de física avanzado. Es el "mundo virtual" que calcula la gravedad, las colisiones y cómo se mueven las articulaciones.
- **PyTorch:** Una librería de Inteligencia Artificial. La usamos para cargar el "cerebro" del robot que sabe caminar.
- **GLFW:** Una librería que permite abrir una ventana en tu pantalla y detectar cuando presionas teclas (teclado).

Conceptos Clave (Glosario)

- **Joint (Articulación):** Punto donde se unen dos partes del robot y permite movimiento (ej. codo, rodilla).
- **DOF (Grados de Libertad):** Número de movimientos independientes. Este robot tiene 23 motores/articulaciones.
- **Torque:** La fuerza de rotación que aplican los motores para mover las extremidades.
- **Quaternion:** Una forma matemática compleja de representar la rotación de un objeto en 3D sin usar ángulos tradicionales.

---

2. Matemáticas Auxiliares: Entendiendo la Orientación

Antes de mover el robot, necesitamos saber hacia dónde mira. El simulador nos da la orientación en "Quaterniones" (4 números), pero los humanos entendemos mejor los "Ángulos de Euler" (Roll, Pitch, Yaw).

Función `quatToEuler`

Esta función es un traductor. Convierte el lenguaje del robot (Quaterniones) al lenguaje humano (Ángulos).

Python

```
def quatToEuler(quat):
    # ... cálculos trigonométricos ...
```

```
    # Retorna: [Roll, Pitch, Yaw]
    # Roll: Inclinación lateral
    # Pitch: Inclinación hacia adelante/atrás
    # Yaw: Giro o dirección (Norte, Sur, Este, Oeste)
    return eulerVec
```

- **Uso:** Se utiliza para saber si el robot se está cayendo o para orientar su caminata.

---

3. La Interfaz de Control: `_key_callback`

Esta sección conecta tu teclado con el cerebro del robot. Funciona como un "escucha" (listener) que espera a que presiones una tecla.

¿Cómo funciona?

Cada vez que presionas una tecla, esta función cambia una variable dentro del programa.

- **Locomoción (Caminar):** Teclas `W`, `A`, `S`, `D` modifican la velocidad y dirección.
- **Acciones Especiales:**
  - Tecla `G`: Activa la bandera `start_wave_sequence` (Iniciar saludo).
  - Tecla `R`: Activa la bandera `reset_to_default_pose` (Volver a postura normal).

Python

```
elif key == glfw.KEY_G:
    # Le dice al sistema: "El usuario quiere saludar"
    self.start_wave_sequence = True
    print("\n>>> INICIANDO SECUENCIA DE SALUDO <<<")
```

---

4. El Coreógrafo: Clase `WaveController`

Esta es la parte más "artística" del código. Como el robot no sabe saludar por defecto (solo sabe caminar), tuvimos que enseñarle paso a paso. Esta clase es un **Controlador de Estados Finitos**.

4.1. El Mapa del Cuerpo (`joint_map`)

Primero, le damos nombres humanos a los números de los motores. El motor 22 es el "codo derecho", el 19 es el "hombro derecho", etc.

4.2. Las Poses (Fotogramas Clave)

Definimos "fotos" de cómo debe verse el robot en distintos momentos del saludo. Es igual a cómo se hace una animación tradicional.

Python

```
self.poses = [
    # Paso 0: Quieto
    { ... },
    # Paso 2: Mano hacia la izquierda
    {
        'name': "Saludo izquierda",
        'joints': {
            'right_shoulder_roll': -1.25, # Levantar brazo lateralmente
            'right_elbow': -0.827,        # Doblar codo
            # ... otros motores ...
        }
    },
    # ...
]
```

4.3. Interpolación (Suavizado)

El robot no puede teletransportarse de la pose A a la pose B. Necesita moverse suavemente. Usamos una función matemática (Coseno) para calcular los puntos intermedios.

Python

```
def interpolate_position(self, q_init, q_target):
    # Calcula cuánto ha avanzado el tiempo (ratio de 0 a 1)
    # y devuelve la posición intermedia exacta.
    ratio = (1 - math.cos(math.pi * (self.trajectory_t / 
self.trajectory_T))) / 2
    return q_init + (q_target - q_init) * ratio
```

4.4. La Secuencia

El método `update` revisa el reloj constantemente. Si una pose terminó, carga la siguiente automáticamente:

1. Levantar brazo.
2. Mover a la izquierda.
3. Mover a la derecha.
4. Repetir.
5. Bajar brazo.

---

5. El Simulador Principal: Clase `HumanoidEnv`

Esta es la clase maestra que contiene todo: el robot, la física y la lógica principal.

5.1. Inicialización (`__init__`)

Aquí se configura el robot físico.

- `stiffness` **(Rigidez):** Qué tan duros son los "músculos" (motores).
- `damping` **(Amortiguación):** Qué tanto se resisten al movimiento rápido (para que no vibre).
- **Carga del Modelo:** Lee el archivo `.xml` que describe la forma 3D del robot.

5.2. El "Cerebro" de Caminata (`policy_jit`)

El código carga un archivo `.pt` (PyTorch). Este es una red neuronal ya entrenada.

- **Entrada:** Velocidad actual, posición de las piernas, comando del usuario (ir adelante).
- **Salida:** Posición deseada de las articulaciones de las piernas.

5.3. Observación (`get_observation`)

El robot necesita "sentir" su entorno antes de actuar. Recopila datos:

- `dof_pos` : ¿En qué ángulo están mis rodillas/codos?
- `dof_vel` : ¿Qué tan rápido se mueven?
- `ang_vel` : ¿Estoy girando?
- `commands` : ¿Qué quiere el humano que haga?

Estos datos se empaquetan y se envían a la Red Neuronal.

---

6. El Bucle Principal: Método `run()`

Aquí es donde ocurre la magia, repitiéndose miles de veces por segundo.

Paso a Paso del Bucle:

1. **Extracción de Datos:** Lee los sensores del simulador.
2. **Decisión (Locomoción):**
   - Toma los datos de los sensores.
   - Se los pasa a la Red Neuronal (`self.policy_jit`).
   - Recibe las acciones para las piernas (para mantener el equilibrio y caminar).
3. **Decisión (Brazos/Saludo):**
   - Revisa si el `WaveController` está activo.

- Si está saludando, ignora lo que diga la red neuronal para los brazos y usa las posiciones calculadas por el `WaveController`.
- Si no está saludando, usa una posición por defecto o movimiento aleatorio.

Python

```
    # Lógica simplificada
    if self.use_wave_control:
        # Obtener siguiente frame de la animación de saludo
        wave_target = self.wave_controller.update(...)
        target_final = wave_target
    else:
        # Usar caminata normal
        target_final = acciones_red_neuronal
```

- **Control PD (Proporcional-Derivativo):**
  El robot sabe *dónde quiere estar* (`pd_target`), pero necesita calcular *cuánta fuerza* (torque) aplicar para llegar ahí. Esta es la fórmula fundamental de la robótica de control:
  - Si estás lejos del objetivo (Posdeseada−PosactualPosdeseada−Posactual es grande), aplica mucha fuerza.
  - Si te mueves muy rápido (VelocidadVelocidad), frena un poco para no pasarte (Amortiguación).

Python

```
    torque = (pd_target - self.dof_pos) * self.stiffness - self.dof_vel * self.damping
```

- **Simulación:**
  Le dice a MuJoCo: "Aplica estos torques a los motores y avanza el tiempo 0.002 segundos".

Python

```
self.data.ctrl = torque
mujoco.mj_step(self.model, self.data)
```

2. **Renderizado:** Dibuja el robot en la pantalla.

---

7. Guía de Uso para el Usuario

Si ejecutas este código, verás una ventana con el robot G1.

1. **Para moverlo:**
   - Presiona **W** varias veces para aumentar la velocidad hacia adelante.

- Presiona **A** o **D** para girar.

2. **Para el Saludo (La función nueva):**
   - Presiona la tecla **G**.
   - Verás que el robot ignora el movimiento normal de sus brazos, levanta el brazo derecho y realiza un movimiento de saludo de lado a lado.
   - Al terminar, baja el brazo suavemente y vuelve al control normal.
3. **Emergencia:**
   - Si el robot queda en una pose extraña, presiona **R** para forzar los brazos a la posición inicial.

Resumen Técnico

Este script es un sistema híbrido. Combina:

1. **Aprendizaje por Refuerzo (RL):** Para la tarea compleja y dinámica de caminar y mantener el equilibrio (usando la red neuronal `policy_jit`).
2. **Control Clásico de Trayectorias:** Para la tarea precisa y coreografiada de saludar (usando `WaveController` con interpolación matemática).
3. **Simulación Física:** Usando MuJoCo para calcular las interacciones reales de fuerzas y gravedad

---

> Enlace del código completo para poder copiar, junto a las políticas de uso.
>
> - En el código se explica la funcionalidad de cada función y en las variables, para la ejecución
>   Saludo.py
>
>   > ⚠ **Recomendación**
>   > Revisar el código fuente.
>   > **Por favor leer los comentarios del código**
>
> - Enlace del repositorio donde estás todos los recursos
>   Repositorio