Universidad Politécnica de Madrid

Robots autónomos

Práctica A2: Construcción de mapas

Autores: Juan Esteban Rincón Marín
Arturo Soto Ruedas
Jesús Samuel García Carballo

14 de diciembre de 2025

1. Introducción

El objetivo principal de esta práctica es el desarrollo de un sistema de control para un robot móvil capaz de explorar un entorno desconocido y generar un mapa de ocupación.

Para lograr esto, hemos diseñado una arquitectura híbrida que desacopla la simulación del control:

1. **Simulación:** *CoppeliaSim* para la física, el entorno y los sensores.
2. **Control y SLAM:** *ROS 2 Humble* ejecutándose en un contenedor Docker, gestionando la lógica de navegación y el mapeo.

El sistema implementa un algoritmo de navegación reactiva (**Bug2**), extraído de la práctica anterior, para garantizar que el robot pueda explorar el entorno sin quedar atrapado. Por su parte, el paquete `slam_toolbox` se encarga de construir el mapa de ocupación en tiempo real utilizando los datos del LIDAR y la odometría del robot

2. Arquitectura del Sistema

El sistema se divide en dos bloques principales comunicados por (*localhost*):

- **CoppeliaSim (Local):** Ejecuta la escena, el modelo del robot y gestiona los sensores físicos (Sonares + Hokuyo LaserScan). Se mantiene en el host local para asegurar la estabilidad gráfica y facilitar la depuración visual.
- **ROS 2 Humble (Docker):** Ejecuta la lógica "inteligente" del robot, incluyendo el nodo de interfaz con Coppelia, el controlador Bug2 y el sistema SLAM. Se encapsula en Docker para garantizar la reproducibilidad y aislar las dependencias del sistema host.

2.1. Paquetes de ROS 2

Para el desarrollo de la práctica se han utilizado una serie de paquetes:

slam_toolbox: Es el núcleo del sistema de mapeo. Se utiliza el nodo `async_slam_toolbox_node`.

- Consume: `/scan` (LaserScan) y odometría.
- Publica: `/map` (OccupancyGrid) y transformación `map` → `odom`.

nav2_map_server: Utilizado exclusivamente para la persistencia del mapa. Mediante la herramienta `map_saver_cli` se generan los ficheros `.pgm` y `.yaml` requeridos.

tf2_ros: Gestiona el árbol de transformaciones, incluyendo un `static_transform_publisher` para definir la posición fija del láser respecto a la base del robot (`base_link` → `laser`).

entrega_mapas_package: Paquete propio desarrollado por el grupo que contiene los nodos de interfaz, control Bug2 y gestión de metas.

3. Diseño de nodos

La arquitectura del sistema se ha diseñado de forma modular, dividiendo la responsabilidad en cuatro nodos especializados configurados mediante el script `bug2_system.launch.py`. A continuación se detalla la lógica interna de cada uno:

3.1. Interfaz (`coppelia_interface_node`)

Este nodo actúa como capa de abstracción de hardware, aislando la simulación del resto del sistema ROS 2.

- **Frecuencia de Actualización:** Se ha establecido un ciclo de control de **20 Hz**. Esta tasa se eligió como compromiso técnico: es suficiente para capturar la dinámica del robot Pioneer P3DX sin saturar el canal de comunicación ZMQ con el simulador.
- **Odometría (Ground Truth):** El nodo extrae la posición absoluta del simulador para calcular una odometría libre de deriva ($\Delta x/\Delta t$), proporcionando una localización perfecta para el mapeo.
- **Calibración Extrínseca:** Mediante `tf2_ros`, se publica una transformación estática del láser a $(0{,}2, 0{,}0, 0{,}2)\,m$ respecto a la base, correspondiente a la ubicación física del sensor Hokuyo.

3.2. Mapeo y localización (`slam_toolbox`)

Para la construcción del mapa de ocupación se utiliza el paquete estándar `slam_toolbox` en modo asíncrono (`async_slam_toolbox_node`).

- **Técnica SLAM:** Implementa *Pose Graph SLAM* (optimización de grafos) con *Scan Matching*. Esto permite corregir pequeños errores de localización alineando las lecturas actuales del láser con el mapa construido hasta el momento.
- **Modo Asíncrono:** Se ha seleccionado este modo para desacoplar el procesamiento del mapa del bucle de odometría, asegurando que el robot nunca pierda su localización local (`odom` $\rightarrow$ `base_link`) incluso si la optimización global del mapa consume mucha CPU.

3.3. Control de navegación (`bug2_controller_node`)

El núcleo de navegación implementa una Máquina de Estados Finitos (FSM) basada en el algoritmo Bug2, ajustada experimentalmente para garantizar la seguridad.

3.3.1. Estados y Parámetros Dinámicos

1. **MOTION_TO_GOAL:** El robot avanza hacia la meta con un control proporcional.
   - *Velocidad:* Se permite una velocidad lineal máxima de **4.0 m/s** en tramos rectos para agilizar la exploración.
   - *Transición:* Si se detecta un obstáculo a $d < 0{,}6\,m$, se activa el modo de evasión.

2. **WALL_FOLLOWING:** El robot bordea el obstáculo. Para este estado crítico, se han aplicado restricciones conservadoras en el *launch file*:
    - **Velocidad Reducida:** Se limita a **1.0 m/s** para dar tiempo de reacción a los sonares en las esquinas.
    - **Distancia de Referencia:** Se fija en **0.75 m**. Este valor, superior al umbral de colisión, compensa el ruido de los ultrasonidos y evita roces laterales.

3.3.2. Criterio de salida

El robot abandona el muro solo si cumple la condición lógica:

$$\text{Salida} = (\text{En M-Line}) \wedge (\text{Progreso Real}) \wedge (\text{Camino Libre}) \tag{1}$$

Donde:

- **En M-Line:** Debe estar en la M-Line, definida como la línea recta entre la posición inicial y la meta.
- **Progreso Real:** La distancia actual a la meta es estrictamente menor que el *hit_point*.
- **Camino Libre:** El camino directo a la meta debe estar libre.

3.4. Gestión de metas (`goal_manager_node`)

Para automatizar la exploración, este nodo genera metas aleatorias dentro de las coordenadas operativas ($X \in [-7{,}4, 2{,}45]$, $Y \in [-2{,}45, 2{,}45]$).

- **Buffer de seguridad:** Se aplica un margen de **0.2 m** respecto a los bordes del mapa. Esto evita generar metas inalcanzables pegadas a los muros exteriores.
- **Robustez (QoS):** Las metas se publican con política *Transient Local*, asegurando que el controlador reciba la última misión válida inmediatamente tras cualquier reinicio de la red.

4. Arquitectura de comunicaciones

Para visualizar el flujo de datos entre los distintos módulos Python y los paquetes externos, se presenta el siguiente diagrama de grafo de tópicos.

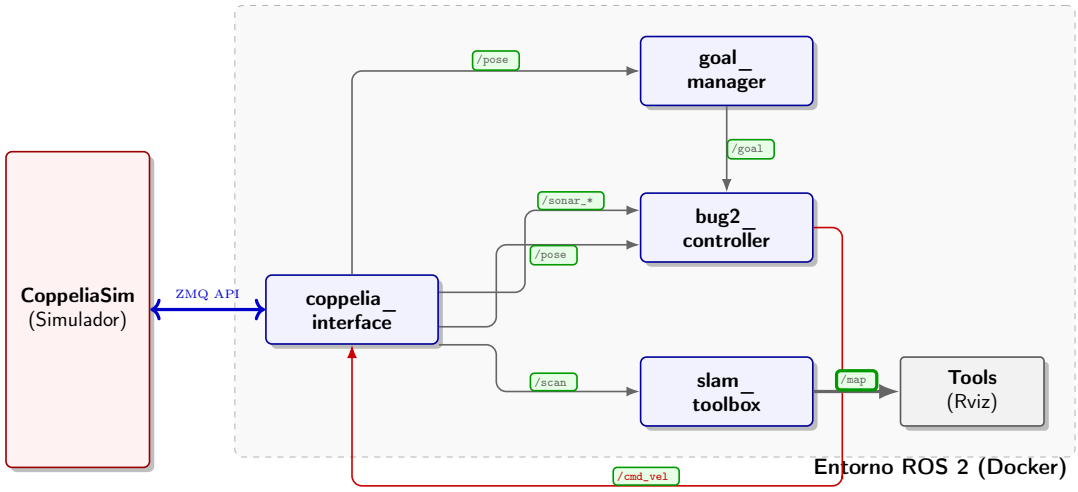

Figura 1: Grafo de comunicaciones del sistema. Se observa el flujo de datos desde la simulación (izquierda) y el lazo de control externo (línea roja).

5. Diseño del entorno de simulación

Para la validación experimental del sistema, se ha trabajado sobre el entorno de simulación propuesto en el enunciado de la práctica. A continuación se detallan las características del escenario y las modificaciones realizadas para garantizar la fidelidad sensorial.

5.1. Recreación del entorno

Hemos recreado en coppeliaSim la escena `mapa_practica2.ttt`, la cual replica un entorno con las siguientes características:

- **Estructura:** Un recinto cerrado de 10 x 20 compuesto por múltiples estancias conectadas mediante pasillos y puertas.
- **Obstáculos:** Se incluyen columnas centrales y barreras que obligan al robot a maniobrar, evitando líneas de visión directas y forzando la exploración activa.

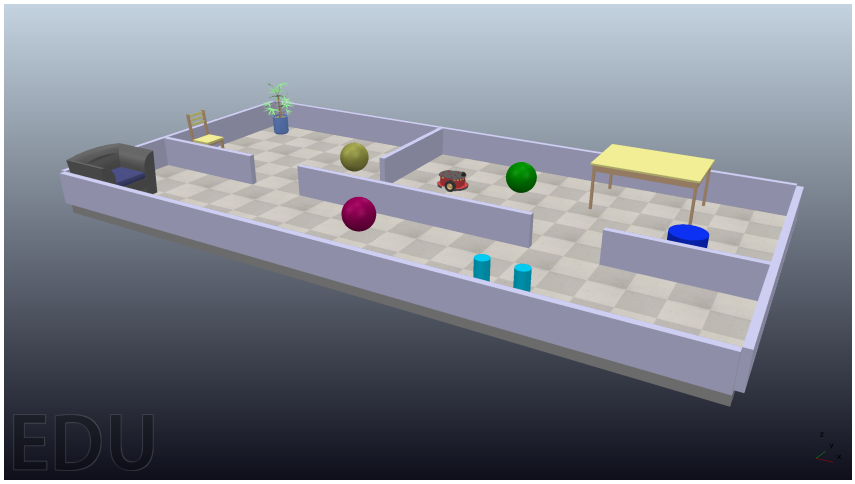

Figura 2: Escenario de simulación en CoppeliaSim

5.2. Sensores

Con respecto a la práctica anterior, se han realizado ajustes específicos, como la integración del sensor LIDAR Hokuyo. Además fue necesario realizar una modificación en las propiedades de los objetos de la escena (muros y obstáculos estáticos) para asegurar su detección mediante ultrasonidos.

Se activó explícitamente el flag *detectable* en las propiedades dinámicas de los volúmenes de la escena. Sin este ajuste, el motor físico de CoppeliaSim no generaba los rebotes necesarios para los sensores de rango (`sensor_msgs/Range`), lo que provocaba que el robot "viese" a través de las paredes.

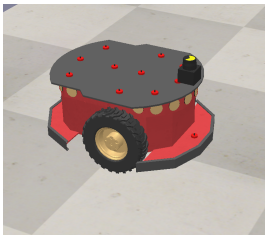

Figura 3: Modelo del robot Pioneer P3DX con sensores integrados, incluyendo el LIDAR Hokuyo y los sonares.

6. Ejecución del sistema

6.1. Despliegue con Docker

Para garantizar la reproducibilidad y evitar problemas de dependencias, se utiliza Docker. Comandos básicos:

```
# Construccion de la imagen
docker compose build

# Ejecucion en segundo plano
docker compose up -d

# Acceso al entorno de desarrollo
docker compose exec servicio_ros bash
```

Listing 1: Comandos de Docker

6.2. Compilación y Ejecución

Dentro del contenedor, el flujo de trabajo es el estándar de ROS 2:

```
# Compilacion
colcon build --symlink-install
source install/setup.bash

# Lanzamiento del sistema completo
```

```
ros2 launch entrega_mapas_package sistema_completo.launch.py
```

Listing 2: Compilación y Ejecución

6.3. Interfaces de Comunicación (Topics)

El contrato de comunicación del sistema se define en la Tabla 1.

Topic	Tipo de Mensaje	Descripción
/cmd_vel	geometry_msgs/Twist	Comando de velocidad al robot.
/scan	sensor_msgs/LaserScan	Datos del LIDAR (Input SLAM).
/map	nav_msgs/OccupancyGrid	Mapa generado por SLAM.
/robot/sonar_*	sensor_msgs/Range	Lecturas de ultrasonidos.
/goal	geometry_msgs/PoseStamped	Meta actual de exploración.

Cuadro 1: Topics principales del sistema

7. Generación del mapa

7.1. Proceso de mapeado

Durante la ejecución, el nodo `async_slam_toolbox_node` fusiona la información de odometría con las lecturas del láser para expandir el mapa y corregir la deriva, mientras el robot recorre el entorno guiado por el algoritmo Bug2.

7.2. Guardado del mapa

Una vez completada la exploración, se ha utilizado `nav2_map_server` para exportar el resultado. Es crucial configurar la QoS correctamente (`transient_local`) si el topic no es volátil:

```
ros2 run nav2_map_server map_saver_cli -f /ros2_ws/maps/
   mapa_practica --ros-args -p map_subscribe_transient_local:=true
```

Esto genera los archivos `mapa_practica.pgm` y `mapa_practica.yaml` adjuntos en la entrega.

8. Resultado

El resultado principal de la práctica es el mapa de ocupación generado tras la exploración completa del escenario. A continuación se presenta el mapa final obtenido y se analiza su fidelidad respecto al entorno de simulación original.

8.1. Mapa de ocupación

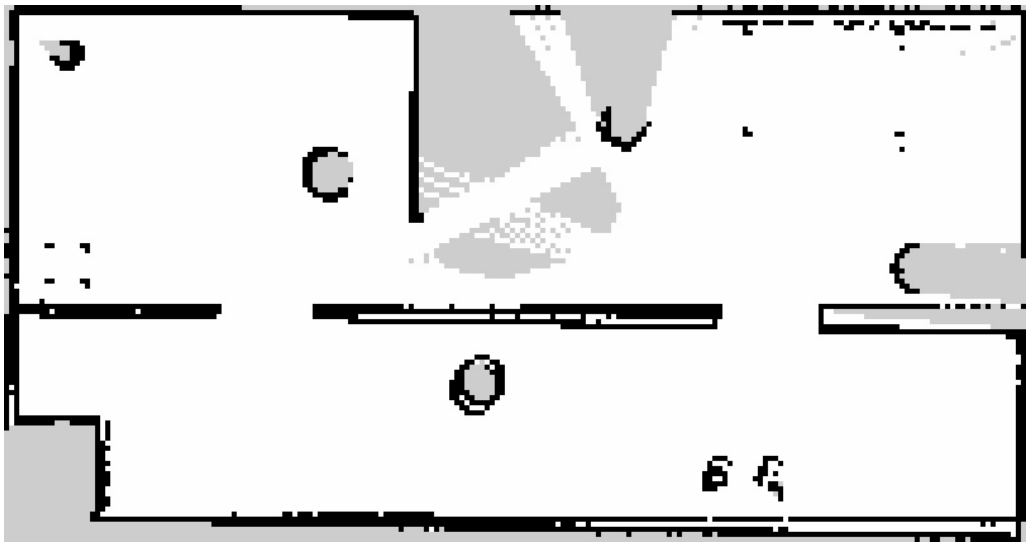

Figura 4: Comparativa: Mapa de ocupación generado por SLAM Toolbox (Izquierda) frente al esquema del escenario en CoppeliaSim (Derecha).

8.2. Análisis

Al contrastar el mapa generado (Figura 4) con el entorno real de CoppeliaSim, se observan los siguientes puntos clave:

1. **Fidelidad geométrica:** Las paredes y esquinas presentan una rectitud notable, sin el efecto de "doble muro" o curvatura característico de los errores de deriva angular. Esto valida la eficacia de la interfaz de odometría implementada en el nodo `coppelia_interface_node`.
2. **Consistencia topológica:** El algoritmo `slam_toolbox` ha cerrado correctamente los bucles al revisitar zonas previamente exploradas (como el pasillo central). No se aprecian discontinuidades en el mapa, lo que indica que el módulo de *Scan Matching* y la optimización del grafo han funcionado correctamente con los parámetros de resolución de 0.05 m.
3. **Capacidad explorativa:** Gracias a la estrategia de generación de metas aleatorias del `goal_manager` dentro de las cotas, se han mapeado incluso las zonas de sombra detrás de las columnas centrales, eliminando las áreas grises (espacio desconocido) dentro del perímetro operativo.

9. Análisis crítico

10. Conclusiones

La realización de esta práctica ha permitido validar con éxito una arquitectura robótica híbrida donde el desacoplamiento entre la simulación en CoppeliaSim y el control contenerizado en Docker ha sido clave para la estabilidad del sistema. Gracias a la implementación

de una odometría basada en *Ground Truth* y su fusión con un algoritmo de *Graph SLAM* (`slam_toolbox`), se ha logrado mitigar la deriva angular típica de la navegación, generando un mapa de ocupación de alta fidelidad geométrica. Asimismo, la robustez del controlador reactivo Bug2, ha garantizado una cobertura completa del entorno sin atrapamientos en mínimos locales, entregando un mapa con una resolución de 5 cm que satisface plenamente los requisitos operativos para las futuras tareas de planificación de camino.