

Laboratorio 5: Aprendiendo Docker

Implementación de Entornos Robóticos con ROS, MoveIt, Gazebo y PyBullet

Cristian David Losada
Yojan Arley Contreras
Universidad Santo Tomás
Digitales 3

22 de octubre de 2025

Resumen

Este laboratorio tiene como objetivo principal el aprendizaje y aplicación de Docker en el contexto de la robótica y el Internet de las Cosas (IoT). Se implementan diversos entornos de simulación robótica utilizando ROS, MoveIt, Gazebo y PyBullet, demostrando la portabilidad y reproducibilidad que ofrece la contenerización mediante Docker.



Índice

1. Introducción	2
1.1. Objetivos del Laboratorio	2
2. Marco Teórico	2
2.1. Herramientas ROS, MoveIt y Gazebo en el contexto del IoT	2
2.1.1. ROS (Robot Operating System)	2
2.1.2. MoveIt	3
2.1.3. Gazebo	3
2.2. Robótica actual, sistemas LIDAR y sistemas SLAM orientados al IoT . .	4
2.2.1. Robótica actual y su integración con el IoT	4
2.2.2. Sistemas LIDAR (Light Detection and Ranging)	5
2.2.3. Sistemas SLAM (Simultaneous Localization and Mapping)	6
3. Desarrollo del Laboratorio	7
3.1. Potencial de Docker con ROS / MoveIt / Gazebo en IoT	7
3.2. Instalación de Docker	7
3.3. Ejemplo A - Robot Sencillo con ROS + MoveIt + Gazebo en Docker . .	9
3.4. Ejemplo B - Robot con LIDAR + SLAM (TurtleBot3 + gmapping) en Docker	12
3.5. Ejemplo C - Simulación de un Cuadrúpedo con PyBullet	14
4. Conclusiones	17
5. Recomendaciones	17

1 Introducción

1.1 Objetivos del Laboratorio

- Comprender los conceptos fundamentales de Docker y su aplicación en robótica
- Implementar contenedores Docker para entornos ROS, MoveIt y Gazebo
- Desarrollar simulaciones robóticas con sistemas LIDAR y SLAM
- Crear y ejecutar una simulación de cuadrúpedo usando PyBullet
- Analizar el potencial de Docker en sistemas IoT-Robóticos

2 Marco Teórico

2.1 Herramientas ROS, MoveIt y Gazebo en el contexto del IoT

2.1.1. ROS (Robot Operating System)

Definición:

ROS (Robot Operating System) es un marco de software de código abierto diseñado para el desarrollo de aplicaciones robóticas. Proporciona una arquitectura modular basada en nodos, que se comunican mediante mensajes a través de topics, servicios o acciones.

ROS no es un sistema operativo en sí, sino una plataforma middleware que permite la integración de sensores, actuadores, controladores y algoritmos en un entorno distribuido.

Alcance en el sector IoT:

En el contexto del Internet de las Cosas (IoT), ROS permite la conexión e integración de robots, sensores inteligentes, dispositivos embebidos y servicios en la nube.

Gracias a su capacidad de comunicación mediante protocolos como MQTT, WebSocket o REST API, ROS puede formar parte de sistemas distribuidos inteligentes, donde los robots interactúan con otros dispositivos IoT para la automatización, monitoreo o análisis de datos en tiempo real.

Aplicaciones Posibles:

- Domótica Inteligente: robots conectados con sistemas IoT domésticos (luces, cámaras, sensores ambientales).
- Industria 4.0: robots colaborativos conectados a redes IoT que reportan su estado a servidores o nubes industriales.
- Agricultura de precisión: drones y robots agrícolas controlados remotamente mediante ROS + IoT para supervisar cultivos.
- Telemedicina y logística: robots móviles monitoreados desde la nube para entrega de medicinas o transporte de insumos.

2.1.2. MoveIt

Definición:

MoveIt es un framework de planificación de movimiento y manipulación robótica integrado con ROS. Permite controlar brazos robóticos, calcular trayectorias seguras, evitar colisiones y realizar cinemática directa e inversa.

Se utiliza tanto para simulación como para control real de robots.

Alcance en el sector IoT:

En un entorno IoT, MoveIt permite que los brazos robóticos se integren en redes inteligentes, recibiendo órdenes desde sistemas en la nube o sensores distribuidos. Por ejemplo, un sistema IoT puede detectar un evento (como un objeto en una cinta transportadora) y enviar una instrucción al robot (a través de ROS y MoveIt) para realizar una acción específica.

Aplicaciones Posibles:

- Robótica colaborativa industrial: control de manipuladores conectados a sensores IoT que detectan piezas o movimientos humanos.
- Almacenes inteligentes: robots que planifican trayectorias automáticamente según datos en tiempo real de sensores o RFID.
- Cirugía robótica asistida: planeación de movimientos precisos en respuesta a señales biomédicas o comandos remotos.
- Laboratorios automatizados: brazos robóticos que manipulan muestras según lecturas de sensores IoT.

2.1.3. Gazebo

Definición:

Gazebo es un simulador 3D de entornos robóticos que permite modelar robots, sensores, objetos y escenarios complejos con física realista.

Se integra estrechamente con ROS, lo que permite simular la interacción del software de control con el entorno antes de implementarlo en el robot real.

Alcance en el sector IoT:

Gazebo permite simular ecosistemas IoT completos, donde los robots interactúan con sensores virtuales, redes inalámbricas y dispositivos conectados.

Esto reduce costos y riesgos, ya que se pueden probar sistemas IoT-robóticos completos sin necesidad de hardware físico.

Aplicaciones Posibles:

- Simulación de robots conectados en ciudades inteligentes (Smart Cities).
- Desarrollo de sistemas de transporte autónomos (vehículos, drones, AGVs) con comunicación IoT.

- Pruebas virtuales de sistemas domóticos antes de la instalación física.
- Educación y entrenamiento en robótica e IoT sin requerir equipos costosos.

2.2 Robótica actual, sistemas LIDAR y sistemas SLAM orientados al IoT

2.2.1. Robótica actual y su integración con el IoT

Definición y panorama actual:

La robótica actual ha evolucionado hacia sistemas inteligentes, conectados y colaborativos, capaces de percibir su entorno, tomar decisiones autónomas y comunicarse con otros dispositivos o plataformas.

Gracias al Internet de las Cosas (IoT), los robots ahora forman parte de ecosistemas interconectados, donde los datos provenientes de sensores, cámaras, redes y nubes son utilizados para optimizar su desempeño en tiempo real.

Tendencias en la robótica moderna:

1. Robots Colaborativos (cobots): diseñados para trabajar junto a humanos en entornos industriales, conectados a sensores IoT que monitorean seguridad y eficiencia.
2. Robots Móviles Autónomos (AMR): se comunican con servidores IoT para recibir rutas o tareas logísticas.
3. Robots Sociales y de Servicio: conectados a internet para reconocimiento de voz, navegación, atención al cliente o asistencia doméstica.
4. Robots en la Nube (Cloud Robotics): procesan información y algoritmos pesados desde servidores IoT o plataformas en la nube.
5. Edge Robotics: los robots procesan datos localmente (en el borde de la red) para reducir la latencia y depender menos de internet.

Aplicaciones en IoT:

- Smart Factories (Industria 4.0): comunicación robot-sensor-máquina para manufactura flexible.
- Agricultura Inteligente: drones y robots recolectan datos de humedad, temperatura o crecimiento de cultivos.
- Salud y Telepresencia: robots médicos o de asistencia conectados a redes IoT hospitalarias.
- Ciudades Inteligentes: robots de seguridad o limpieza gestionados desde sistemas IoT urbanos.

2.2.2. Sistemas LIDAR (Light Detection and Ranging)

Definición:

El LIDAR es una tecnología de detección y medición por luz láser, que emite pulsos y calcula la distancia a los objetos midiendo el tiempo que tarda la luz en regresar.

Produce nubes de puntos tridimensionales del entorno, fundamentales para la percepción y navegación de robots y vehículos autónomos.

Tipos principales de LIDAR:

1. LIDAR mecánico:

- Utiliza espejos giratorios para escanear 360°.
- Muy usado en vehículos autónomos, robots móviles y drones.
- Ejemplo: Velodyne HDL-64E.

2. LIDAR Sólido (Solid-State):

- Sin partes móviles, más compacto y resistente.
- Ideal para integración en dispositivos IoT o robots de bajo costo.
- Ejemplo: Luminar Iris.

3. LIDAR Flash:

- Captura toda la escena con un solo pulso láser.
- Menor alcance, pero alta velocidad de captura.
- Usado en robots domésticos, drones y smartphones.

4. LIDAR MEMS (Micro-Electro-Mechanical Systems):

- Miniaturizado, bajo consumo energético, ideal para IoT portátil.
- Utilizado en mapeo ambiental, wearables o robots pequeños.

Relación con el IoT:

Los sistemas LIDAR conectados a IoT permiten transmitir datos espaciales en tiempo real hacia plataformas de análisis en la nube, facilitando tareas de monitoreo, cartografía o navegación autónoma colectiva.

Ejemplo: múltiples robots o vehículos compartiendo sus mapas LIDAR en una red IoT para evitar colisiones o actualizar mapas urbanos.

Aplicaciones:

- Vehículos autónomos conectados a redes IoT.
- Robots de entrega o limpieza con percepción compartida.
- Monitoreo ambiental y de infraestructura (vías, puentes, cultivos).
- Gestión de almacenes y control logístico inteligente.

2.2.3. Sistemas SLAM (Simultaneous Localization and Mapping)

Definición:

El SLAM (Localización y Mapeo Simultáneo) es el conjunto de algoritmos que permite a un robot crear un mapa de su entorno y localizarse dentro de él al mismo tiempo, sin depender de un GPS.

Combina información de sensores LIDAR, cámaras, ultrasonido, IMU o radar, junto con técnicas de fusión sensorial y estimación probabilística.

Tipos principales de SLAM:

1. Visual SLAM (vSLAM):

- Utiliza cámaras monoculares, estéreo o RGB-D.
- Ideal para drones, robots domésticos o aplicaciones IoT con cámaras.
- Ejemplo: ORB-SLAM, LSD-SLAM.

2. LIDAR-based SLAM:

- Usa escáneres LIDAR 2D/3D.
- Alta precisión en entornos estructurados (interiores industriales).
- Ejemplo: Gmapping, Hector SLAM, Cartographer.

3. Inertial SLAM (IMU-SLAM):

- Integra datos de acelerómetros y giróscopos (IoT inercial).
- Utilizado en dispositivos móviles, drones y robots de exploración.

4. RGB-D SLAM:

- Usa cámaras de profundidad (como Kinect o RealSense).
- Permite mapas 3D densos en entornos complejos.

5. Multi-robot SLAM:

- Varios robots IoT cooperan compartiendo mapas en red.
- Aplicado en exploración colaborativa o logística inteligente.

Relación con el IoT:

En sistemas IoT, el SLAM se complementa con la comunicación en red: los robots pueden compartir sus mapas, actualizar datos en la nube y recibir correcciones desde un servidor remoto.

Esto genera ecosistemas inteligentes distribuidos, donde varios robots o vehículos actualizan un mapa común en tiempo real.

Aplicaciones:

- Robots autónomos conectados en fábricas o almacenes IoT.

- Vehículos autónomos urbanos.
- Drones agrícolas con mapeo colaborativo.
- Robots de limpieza o vigilancia conectados a plataformas IoT.

3 Desarrollo del Laboratorio

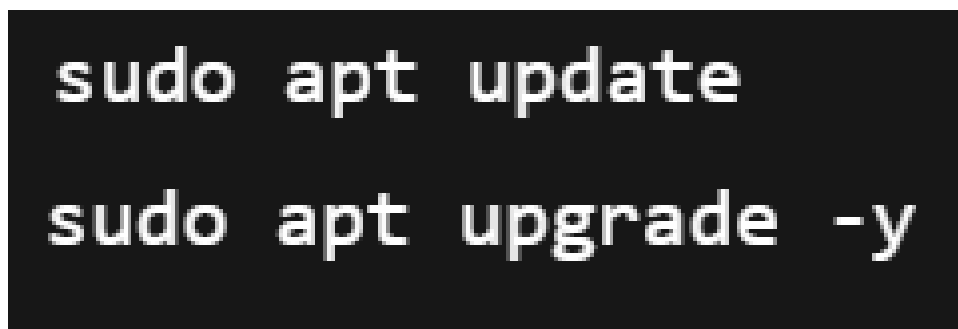
3.1 Potencial de Docker con ROS / MoveIt / Gazebo en IoT

- **Reproducibilidad y aislamiento:** Docker encapsula versiones exactas de ROS, MoveIt y Gazebo (dependencias, bibliotecas, versiones de Ubuntu). Esto evita el ^{en} mi máquina funciona y facilita desplegar en múltiples gateways/edge servers del ecosistema IoT.
- **Portabilidad y despliegue en edge/cloud:** las imágenes Docker se pueden ejecutar en PCs de desarrollo, servidores en la nube o gateways IoT (si la arquitectura lo permite). Permite pasar contenedores probados en simulación a infraestructuras de integración continua o a nodos edge.
- **Orquestación y microservicios:** se pueden correr nodos ROS (masters, nodos sensores, nodos de procesamiento ML, bases de datos) en contenedores separados y orquestarlos (docker-compose, Kubernetes) para sistemas distribuidos IoT.
- **Testing y CI/CD:** facilita ejecutar pruebas automáticas (simulaciones con Gazebo) en servidores CI antes de desplegar robots físicos.
- **Soporte de GUI y hardware (X11 / GPU / USB):** con las opciones correctas (montar /tmp/.X11-unix, pasar DISPLAY, y/o drivers NVIDIA), puedes ejecutar RViz/Gazebo dentro de un contenedor y acceder a GPU o a interfaces USB (LIDAR real) desde el host.

3.2 Instalación de Docker

1. Actualizar el sistema

Abre la terminal y ejecuta:



```
sudo apt update
sudo apt upgrade -y
```

Figura 1: Actualizar el sistema

2. Instalar dependencias necesarias

Estas herramientas permiten añadir el repositorio oficial de Docker:

```
sudo apt install ca-certificates curl gnupg lsb-release -y
```

Figura 2: Instalar Dependencias

3. Añadir la clave GPG oficial de Docker

```
sudo mkdir -p /etc/apt/keyrings
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg
```

```
sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

Figura 3: Añadir Clave GPG

4. Añadir el repositorio de Docker

```
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Figura 4: Añadir Repositorio

5. Instalar Docker Engine, CLI y containerd

```
sudo apt update
sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
```

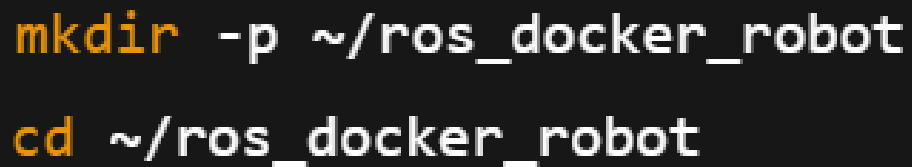
```
docker-compose-plugin -y
```

Figura 5: Instalación de Docker

3.3 Ejemplo A - Robot Sencillo con ROS + MoveIt + Gazebo en Docker

Objetivo: Crear un robot sencillo con movimiento básico usando ROS, MoveIt y Gazebo dentro de Docker.

1. Crear un directorio de trabajo

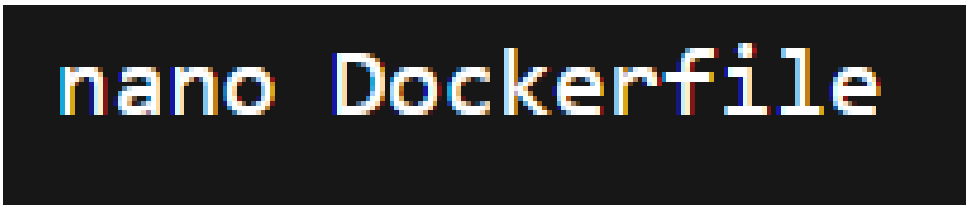


```
mkdir -p ~/ros_docker_robot  
cd ~/ros_docker_robot
```

Figura 6: Creación de directorio

2. Crear el archivo Dockerfile

Crea el archivo:



```
nano Dockerfile
```

Escribimos esto dentro del archivo:

```
# Imagen base con ROS Noetic + Gazebo + MoveIt
FROM osrf/ros:noetic-desktop-full

# Configuración del entorno
SHELL ["/bin/bash", "-c"]
ENV ROS_DISTRO=noetic

# Instalar MoveIt y TurtleBot3
RUN apt update && apt install -y \
    ros-noetic-moveit \
    ros-noetic-turtlebot3-simulations \
    ros-noetic-turtlebot3-gazebo \
    && rm -rf /var/lib/apt/lists/*

# Configurar variables de entorno
ENV TURTLEBOT3_MODEL=burger

# Comando por defecto al iniciar el contenedor
CMD ["bash"]
```

Figura 7: Creación del archivo Dockerfile

3. Construir la imagen

```
docker build -t ros-moveit-gazebo .
```

Figura 8: Construcción de la imagen

4. Ejecutar el contenedor con interfaz gráfica

Para que Gazebo y RViz se vean, habilitamos X11:

```
xhost +local:root
docker run -it --rm \
  --name ros_robot \
  --env="DISPLAY" \
  --env="QT_X11_NO_MITSHM=1" \
  --volume="/tmp/.X11-unix:/tmp/.X11-unix:rw" \
  ros-moveit-gazebo
```

Figura 9: Ejecución del contenedor

5. Lanzar la simulación del robot TurtleBot3

Dentro del contenedor:

```
source /opt/ros/noetic/setup.bash
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

Figura 10: Lanzamiento de la simulación

6. Controlar el movimiento del robot

Ahora abrimos otra terminal de Ubuntu y ejecutamos:

```
docker exec -it ros_robot bash
source /opt/ros/noetic/setup.bash
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Figura 11: Control del robot con el teclado

7. Resultado final:

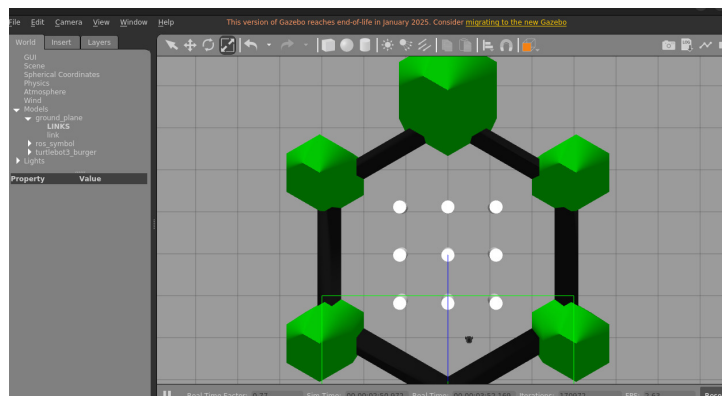
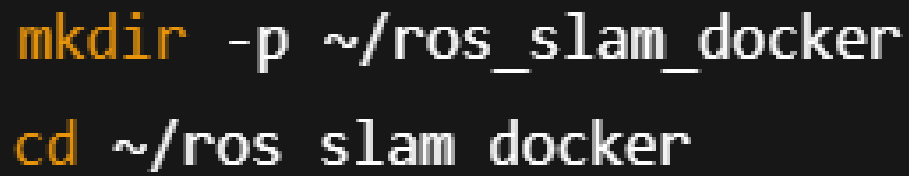


Figura 12: Resultado Final Robot TurtleBot3

3.4 Ejemplo B - Robot con LIDAR + SLAM (TurtleBot3 + gmapping) en Docker

Objetivo: Implementar un robot con sistema LIDAR y SLAM corriendo dentro de Docker usando SLAM Toolbox para crear un mapa del entorno y navegar en él.

1. Crear un nuevo directorio



```
mkdir -p ~/ros_slam_docker  
cd ~/ros_slam_docker
```

Figura 13: Creación de nuevo directorio

2. Crear el archivo Dockerfile



```
nano Dockerfile
```

```
# Imagen base con ROS Noetic + Gazebo
FROM osrf/ros:noetic-desktop-full

SHELL ["/bin/bash", "-c"]
ENV ROS_DISTRO=noetic
ENV TURTLEBOT3_MODEL=burger

# Instalar dependencias
RUN apt update && apt install -y \
    ros-noetic-turtlebot3-simulations \
    ros-noetic-turtlebot3-gazebo \
    ros-noetic-slam-toolbox \
    ros-noetic-map-server \
    && rm -rf /var/lib/apt/lists/*

CMD ["bash"]
```

Figura 14: Creación del archivo Dockerfile

3. Construir la imagen

```
docker build -t ros-slam-gazebo .
```

Figura 15: Construcción de la imagen

4. Ejecutar el contenedor con soporte gráfico

```
xhost +local:root
docker run -it --rm \
  --name ros_slam_robot \
  --env="DISPLAY" \
  --env="QT_X11_NO_MITSHM=1" \
  --volume="/tmp/.X11-unix:/tmp/.X11-unix:rw" \
  ros-slam-gazebo
```

Figura 16: Ejecución del contenedor

5. Lanzar Gazebo con TurtleBot3

Dentro del contenedor:

```
source /opt/ros/noetic/setup.bash
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

Figura 17: Lanzamiento de Gazebo

6. Resultado Final:

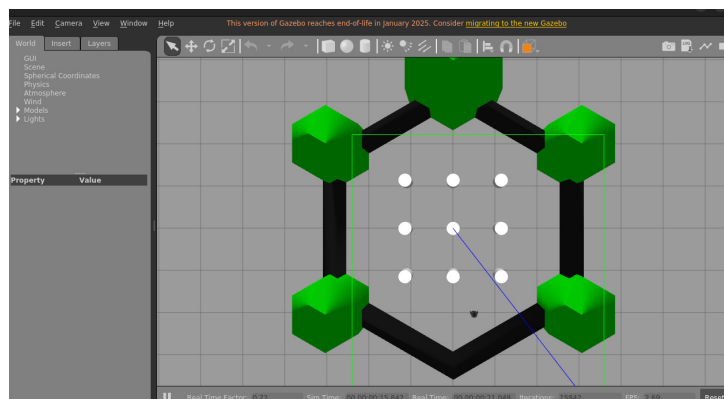


Figura 18: Resultado Final con LIDAR y SLAM

3.5 Ejemplo C - Simulación de un Cuadrúpedo con PyBullet

Objetivo: Controlar un cuadrúpedo con PyBullet y desplegarlo en Docker.

1. Clonar el repositorio

```
cris1711@CrisNightWolf1711:~$ cd ~
git clone https://github.com/erwincoumans/motion_imitation.git
cd motion_imitation
Clonando en 'motion_imitation'...
remote: Enumerating objects: 1850, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 1850 (delta 0), reused 1 (delta 0), pack-reused 1847 (from 1)
Recibiendo objetos: 100% (1850/1850), 18.38 MiB | 1.22 MiB/s, listo.
Resolviendo deltas: 100% (451/451), listo.
cris1711@CrisNightWolf1711:~/motion_imitation$
```

Figura 19: Clonación del repositorio

2. Crear el archivo Dockerfile

Creamos un archivo llamado Dockerfile dentro de la carpeta motion imitation:

```
cris1711@CrisNightWolf1711:~/motion_imitation$ nano Dockerfile
```

Colocamos esto dentro del archivo:

```
FROM python:3.8-slim

RUN apt-get update && apt-get install -y \
    git \
    python3-pip \
    xvfb \
    x11-utils \
    && rm -rf /var/lib/apt/lists/*

WORKDIR /app

COPY . /app

RUN pip install --upgrade pip && \
    pip install -e . && \
    pip install pybullet numpy matplotlib

CMD ["python3", "scripts/run_simulation.py"]
```

Figura 20: Creación de archivo Dockerfile

3. Crear el archivo run_simulation.py


```
cris1711@CrisNightWolf1711:~/motion_imitation$ nano scripts/run_simulation.py
```

Colocamos esto dentro del archivo:

```
import pybullet as p
import pybullet_data
import time

p.connect(p.GUI)
p.setAdditionalSearchPath(pybullet_data.getDataPath())
p.loadURDF("plane.urdf")
robot = p.loadURDF("quadruped/minitaur.urdf", [0, 0, 0.2])
p.setGravity(0, 0, -9.8)

for i in range(2000):
    p.stepSimulation()
    time.sleep(1./240.)

p.disconnect()
```

Figura 21: Creación de archivo de la simulación

4. Construimos la imagen de Docker

```
cris1711@CrisNightWolf1711:~/motion_imitation$ sudo docker build -t pybullet_quadruped .
```

Figura 22: Construcción de la imagen de PyBullet

5. Ejecutamos el contenedor con la interfaz gráfica

1. Activamos el servidor X11:

```
cris1711@CrisNightWolf1711:~/motion_imitation$ xhost +
access control disabled, clients can connect from any host
cris1711@CrisNightWolf1711:~/motion_imitation$
```

2. Ejecutamos el contenedor:

```
cris1711@CrisNightWolf1711:~$ sudo docker run -it --rm \
--env="DISPLAY" \
--net=host \
-v /tmp/.X11-unix:/tmp/.X11-unix \
pybullet_quadruped
```

Figura 23: Ejecución del contenedor con PyBullet

6. Resultado final:

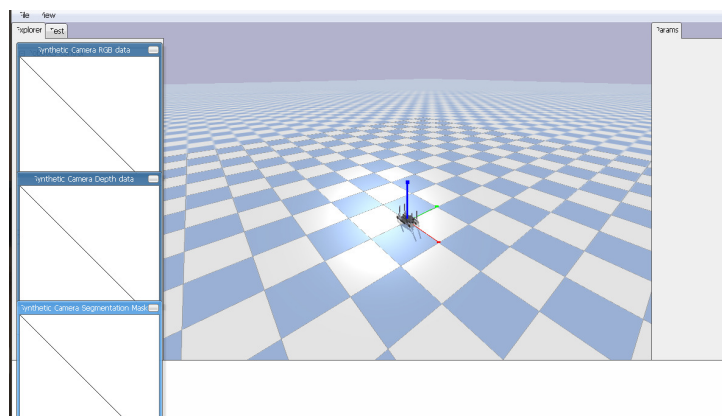


Figura 24: Simulación del cuadrúpedo en PyBullet

4 Conclusiones

- Docker demostró ser una herramienta fundamental para la containerización de aplicaciones robóticas, permitiendo la reproducibilidad y portabilidad de los entornos de desarrollo.
- La integración de ROS, MoveIt y Gazebo en contenedores Docker facilita el despliegue de sistemas robóticos en entornos IoT distribuidos.
- Los sistemas LIDAR y SLAM containerizados muestran el potencial de Docker para aplicaciones de percepción y navegación autónoma en robótica.
- PyBullet en Docker permite la simulación avanzada de robots complejos como cuadrúpedos, demostrando la versatilidad de la containerización.
- El uso de Docker en robótica e IoT representa una metodología moderna para el desarrollo, prueba y despliegue de sistemas inteligentes distribuidos.

5 Recomendaciones

- Utilizar Docker Compose para orquestar múltiples contenedores en sistemas robóticos complejos.
- Implementar estrategias de CI/CD con Docker para automatizar el testing de aplicaciones robóticas.
- Explorar el uso de Docker Swarm o Kubernetes para el despliegue de flotas de robots en entornos IoT.
- Considerar el uso de Docker en combinación con plataformas edge computing para aplicaciones robóticas con baja latencia.
- Documentar adecuadamente las imágenes Docker creadas para facilitar su reutilización y mantenimiento.