# Implementación de un contenedor - Ejecutando la simulación en Pybullet de un brazo robótico

Paula Sandoval

Abril 2025

## 1. Parte I - main.py

El script `main.py` desarrolla una simulación interactiva de un robot de dos articulaciones utilizando la librería `PyBullet`.

Permite al usuario controlar las articulaciones y la velocidad de simulación mediante una interfaz gráfica.

### 1.1. Importación de librerías

- `pybullet`: Motor de simulación física.
- `pybullet_data`: Acceso a modelos de ejemplo como el plano base.
- `time`: Control de retardos temporales.
- `numpy`: Manejo de constantes matemáticas como $\pi$.
- `os`: Verificación de existencia de archivos.

### 1.2. Configuración inicial

#### 1.2.1. Verificación del archivo URDF

Se comprueba que el archivo urdf esté en el directorio correcto para poder cargar el modelo del robot

```
if not os.path.exists(robot_urdf_path):
    print("ERROR: No se encuentra el archivo URDF")
    input("Presiona Enter para salir...")
    exit()
```

### 1.2.2. Inicialización de PyBullet

Se conecta PyBullet en modo GUI, se configura la gravedad y la cámara visual.

```
p.connect(p.GUI)
p.setGravity(0, 0, -9.8)
p.setAdditionalSearchPath(pybullet_data.getDataPath())
p.resetDebugVisualizerCamera(...)
```

### 1.2.3. Plano base y robot

Se carga el plano (`plane.urdf`) y luego el modelo del robot especificado.

```
planeId = p.loadURDF("plane.urdf")
robotId = p.loadURDF(robot_urdf_path, [0, 0, 0.1], useFixedBase=True)
```

## 1.3. Controles de usuario

Se crean deslizadores para controlar:

- **Joint 1**: Primera articulación.
- **Joint 2**: Segunda articulación.
- **Velocidad**: Ajuste dinámico de la velocidad de simulación.
- **Pausa/Continuar**: Botón para pausar o reanudar la simulación.

```
joint1_slider = p.addUserDebugParameter("Joint␣1", -np.pi, np.pi, 0)
joint2_slider = p.addUserDebugParameter("Joint␣2", -np.pi, np.pi, 0)
speed_control = p.addUserDebugParameter("Velocidad", 0.1, 10.0, 1.0)
pause_button = p.addUserDebugParameter("Pausar/Continuar", 1, 0, 1)
```

## 1.4. Bucle principal de simulación

Dentro de un bucle infinito, el programa:

1. Lee el estado del botón de pausa.
2. Obtiene los valores actuales de los deslizadores.
3. Ajusta las posiciones de las articulaciones usando `POSITION_CONTROL`.
4. Avanza la simulación un paso.
5. Ajusta el tiempo de espera para controlar la velocidad de simulación.

```
joint1_value = p.readUserDebugParameter(joint1_slider)
joint2_value = p.readUserDebugParameter(joint2_slider)
p.setJointMotorControl2(robotId, 0, p.POSITION_CONTROL, targetPosition=
    joint1_value, force=500)
p.setJointMotorControl2(robotId, 1, p.POSITION_CONTROL, targetPosition=
    joint2_value, force=500)
p.stepSimulation()
time.sleep(0.01 / speed)
```

### 1.5. Interrupciones

Se utiliza un bloque `try-except-finally` para:

- Capturar la interrupción por teclado (`KeyboardInterrupt`).
- Desconectar PyBullet de forma segura.
- Mantener la ventana abierta hasta que el usuario presione Enter.

El script `main.py` proporciona una simulación básica pero potente para estudiar el control de robots de múltiples articulaciones en PyBullet. La interfaz basada en deslizadores facilita la interacción y visualización inmediata de los movimientos.

## 2. Parte 2 - Dockerfile

El script Dockerfile desarrolla una imagen de Docker capaz de simular un entorno robótico basado en `PyBullet` y `ROS`.

Este documento resume y explica las instrucciones contenidas en el archivo `Dockerfile`, el cual prepara un entorno de ejecución para simular un robot utilizando `PyBullet`.

### 2.1. Inicialización

Se utiliza como imagen base `python:3`, una distribución oficial de Python 3 sobre Debian.

```
FROM python:3
```

### 2.2. Instalación de adiciones del sistema

Se actualizan los repositorios y se instalan paquetes necesarios para el soporte gráfico:

- `xvfb`: Servidor de visualización virtual.
- `x11-utils`: Utilidades para trabajar con el sistema de ventanas X11.
- `libgl1-mesa-glx`: Soporte para gráficos OpenGL.

```
RUN apt-get update && apt-get install -y \
    xvfb \
    x11-utils \
    libgl1-mesa-glx \
    && rm -rf /var/lib/apt/lists/*
```

## 2.3. Copia de archivos al contenedor

Se copian los archivos del proyecto (`main.py` y `urdf`) al contenedor.

```
COPY main.py .
COPY two_joint_robot_custom.urdf .
```

## 2.4. Instalación de dependencias en Python

Se instalan las librerías necesarias usando `pip`:

- `pybullet`: Motor de simulación física.
- `numpy`: Librería de operaciones numéricas.

```
RUN pip install pybullet
RUN pip install numpy
```

## 2.5. Comando de inicio

Se define el comando que ejecuta automáticamente el archivo `main.py` cuando se inicia el contenedor:

```
CMD command python3 main.py
```

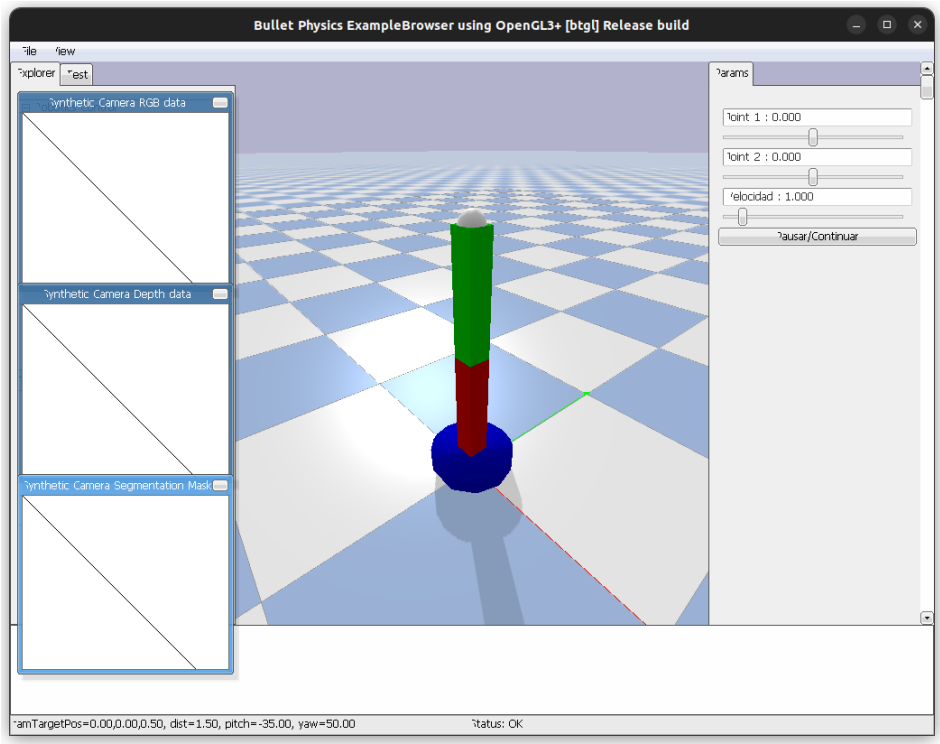


Figura 1: Generación de entorno Pybullet

# 3. Conclusión

Este `Dockerfile` configura un contenedor ligero y funcional para ejecutar simulaciones de robots utilizando PyBullet, asegurando compatibilidad gráfica y un entorno Python completo. Ideal para tareas de robótica, simulación y testing de movimientos articulados.