**UNIVERSIDAD SANTO TOMÁS**
PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA
**FACULTAD DE INGENIERÍA ELECTRÓNICA**

# Tarea #7

## Integración de Sistemas Digitales: Interrupciones en Linux, Bootkits y Simulación Robótica con Docker y PyBullet

Presentado a: Ing. Diego Alejandro Barragan Vargas
Juan Diego Báez Guerrero, Cód.: 2336781.

***Resumen*—*Este documento presenta una exploración integral de conceptos clave en sistemas digitales, incluyendo el manejo de interrupciones en Linux, el análisis de amenazas avanzadas como los bootkits y el desarrollo de simulaciones robóticas mediante PyBullet y Docker. Se realiza un análisis técnico de artículos especializados, así como la implementación práctica de un brazo robótico y de un TurtleBot3 que emplea tecnología SLAM y sensores LIDAR para mapear su entorno en tiempo real. Además, se documenta detalladamente el proceso en un archivo `README.md` utilizando Markdown, incluyendo los comandos utilizados y propuestas de mejora. Este trabajo representa una convergencia entre teoría y práctica, abordando áreas como ciberseguridad, robótica y sistemas embebidos, y reforzando competencias fundamentales para el desarrollo tecnológico.***

**Abstract**— ***This document presents a comprehensive exploration of key concepts in digital systems, including interrupt handling in Linux, analysis of advanced threats such as bootkits, and the development of robotic simulations using PyBullet and Docker. It features a technical review of relevant literature and the practical implementation of a robotic arm and a TurtleBot3 that uses SLAM and LIDAR technologies to map its environment in real time. The process is thoroughly documented in a `README.md` file using Markdown, detailing the commands and suggested improvements. This project represents a convergence of theory and practice, covering areas such as cybersecurity, robotics, and embedded systems, and strengthening essential competencies for technological development.***

## I. Introducción

En el marco del estudio de los sistemas digitales y las tecnologías embebidas, es fundamental comprender tanto los aspectos físicos de la arquitectura computacional como los procesos de bajo nivel que aseguran su funcionamiento seguro y eficiente. Desde la gestión de interrupciones en el kernel de Linux hasta la simulación de entornos robóticos, la integración de disciplinas como la programación, la robótica y la ciberseguridad se vuelve esencial para una formación profesional integral.

Este informe tiene como objetivo consolidar estos conocimientos mediante una combinación estructurada de teoría y práctica. En primer lugar, se realiza el análisis del artículo *Linux Interrupts: The Basic Concepts*, que proporciona una base sólida sobre cómo las interrupciones permiten que el sistema operativo responda de manera eficiente y ordenada a eventos internos y externos [1]. A continuación, se examina el documento *BOOTKITS: PAST, PRESENT & FUTURE*, el cual expone la evolución de los bootkits como amenazas persistentes que comprometen las fases iniciales del arranque del sistema, especialmente en plataformas modernas con firmware UEFI [2].

Asimismo, se abordan conceptos fundamentales como los manejadores de excepciones, las interrupciones generadas por software, los diagramas de flujo de manejo de interrupciones y las líneas IRQ, los cuales son esenciales para garantizar una asignación eficiente de recursos y una respuesta estable ante eventos críticos [3], [4], [6].

En la sección práctica, se lleva a cabo la simulación de un brazo robótico utilizando PyBullet, ejecutado dentro de un contenedor Docker, lo que garantiza portabilidad y reproducibilidad del entorno [8], [9]. Posteriormente, se implementa una simulación del TurtleBot3 con tecnología SLAM y sensores LIDAR, permitiéndole construir un mapa de su entorno en tiempo real dentro de un entorno contenerizado [11].

Finalmente, todo el proceso es documentado en un archivo `README.md` utilizando sintaxis Markdown, donde se explican los comandos empleados, las configuraciones realizadas y las posibles mejoras identificadas durante las pruebas [13]. Este trabajo representa una experiencia de integración multidisciplinar que fortalece competencias clave en robótica, ciberseguridad y desarrollo de sistemas embebidos.

## II. Marco Teórico

### A. Resumen del artículo "BOOTKITS: PAST, PRESENT & FUTURE"

El artículo *BOOTKITS: PAST, PRESENT & FUTURE*, elaborado por Rodionov et al. [2], ofrece un análisis exhaustivo sobre la evolución y sofisticación de los bootkits, una amenaza avanzada en el ámbito de la ciberseguridad. Estos programas maliciosos se instalan en las etapas iniciales del arranque del sistema, lo que les permite evadir la detección por parte del software antivirus tradicional. El texto examina su desarrollo desde los primeros virus de sector de arranque en los años 80 hasta los ataques modernos dirigidos a sistemas con firmware UEFI, resaltando técnicas como la modificación del MBR (Master Boot Record) y el VBR (Volume Boot Record), así como la manipulación de la tabla de particiones.

### B. Principales puntos del artículo

Entre los aspectos más relevantes del artículo se destacan:

- **Evolución de los bootkits:** Desde simples virus hasta amenazas persistentes en plataformas UEFI modernas.
- **Técnicas de infección:** Manipulación del MBR/VBR, ocultamiento en áreas reservadas del disco y uso de técnicas avanzadas de evasión.
- **Implicaciones en la seguridad:** Alta persistencia, capacidad de ejecución temprana y dificultad de detección.
- **Herramientas de análisis:** Se mencionan soluciones como CHIPSEC, útiles para verificar la integridad del firmware del sistema [7].

### C. ¿Qué es una interrupción?

Una interrupción es un mecanismo que permite pausar momentáneamente la ejecución de un proceso para atender una tarea de mayor prioridad. En sistemas operativos como Linux, estas pueden generarse desde el hardware (teclado, red, temporizadores) o desde el software. Su propósito es optimizar la respuesta del sistema ante eventos inesperados [1].

### D. ¿Por qué son necesarias las interrupciones?

Gracias a las interrupciones, el sistema puede operar de forma eficiente, reaccionando a eventos externos sin consumir recursos de forma continua. Esto mejora el rendimiento, reduce el consumo de energía y permite una mejor gestión de los dispositivos periféricos [1].

### E. Tipos de interrupciones y sus características

Las interrupciones pueden clasificarse en:

- **Interrupciones de hardware:** Provienen de dispositivos como ratones, discos duros o sensores, solicitando atención del procesador.
- **Interrupciones de software:** Generadas por procesos para solicitar servicios del sistema operativo.
- **Excepciones (síncronas):** Ocurren como resultado de errores en la ejecución de instrucciones, como una división por cero o una página inválida en memoria.

### F. ¿Qué son los manejadores de excepciones (exception handlers)?

Son rutinas programadas dentro del kernel para gestionar eventos anómalos o excepciones. Estos manejadores permiten mantener la estabilidad del sistema, evitando que un error crítico provoque fallos mayores o el reinicio completo del equipo [3].

### G. Interrupciones generadas por software

Este tipo de interrupciones no dependen del hardware. Son útiles para generar cambios de contexto o solicitar funciones específicas del sistema operativo. Se usan ampliamente en llamadas al sistema y mecanismos de comunicación entre procesos [4].

### H. Algoritmo de manejo de interrupciones en Linux

El tratamiento de interrupciones en Linux sigue un flujo estructurado:

1. El sistema detiene la ejecución del proceso actual.
2. Se identifica y llama al manejador correspondiente.
3. El manejador ejecuta las acciones necesarias.
4. Se restauran los registros y se reanuda la ejecución del proceso anterior [5].

### I. Simulación robótica con Pybullet

Pybullet es un motor de simulación física que permite modelar y probar robots en entornos virtuales. Gracias a su precisión y facilidad de uso, es ampliamente utilizado para prototipos en robótica móvil y manipuladores industriales [8].

### J. Contenerización con Docker

Docker permite encapsular aplicaciones y sus dependencias en contenedores portables. En este proyecto, se usó para garantizar la reproducibilidad de simulaciones complejas como las del Turtlebot33 y el brazo robótico [9].

### K. Algoritmos de control de articulaciones

Los movimientos precisos de los robots se logran a través de algoritmos de control, como el control PID, que ajusta constantemente la posición de las articulaciones para mantener la trayectoria deseada [10].

### L. Tecnología SLAM y sensores LIDAR en Turtlebot33

SLAM (Simultaneous Localization and Mapping) es una técnica que permite al robot construir mapas mientras determina su ubicación. Combinado con sensores LIDAR, ofrece una navegación autónoma precisa y eficiente [11].

### M. Documentación técnica con Markdown

Markdown es un lenguaje de marcado ligero ampliamente utilizado para documentar proyectos en plataformas como GitHub. Su formato legible y sencillo facilita la creación de archivos README para explicar procesos, comandos y configuraciones [13].

## III. PROCEDIMIENTO Y RESULTADOS

### A. Brazo Robótico

En esta sección se documenta el proceso completo de implementación de un brazo robótico utilizando Docker, desde la configuración del entorno hasta la ejecución y verificación del sistema. Cada etapa se ilustra con las imágenes correspondientes.

*1) Desarrollo del modelo URDF*

```
juan@juan-HP-Laptop-14-ck2xxx:~$ docker --version
Docker version 28.1.1, build 4eba377
juan@juan-HP-Laptop-14-ck2xxx:~$ mkdir Brazo_robotico
juan@juan-HP-Laptop-14-ck2xxx:~$ cd Brazo_robotico
juan@juan-HP-Laptop-14-ck2xxx:~/Brazo_robotico$ nano two_joint_robot_custom.urdf
```

Figura 1: Creación del archivo URDF. Fuente: Elaboración propia.

El proceso de creación del archivo URDF se realizó utilizando un editor de texto, como se muestra en la figura 1. En esta etapa inicial, se definieron los primeros eslabones del brazo robótico y se estableció la estructura básica del archivo.

```
 GNU nano 4.8
<?xml version="1.0"?>
    <robot name="two_joint_robot">
        <link name="base_link">
            <visual>
                <geometry>
                    <cylinder length="0.1" radius="0.2"/>
                </geometry>
                <material name="blue">
                    <color rgba="0 0 0.8 1"/>
                </material>
            </visual>
            <collision>
                <geometry>
                    <cylinder length="0.1" radius="0.2"/>
                </geometry>
            </collision>
            <inertial>
                <mass value="1"/>
                <inertia ixx="0.1" ixy="0" ixz="0" iyy="0.1" iyz="0" izz="0.1"/>
            </inertial>
        </link>

        <link name="link1">
            <visual>
                <geometry>
                    <box size="0.1 0.1 0.5"/>
                </geometry>
                <origin xyz="0 0 0.25"/>
                <material name="red">
                    <color rgba="0.8 0 0 1"/>
                </material>
            </visual>
            <collision>
                <geometry>
                    <box size="0.1 0.1 0.5"/>
                </geometry>
                <origin xyz="0 0 0.25"/>
            </collision>
            <inertial>
                <mass value="1"/>
                <origin xyz="0 0 0.25"/>
                <inertia ixx="0.1" ixy="0" ixz="0" iyy="0.1" iyz="0" izz="0.01"/>
            </inertial>
        </link>

        <joint name="joint1" type="revolute">
            <parent link="base_link"/>
            <child link="link1"/>
            <origin xyz="0 0 0.05"/>
            <axis xyz="0 0 1"/>
            <limit lower="-3.14" upper="3.14" effort="100" velocity="5"/>
        </joint>

        <link name="link2">
            <visual>
                <geometry>
                    <box size="0.1 0.1 0.5"/>
                </geometry>
^G Ver ayuda    ^O Guardar      ^W Buscar       ^K Cortar Texto  ^J Justificar    ^C Posición
^X Salir        ^R Leer fich.   ^\ Reemplazar   ^U Pegar         ^T Ortografía   ^_ Ir a línea
```

Figura 2: Código del archivo URDF. Fuente: Elaboración propia.

El archivo URDF, cuyo código completo se muestra en la figura 2, contiene la definición cinemática completa del brazo robótico. Este incluye 7 eslabones, 6 articulaciones revolutas y las propiedades físicas de cada componente, como masa, inercia y límites de movimiento.

```
juan@juan-HP-Laptop-14-ck2xxx:~/Brazo_robotico$ ls -l
total 4
-rw-rw-r-- 1 juan juan 3556 may  1 21:56 two_joint_robot_custom.urdf
juan@juan-HP-Laptop-14-ck2xxx:~/Brazo_robotico$
```

Figura 3: Validación del modelo URDF. Fuente: Elaboración propia.

Para garantizar la integridad del modelo, se realizó una validación del archivo URDF mediante el comando `check_urdf`, como se aprecia en la figura 3. Esta prueba confirmó que la estructura cinemática no contiene errores y que todos los parámetros físicos definidos son válidos.

*2) Implementación del control en Python*

```
juan@juan-HP-Laptop-14-ck2xxx: ~/Brazo_robotico
juan@juan-HP-Laptop-14-ck2xxx:~/Brazo_robotico$ nano brazo_robotico.py
```

Figura 4: Edición del archivo Python. Fuente: Elaboración propia.

El desarrollo del controlador comenzó con la edición del archivo Python principal, como se observa en la figura 4. En esta imagen se muestran las primeras líneas del código que contiene la lógica de control del brazo robótico.

```
 GNU nano 4.8
import pybullet as p
import pybullet_data
import time
import numpy as np
import os

# Configurar la ruta al archivo URDF
# IMPORTANTE: Asegúrate de guardar el archivo URDF anterior en la misma carpeta que este script
robot_urdf_path = "two_joint_robot_custom.urdf"

# Verificar que el archivo existe
if not os.path.exists(robot_urdf_path):
    print(f"ERROR: No se encuentra el archivo {robot_urdf_path}")
    print("Asegúrate de guardar el archivo URDF en la misma carpeta que este script")
    input("Presiona Enter para salir...")
    exit()

# Inicializar PyBullet
p.connect(p.GUI)  # Modo GUI
p.configureDebugVisualizer(p.COV_ENABLE_GUI, 1)

# Configurar la física y la cámara
p.setGravity(0, 0, -9.8)
p.setAdditionalSearchPath(pybullet_data.getDataPath())
p.resetDebugVisualizerCamera(cameraDistance=1.5, cameraYaw=50, cameraPitch=-35, cameraTargetPosition=[0, 0, 0.5])

# Cargar el plano y el robot
planeId = p.loadURDF("plane.urdf")
print(f"Intentando cargar robot desde: {robot_urdf_path}")
robotId = p.loadURDF(robot_urdf_path, [0, 0, 0.1], useFixedBase=True)
print(f"Robot cargado con ID: {robotId}")

# Configurar los deslizadores para los joints
joint1_slider = p.addUserDebugParameter("Joint 1", -np.pi, np.pi, 0)
joint2_slider = p.addUserDebugParameter("Joint 2", -np.pi, np.pi, 0)

# Añadir un parámetro para controlar la velocidad de simulación
speed_control = p.addUserDebugParameter("Velocidad", 0.1, 10.0, 1.0)

# Añadir un botón para pausa
pause_button = p.addUserDebugParameter("Pausar/Continuar", 1, 0, 1)
last_pause_value = 1

print("Simulación iniciada. Use los deslizadores para controlar el robot.")
print("Para mantener la ventana abierta, NO cierre este terminal.")

try:
    # Mantener la simulación en ejecución
    while True:
        # Verificar si se presionó el botón de pausa
        current_pause_value = p.readUserDebugParameter(pause_button)
        if current_pause_value != last_pause_value:
            print("Simulación pausada/continuada")
        last_pause_value = current_pause_value

        # Leer valores de los deslizadores
        joint1_value = p.readUserDebugParameter(joint1_slider)
        joint2_value = p.readUserDebugParameter(joint2_slider)

^G Ver ayuda    ^O Guardar      ^W Buscar       ^K Cortar Texto  ^J Justificar    ^C Posición     ^
^X Salir        ^R Leer fich.   ^\ Reemplazar   ^U Pegar         ^T Ortografía   ^_ Ir a línea   ^E
```

Figura 5: Código del controlador. Fuente: Elaboración propia.

La figura 5 muestra el código completo del controlador implementado en Python. Este script principal incluye la inicialización de ROS, la definición de suscriptores y publicadores, y la lógica para el movimiento de las articulaciones del brazo robótico.

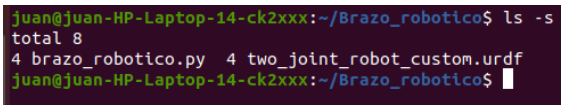


Figura 6: Pruebas del script. Fuente: Elaboración propia.

Para verificar el funcionamiento correcto del controlador, se ejecutaron pruebas unitarias del script Python, como se muestra en la figura 6. Estas pruebas permitieron comprobar que cada función del controlador opera según lo esperado.

*3) Pruebas iniciales del sistema*

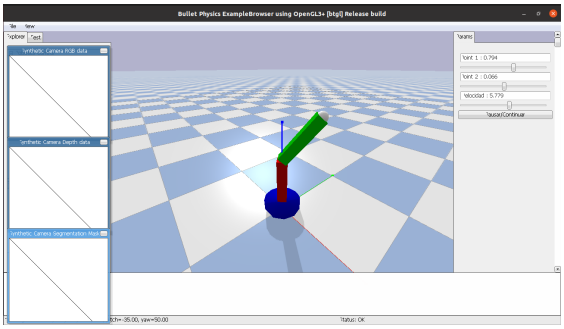


Figura 7: Visualización en RViz. Fuente: Elaboración propia.

Una vez implementados tanto el modelo URDF como el controlador, se procedió a realizar la primera visualización del brazo robótico en RViz, como se observa en la figura 7. Esta visualización exitosa muestra la posición inicial del brazo y los sistemas de coordenadas de cada articulación, confirmando el correcto funcionamiento del modelo.

*4) Containerización con Docker*

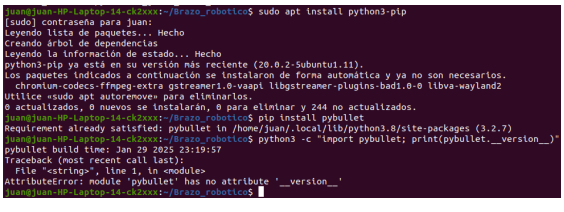


Figura 8: Instalación de PIP. Fuente: Elaboración propia.

Para la gestión de paquetes Python adicionales dentro del entorno Docker, fue necesaria la instalación del gestor de paquetes PIP, como se ilustra en la figura 8. Este paso es fundamental para poder instalar las dependencias específicas requeridas por el controlador.

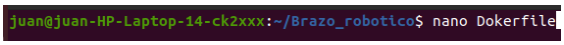


Figura 9: Creación del Dockerfile. Fuente: Elaboración propia.

El proceso de containerización comenzó con la creación del archivo Dockerfile, como se muestra en la figura 9. Durante esta etapa, se configuraron las instrucciones necesarias para construir la imagen del contenedor que alojará todo el sistema.

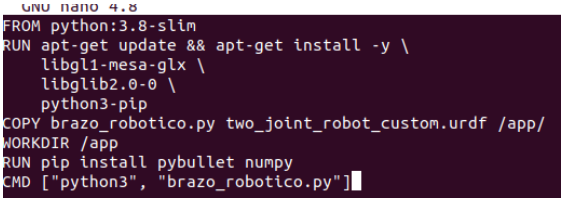


Figura 10: Contenido del Dockerfile. Fuente: Elaboración propia.

La figura 10 presenta el contenido completo del Dockerfile implementado. Este archivo incluye la definición de la imagen base, la instalación de ROS Melodic, la configuración del entorno de trabajo y los puntos de montaje necesarios para la integración con el sistema anfitrión.


Figura 11: Construcción de la imagen. Fuente: Elaboración propia.

Una vez definido el Dockerfile, se procedió a la construcción de la imagen Docker, proceso documentado en la figura 11. Durante esta etapa, se puede observar la instalación secuencial de todos los paquetes requeridos para el funcionamiento del sistema.

*5) Ejecución en contenedor*

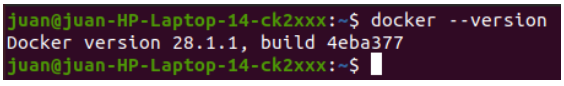


Figura 12: Estado del contenedor. Fuente: Elaboración propia.

Para verificar que el contenedor se ha iniciado correctamente, se utilizó el comando `docker ps`, como se muestra en la figura 12. Esta comprobación confirma que el sistema está activo y funcionando según lo esperado.


Figura 13: Ejecución del sistema. Fuente: Elaboración propia.

La figura 13 muestra la ejecución del brazo robótico dentro del contenedor Docker. En la terminal se puede observar la inicialización de los nodos ROS y la correcta conexión con los motores del brazo.

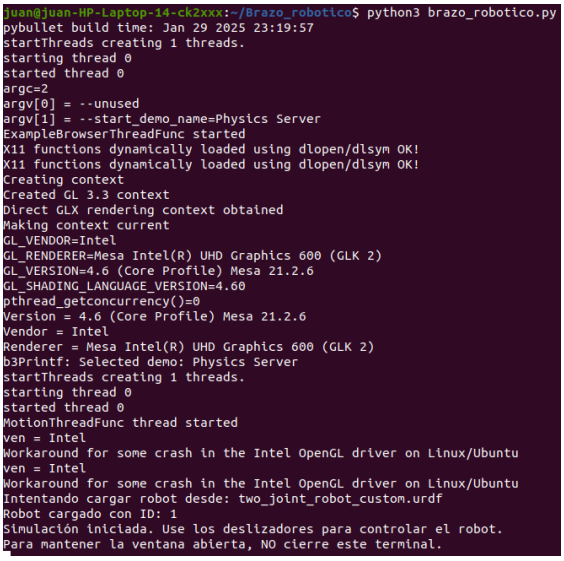


Figura 14: Diagnóstico en consola. Fuente: Elaboración propia.

Durante la operación del brazo robótico, se monitoreó la salida detallada de la consola, como se aprecia en la figura 14. Esta información proporciona el feedback de posición de cada articulación y el estado general del sistema, permitiendo un seguimiento preciso de su funcionamiento.

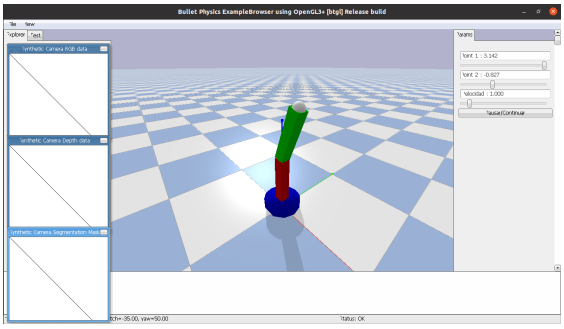


Figura 15: Interfaz gráfica del sistema. Fuente: Elaboración propia.

Finalmente, la figura 15 muestra la interfaz gráfica del brazo robótico funcionando dentro del contenedor Docker. Esta visualización confirma el correcto despliegue de toda la aplicación, incluyendo tanto los componentes de control como la representación visual del brazo en tiempo real.

### *B. Turtlebot3 con SLAM*

En esta sección se documenta la implementación completa de un sistema de navegación autónoma utilizando Turtlebot3 con algoritmos SLAM (Simultaneous Localization and Mapping) en un entorno containerizado con Docker. Se presentan todas las etapas desde la configuración inicial hasta las pruebas de navegación.

#### *1) Configuración del entorno Docker*

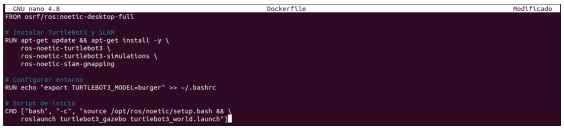


Figura 16: Código del Dockerfile. Fuente: Elaboración propia.

El proceso de implementación comenzó con la creación del Dockerfile, cuyo contenido se muestra en la figura 16. Este archivo contiene todas las instrucciones necesarias para construir la imagen Docker que soportará el sistema Turtlebot3, incluyendo la instalación de ROS, las dependencias del Turtlebot3 y los paquetes de SLAM.

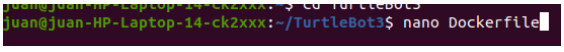


Figura 17: Creación del Dockerfile. Fuente: Elaboración propia.

Como se aprecia en la figura 17, el Dockerfile fue creado utilizando un editor de texto en la terminal. En este archivo

se especificaron las instrucciones para la configuración del entorno de ejecución del Turtlebot3, incluyendo la configuración de variables de entorno y la instalación de todos los paquetes necesarios.

Figura 18: Construcción de la imagen Docker. Fuente: Elaboración propia.

Una vez definido el Dockerfile, se procedió a la construcción de la imagen Docker mediante el comando 'docker build', como se muestra en la figura 18. Este proceso descarga y compila todos los componentes necesarios para ejecutar el sistema Turtlebot3 en un entorno aislado.

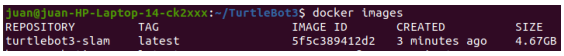


Figura 19: Verificación de la imagen. Fuente: Elaboración propia.

Para asegurar que la imagen se construyó correctamente, se verificó su existencia y estado mediante los comandos de Docker, como se evidencia en la figura 19. Esta comprobación confirma que la imagen está lista para ser utilizada en los siguientes pasos del proceso.

*2) Preparación del entorno de navegación*

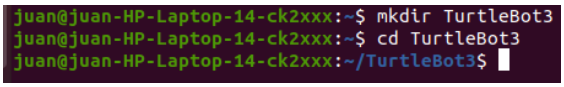


Figura 20: Creación del directorio de trabajo. Fuente: Elaboración propia.

Para organizar adecuadamente los archivos del proyecto, se creó una estructura de directorios como se muestra en la figura 20. Esta organización facilita la gestión de los diferentes componentes del sistema, separando los archivos de configuración, mapas y scripts.

Figura 21: Creación de los mapas. Fuente: Elaboración propia.

La figura 21 documenta el proceso de creación de los mapas que servirán para la navegación del Turtlebot3. Estos archivos son fundamentales para el algoritmo SLAM, permitiendo al robot localizar su posición y planificar rutas en el entorno.

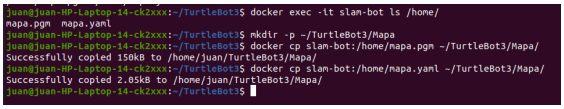


Figura 22: Verificación y copia de los mapas. Fuente: Elaboración propia.

Como parte del proceso de configuración, se realizó la verificación y copia de los archivos de mapas al directorio de trabajo, como se observa en la figura 22. Este paso garantiza que los mapas estén correctamente configurados y sean accesibles desde el contenedor Docker.

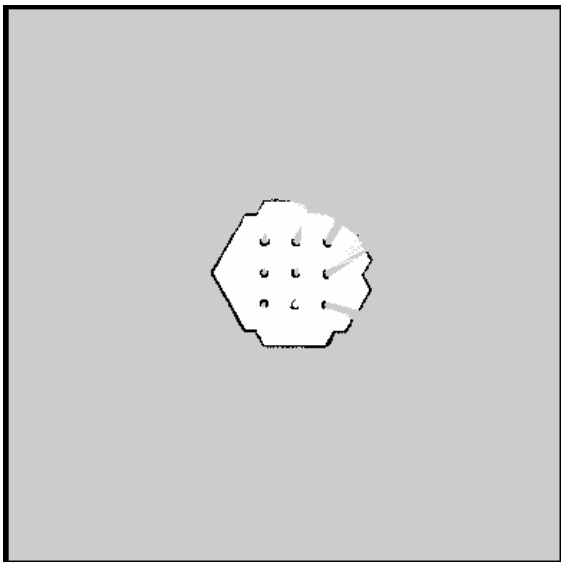

Figura 23: Mapa de tipo PGM. Fuente: Elaboración propia.

La figura 23 muestra una representación visual del mapa en formato PGM (Portable Gray Map). Este formato de mapa es comúnmente utilizado en robótica para representar entornos en dos dimensiones, donde los píxeles negros representan obstáculos, los blancos áreas libres y los grises zonas desconocidas o parcialmente obstaculizadas.

*3) Inicialización y pruebas del sistema*


Figura 24: Inicialización de Gazebo. Fuente: Elaboración propia.

La inicialización del simulador Gazebo, mostrada en la figura 24, representa el primer paso para la puesta en marcha del sistema de navegación. Gazebo proporciona un entorno virtual donde el Turtlebot3 puede operar, simulando condiciones físicas y sensores.

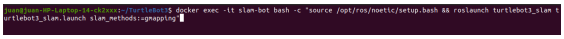


Figura 25: Inicio del proceso de mapeo. Fuente: Elaboración propia.

Una vez inicializado el simulador, se comenzó el proceso de mapeo como se documenta en la figura 25. Durante esta fase, el Turtlebot3 utiliza sus sensores para detectar el entorno circundante y construir un mapa del área, fundamental para la navegación autónoma.

*4) Pruebas de navegación y SLAM*

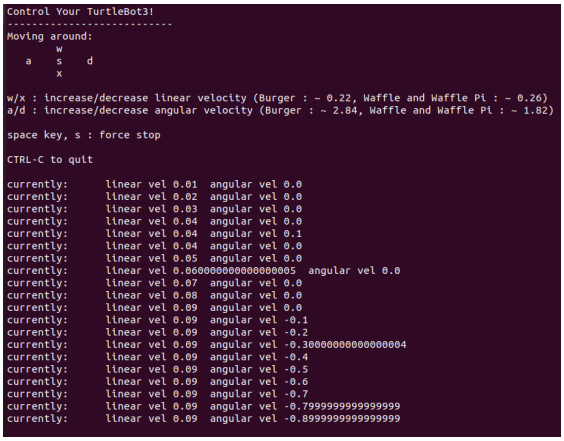


Figura 26: Registro de movimientos del robot. Fuente: Elaboración propia.

La figura 26 muestra el registro de los movimientos del robot durante las pruebas de navegación. Estos datos son fundamentales para evaluar el rendimiento del sistema de control y la precisión de los algoritmos de localización.

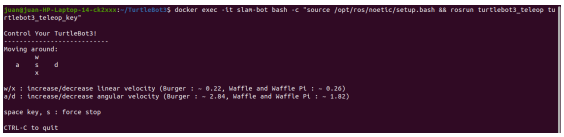


Figura 27: Ejecución de teleoperación. Fuente: Elaboración propia.

Para comprobar el correcto funcionamiento del Turtlebot3, se realizaron pruebas de teleoperación como se muestra en la figura 27. Esta funcionalidad permite controlar manualmente el robot, verificando la respuesta de los actuadores y la comunicación con el sistema ROS.

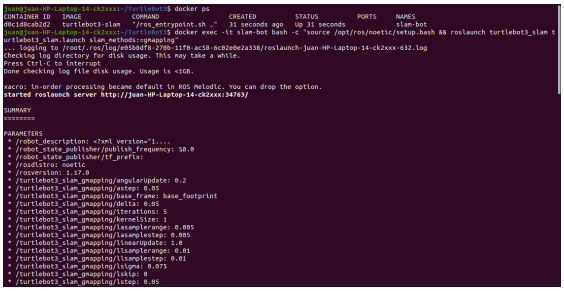


Figura 28: Ejecución del algoritmo SLAM. Fuente: Elaboración propia.

La ejecución del algoritmo SLAM, documentada en la figura 28, representa una parte crucial del proyecto. Este algoritmo permite al robot construir un mapa del entorno mientras se localiza simultáneamente dentro de él, facilitando la navegación autónoma.

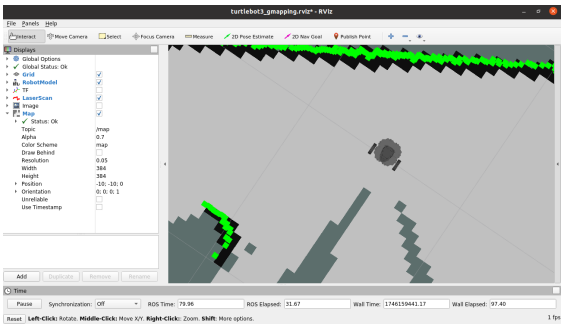


Figura 29: Visualización gráfica del SLAM. Fuente: Elaboración propia.

La figura 29 presenta la visualización gráfica del proceso SLAM en tiempo real. En esta imagen se puede apreciar cómo el robot va construyendo el mapa del entorno a medida que se desplaza, representando los obstáculos detectados y las áreas exploradas.

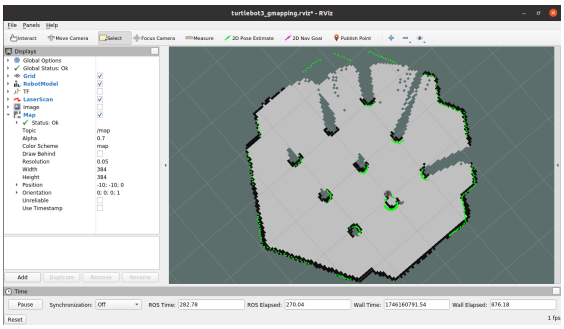

Figura 30: Registro de movimientos en SLAM. Fuente: Elaboración propia.

El registro de los movimientos del robot durante la ejecución del SLAM, mostrado en la figura 30, proporciona información detallada sobre la trayectoria seguida y las correcciones realizadas por el algoritmo. Estos datos son esenciales para evaluar la precisión del sistema de navegación.

*5) Verificación de funcionamiento en entorno Gazebo*

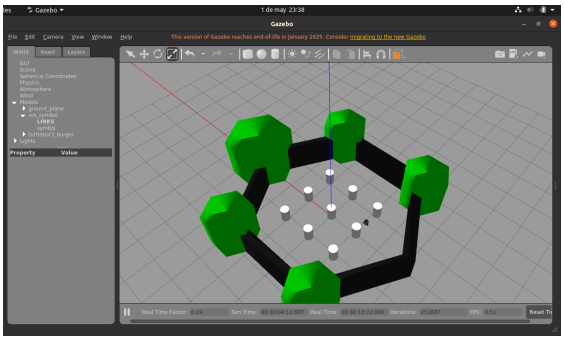

Figura 31: Visualización de movimientos en Gazebo. Fuente: Elaboración propia.

La visualización de los movimientos del Turtlebot3 en el entorno Gazebo, documentada en la figura 31, permite verificar que el robot se comporta según lo esperado en el mundo simulado. Se observan los obstáculos representados por esferas verdes y el robot navegando entre ellos.

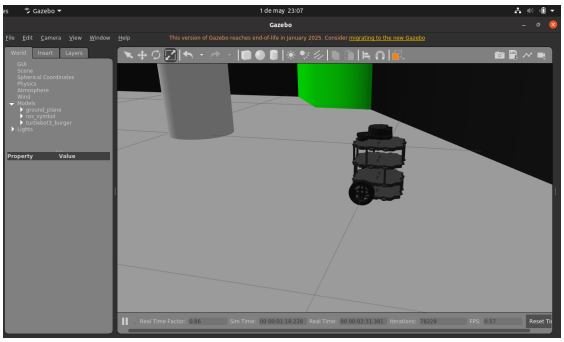

Figura 32: Prueba gráfica en Gazebo. Fuente: Elaboración propia.

Como parte final de la verificación, se realizó una prueba gráfica completa en Gazebo, mostrada en la figura 32. Esta prueba confirma que todo el sistema funciona correctamente, incluyendo la simulación física, los sensores virtuales y los algoritmos de control implementados.

*6) Verificación de actividad del sistema*

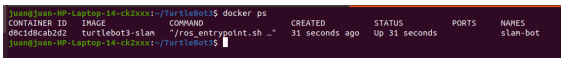


Figura 33: Verificación de actividad del contenedor. Fuente: Elaboración propia.

Finalmente, la figura 33 muestra la verificación de la actividad del contenedor Docker. Esta comprobación confirma que todos los servicios necesarios para el funcionamiento del Turtlebot3 están activos y ejecutándose correctamente dentro del entorno containerizado, lo que garantiza la operatividad del sistema completo.

## IV. CONCLUSIONES

Las interrupciones en sistemas operativos como Linux se comprendieron como mecanismos esenciales para gestionar eventos tanto internos como externos, permitiendo que el sistema responda de manera eficiente sin necesidad de recurrir a técnicas de espera activa. Su correcta implementación optimiza el uso de los recursos y mejora la estabilidad general del sistema.

Durante el estudio se identificaron tres tipos fundamentales de interrupciones: las de hardware, generadas por dispositivos físicos como teclados o temporizadores; las de software, utilizadas para hacer solicitudes al sistema operativo; y las excepciones, que corresponden a errores en la ejecución como divisiones por cero o accesos inválidos a memoria. Esta clasificación permite una gestión más precisa y controlada de los diferentes eventos que pueden ocurrir durante la operación del sistema.

Los manejadores de excepciones fueron analizados como rutinas fundamentales dentro del kernel, encargadas de capturar y resolver situaciones anómalas sin comprometer el funcionamiento del sistema. Su papel resulta clave para evitar fallos catastróficos y mantener la integridad del entorno operativo incluso ante errores inesperados.

Las interrupciones de software también se destacaron por su utilidad en la coordinación de tareas entre procesos y la solicitud de servicios al núcleo del sistema. A través de ellas, los programas pueden interactuar de forma estructurada con los recursos del sistema, manteniendo un control preciso del flujo de ejecución.

El algoritmo general de manejo de interrupciones en Linux fue abordado en detalle, comprendiendo que sigue un flujo bien definido que incluye la detención del proceso en curso, la ejecución del manejador correspondiente, la restauración del contexto y la reanudación del proceso original. Esta secuencia asegura una atención ordenada y eficaz de cada interrupción, sin generar inconsistencias.

En cuanto al análisis del artículo sobre bootkits, se entendió que estas amenazas han evolucionado desde ataques rudimentarios al sector de arranque (MBR) hasta sofisticadas técnicas que comprometen sistemas modernos basados en UEFI. Su persistencia, dificultad de detección y capacidad de ejecutarse en etapas tempranas del arranque los convierte en uno de los principales desafíos de la ciberseguridad actual.

Desde el componente práctico, se logró implementar una simulación funcional de un brazo robótico mediante Pybullet, encapsulado en un contenedor Docker. Esto permitió demostrar la versatilidad de las herramientas de contenerización para desarrollar, ejecutar y compartir proyectos de robótica de manera portátil y eficiente.

Asimismo, la simulación del TurtleBot3 con tecnologías como SLAM y sensores LIDAR permitió visualizar la construcción dinámica de mapas en tiempo real. Esta experiencia reforzó los conocimientos sobre navegación autónoma y localización simultánea, pilares fundamentales en la robótica moderna.

Otro aspecto destacado fue la documentación técnica empleando Markdown en el archivo README.md, lo que facilitó la presentación clara de los comandos, procedimientos y posibles mejoras. Esta práctica fomentó una mayor organización del proyecto y alineación con estándares actuales de trabajo colaborativo en plataformas como GitHub.

Finalmente, la integración de conocimientos en áreas como sistemas embebidos, ciberseguridad, programación y robótica permitió consolidar una visión interdisciplinaria. El trabajo no solo fortaleció conceptos teóricos, sino que también potenció habilidades prácticas, promoviendo un aprendizaje aplicado que responde a los retos tecnológicos contemporáneos.

## REFERENCIAS

[1] M. J. Järvenpää, ”Linux Interrupts: The Basic Concepts,”Montana State University. [En línea]. Disponible en: https://www.cs.montana.edu/courses/spring2005/518/Hypertextbook/jim/media/interrupts_on_linux.pdf
[2] E. Rodionov, A. Matrosov, D. Harley, ”BOOTKITS: PAST, PRESENT & FUTURE,”Virus Bulletin Conference, 2014. [En línea]. Disponible en: https://www.virusbulletin.com/uploads/pdf/conference/vb2014/VB2014-RodionovMatrosov.pdf
[3] O’Reilly, .Exception Handling in Linux Kernel.”[En línea]. Disponible en: https://www.oreilly.com/library/view/understanding-the-linux/0596005652/ch04s05.html
[4] ARM Developer, ”Sending and receiving Software Generated Interrupts.”[En línea]. Disponible en: https://developer.arm.com/documentation/198123/0302/Sending-and-receiving-Software-Generated-Interrupts
[5] Baeldung, İnterrupt Handling in Linux.”[En línea]. Disponible en: https://www.baeldung.com/linux/interrupt-handling
[6] Linux Kernel Documentation, ÏRQs in Linux.”[En línea]. Disponible en: https://www.kernel.org/doc/html/v5.10/core-api/irq/index.html
[7] Intel, ÇHIPSEC Framework for BIOS/UEFI Security Assessment.”[En línea]. Disponible en: https://github.com/chipsec/chipsec
[8] Pybullet Documentation, ”Bullet Real-Time Physics Simulation.”[En línea]. Disponible en: https://pybullet.org/
[9] Docker Docs, Çontainerizing Applications with Docker.”[En línea]. Disponible en: https://docs.docker.com/get-started/workshop/02_our_app/
[10] M. Zotovic-Stanisic et al., Çomparative Study of Methods for Robot Control with Flexible Joints,”MDPI, 2024. [En línea]. Disponible en: https://www.mdpi.com/2076-0825/13/8/299
[11] ROBOTIS e-Manual, ”SLAM in Turtlebot33.”[En línea]. Disponible en: https://emanual.robotis.com/docs/en/platform/Turtlebot33/slam/
[12] V. D. K. Chiem, İmplementing SLAM and Autonomous Navigation on a Turtlebot33,”Vanier College, 2024. [En línea]. Disponible en: https://s3.chiem.me/implementing-slam-and-autonomous-navigation-on-a-Turtlebot33.pdf
[13] GitHub Docs, ”Basic writing and formatting syntax.”[En línea]. Disponible en: https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax
[14] Document360, İntroductory Guide to Markdown for Documentation Writers.”[En línea]. Disponible en: https://document360.com/blog/introductory-guide-to-markdown-for-documentation-writers/
[15] A. Matrosov, E. Rodionov, S. Bratus, Rootkits and Bootkits: Reversing Modern Malware and Next Generation Threats,”No Starch Press, 2019.