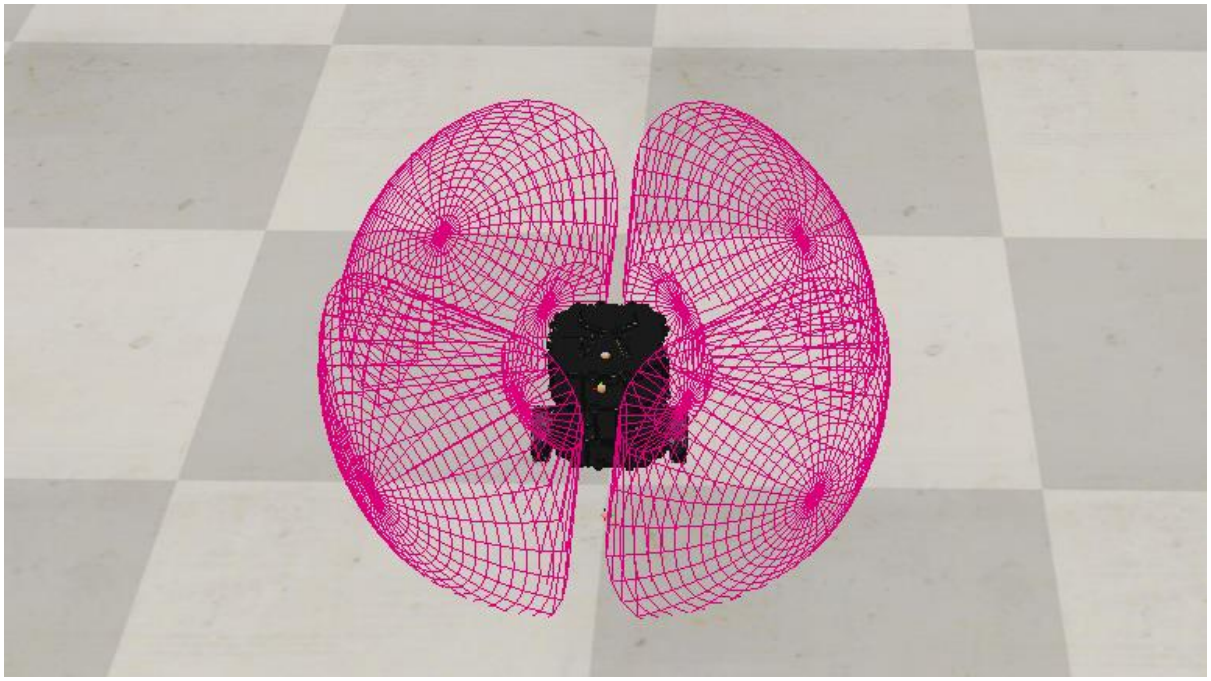
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# Grado en Informática Industrial y Robótica

## Robótica móvil

## *Navegación Autónoma con TurtleBot3 en Entorno Simulado con ROS 2 y CoppeliaSim*



**Autores del proyecto:**
José Luis Galán, Nerea Ros, Gonzalo Albelda y Gonzalo Martín.

**Profesor responsable:**
Leopoldo Armesto Ángel.

# Índice:

# Descripción Técnica del Proyecto

## Arquitectura General

El sistema desarrollado se basa en una arquitectura modular distribuida sobre ROS2 (Robot Operating System 2), diseñada específicamente para controlar el movimiento de un robot móvil en un entorno simulado mediante **CoppeliaSim**. El objetivo principal es lograr un **seguimiento de trayectorias a través de waypoints**, evitando colisiones con obstáculos en tiempo real mediante sensores de ultrasonidos.

La arquitectura se organiza en torno a varios **nodos ROS2 independientes**, cada uno con responsabilidades específicas, permitiendo una separación clara entre la percepción, la planificación y el control. Esta separación facilita el desarrollo, la depuración y la escalabilidad del sistema.

- **waypoint_node**: se encarga de cargar desde un archivo .yaml la secuencia de puntos de la trayectoria (waypoints) y los publica como un mensaje personalizado WayPointPath. Esto permite definir caminos personalizados sin modificar el código fuente.
- **purepursuit_node**: implementa el algoritmo de seguimiento Pure Pursuit. Calcula las referencias de movimiento (velocidad lineal y angular) necesarias para alcanzar el siguiente punto del camino, basándose en la posición y orientación del robot recibidas por odometría.
- **obstacle_avoider_node**: monitoriza continuamente los datos de los sensores de ultrasonidos. Si detecta un obstáculo en el camino, modifica las referencias de movimiento generadas por el purepursuit_node para evitar la colisión. Esta lógica de evasión incluye rotaciones y pequeños desplazamientos incrementales.

Los **sensores de ultrasonidos** son simulados mediante scripts en CoppeliaSim y **no están implementados como nodos ROS2 independientes**, sino que publican directamente las lecturas en tópicos ROS2 gracias a la integración del simulador con el middleware. Estos datos son consumidos por el obstacle_avoider_node.

Todos los nodos se comunican a través del sistema de publicación/suscripción de ROS2, con mensajes estándar (geometry_msgs/Twist, geometry_msgs/Point, etc.) y uno personalizado (project_rom/msg/WayPointPath).

La arquitectura modular y desacoplada permite realizar pruebas de cada componente de forma independiente y facilita su ampliación futura con nuevos nodos, sensores o lógicas de comportamiento.

El robot opera en una escena simulada de CoppeliaSim, configurada específicamente con sensores y entorno de obstáculos. Esta integración permite realizar pruebas realistas del sistema de navegación antes de una implementación física.

## Componentes del Sistema

El sistema completo se compone de diversos elementos de hardware simulado y software, organizados para lograr una navegación autónoma basada en waypoints con capacidad de evasión de obstáculos. A continuación, se describen los componentes principales:

### Software

- **ROS2 Humble**
  Es el middleware principal sobre el que se ejecutan todos los nodos del sistema. Permite la comunicación entre módulos y la gestión de mensajes personalizados y estándar.
- **CoppeliaSim**
  Simulador 3D donde se implementa el entorno virtual, el robot móvil y los sensores ultrasónicos. Utiliza un script embebido para publicar los datos de los sensores directamente en ROS2.
- **Nodos ROS2 del sistema**:
  - **waypoint_node:** carga un archivo .yaml con una lista de puntos objetivo (waypoints) y los publica como un mensaje personalizado.
  - **purepursuit_node:** implementa el algoritmo de seguimiento de trayectoria Pure Pursuit, calculando las referencias de velocidad en base a los waypoints y la odometría.
  - **obstacle_avoider_node:** procesa las lecturas de los sensores de ultrasonides para modificar las referencias cuando hay peligro de colisión.
- **Mensaje personalizado WayPointPath**
  Define una lista de puntos (geometry_msgs/Point) que representa la trayectoria del robot. Es utilizado para transmitir la ruta entre nodos.
- **Configuración y visualización**:
  - Archivos .yaml en config/ para definir rutas.
  - Archivos .ttt de escena para simular el entorno en CoppeliaSim.
  - RViz para la visualización del entorno, sensores, odometría y trayectorias.

## Hardware Simulado

- **Robot móvil (turtlebot)**
  Representado en CoppeliaSim, con un modelo que incluye control de movimiento diferencial y sensores ultrasónicos virtuales. Publica su odometría y recibe comandos de velocidad desde los nodos ROS2.
- **Sensores de ultrasonidos**
  Cuatro sensores distribuidos alrededor del robot (delanteros y traseros, izquierda y derecha), programados en un script LUA dentro de CoppeliaSim. Estos sensores publican las distancias medidas en ROS2 para ser utilizadas por el nodo de evasión.
- **Entorno de pruebas**
  Simulado en CoppeliaSim, con obstáculos y paredes diseñadas para validar la capacidad del sistema de navegación y evasión en condiciones complejas.


## Integración entre Componentes

El sistema ha sido diseñado de forma modular, permitiendo que cada componente pueda desarrollarse, probarse y ejecutarse de manera independiente. Sin embargo, para lograr una operación completa y coordinada del robot, es necesaria una integración eficaz de todos los módulos. Esta integración se lleva a cabo principalmente mediante un archivo de lanzamiento de ROS2 (launch).

## Coordinación mediante archivo launch

El archivo scene_launch.py, ubicado en la carpeta launch/, permite **inicializar todos los elementos clave del sistema con un único comando**. Este archivo orquesta la ejecución de los siguientes componentes:

- **Simulación en CoppeliaSim**, cargando automáticamente la escena del entorno desde el archivo .ttt.
- **Publicación de datos de sensores de ultrasonidos** desde el script LUA de CoppeliaSim hacia ROS2.
- **Nodo waypoint_node**, que carga los waypoints desde un archivo .yaml ubicado en la carpeta config/.
- **Nodo purepursuit_node**, que recibe la ruta y la odometría del robot, calculando las referencias de movimiento.
- **Nodo obstacle_avoider_node**, que se encarga de interceptar las referencias y ajustarlas si detecta obstáculos cercanos.
- **Visualización con RViz**, permitiendo observar la odometría del robot, las trayectorias planificadas y las detecciones de los sensores en tiempo real.

Este archivo launch también permite configurar algunos parámetros clave mediante LaunchConfiguration, como la ruta del archivo de escena (scene_dir) o el uso del tiempo simulado (use_sim_time).

## Flujo de integración entre nodos

Una vez lanzado el sistema, el flujo de datos es el siguiente:

1. **waypoint_node** publica un mensaje WayPointPath con la ruta deseada.
2. **purepursuit_node** lee este mensaje y, junto con la odometría del robot, calcula las referencias de movimiento (geometry_msgs/Twist).
3. **obstacle_avoider_node** evalúa las lecturas de los sensores de ultrasonidos y decide si aplicar la referencia del Pure Pursuit o modificarla para evitar colisiones.
4. Finalmente, la referencia se envía al robot en la simulación a través del tópico correspondiente (/cmd_vel o similar).

La modularidad del sistema permite **añadir nuevos nodos** (por ejemplo, para mapeo o navegación global) sin alterar la estructura base.

## Descripción de los nodos y su funcionamiento

El sistema cuenta con tres nodos ROS2 principales que trabajan de forma conjunta para permitir el seguimiento de trayectorias y la evasión de obstáculos. A continuación, se detalla la lógica interna de cada uno:

### waypoint_node

Se encarga de cargar la trayectoria desde un archivo .yaml y publicarla como mensaje personalizado para que otros nodos puedan utilizarla.

- **Suscripciones:** ninguna.
- **Publicaciones:**
  - waypoint_path (project_rom/msg/WayPointPath): lista de puntos (geometry_msgs/Point) representando la trayectoria deseada.
- **Funcionamiento:**
  - Lee el archivo .yaml de configuración con los puntos de la trayectoria.
  - Convierte cada par (x, y) en un geometry_msgs/Point.
  - Publica todos los puntos juntos como un único mensaje WayPointPath.

### purepursuit_node

Implementa el algoritmo de seguimiento Pure Pursuit y envía directamente los comandos de velocidad al robot. Además, ajusta las referencias de movimiento si recibe una curvatura de evasión desde el nodo de obstáculos.

- **Suscripciones:**
  - odom (nav_msgs/Odometry): posición y orientación del robot.
  - waypoint_path (project_rom/msg/WayPointPath): trayectoria objetivo.
  - avoidance_curvature (std_msgs/Float64): curvatura alternativa calculada por el nodo de evitación de obstáculos.
- **Publicaciones:**
  - cmd_vel (geometry_msgs/Twist): comandos finales de movimiento para el robot.

- **Funcionamiento:**
  - Recibe la posición actual del robot y la trayectoria a seguir.
  - Determina el siguiente punto objetivo siguiendo el orden definido.
  - Calcula la curvatura necesaria con Pure Pursuit.

- Si se recibe una curvatura alternativa desde obstacle_avoider_node, modifica su referencia para adaptarse.
- Publica el comando final de velocidad a /cmd_vel
- Aplica la lógica del algoritmo Pure Pursuit para generar una referencia de velocidad proporcional.
- Publica la referencia para que sea interpretada por el nodo de evasión o directamente por el controlador.

### obstacle_avoider_node

Calcula una curvatura de evasión en tiempo real según la proximidad y posición de los obstáculos detectados por los sensores ultrasónicos.

- **Suscripciones:**
  - sonar0 a sonar3 (sensor_msgs/Range): datos de los sensores de ultrasonidos de CoppeliaSim.
- **Publicaciones:**
  - avoidance_curvature (std_msgs/Float64): curvatura calculada para esquivar obstáculos.
- **Funcionamiento:**
  - Monitoriza continuamente las lecturas de los sensores frontales y laterales.
  - Determina si hay riesgo de colisión.
  - Si detecta un obstáculo, calcula una curvatura adecuada (hacia derecha o izquierda).
  - Publica esta curvatura como recomendación para modificar el movimiento del robot.
  - Si no hay obstáculos, no publica o publica 0.

## Comunicación ROS 2

El sistema está construido sobre el modelo de comunicación **publisher/subscriber** de ROS2, lo que permite una interacción modular y desacoplada entre nodos. Cada nodo intercambia información mediante tópicos específicos, utilizando tanto mensajes estándar como personalizados. A continuación, se detallan los principales canales de comunicación del sistema:

| Tópico | Tipo de mensaje | Publicador | Suscriptor(es) | Descripción |
|---|---|---|---|---|
| **/waypoint_path** | project_rom/msg/WayPointPath | waypoint_node | purepursuit_node | Transmite la trayectoria completa como lista de puntos. |
| **/odom** | nav_msgs/Odometry | (simulación / robot) | purepursuit_node | Proporciona la posición y orientación del robot. |
| **/avoidance_curvature** | std_msgs/Float64 | obstacle_avoider_node | purepursuit_node | Curvatura de evasión calculada a partir de los sensores ultrasónicos. |
| **/cmd_vel** | geometry_msgs/Twist | purepursuit_node | (robot / simulación) | Comando final de movimiento que ejecuta el robot. |
| **/sonar0 - /sonar3** | sensor_msgs/Range | (script en CoppeliaSim) | obstacle_avoider_node | Lecturas de distancia desde los sensores ultrasónicos virtuales. |

## Mensajes utilizados

- **Mensajes estándar:**
  - geometry_msgs/Point: puntos individuales de la trayectoria.
  - geometry_msgs/Twist: velocidad lineal y angular del robot.
  - nav_msgs/Odometry: posición y orientación del robot.
  - sensor_msgs/Range: distancia detectada por sensores ultrasónicos.
  - std_msgs/Float64: valor de curvatura para evasión.
- **Mensaje personalizado:**
  - project_rom/msg/WayPointPath: contiene un array de geometry_msgs/Point que define el camino a seguir.

## Flujo de información

1. El nodo waypoint_node publica la ruta objetivo al tópico /waypoint_path.
2. purepursuit_node se suscribe a esta ruta y, junto con la odometría del robot, calcula una referencia de movimiento.
3. Paralelamente, el nodo obstacle_avoider_node procesa los datos de los sensores ultrasónicos publicados desde CoppeliaSim.
4. Si detecta un obstáculo, obstacle_avoider_node publica una curvatura de evasión en /avoidance_curvature.
5. purepursuit_node adapta su referencia con esta curvatura, y envía el comando final a /cmd_vel.

Este modelo de comunicación desacoplada permite sustituir o modificar módulos de forma sencilla sin afectar al resto del sistema, facilitando su mantenimiento y escalabilidad.

# Requisitos del Sistema

Para ejecutar correctamente el proyecto project_rom, es necesario disponer de un entorno compatible con ROS2, simulación 3D con CoppeliaSim y las herramientas necesarias para compilar y visualizar el sistema.

| Software | Versión recomendada | Descripción |
|---|---|---|
| **ROS2** | Humble Hawksbill | Middleware principal, compatible con Python y C++. |
| **CoppeliaSim** | 4.6 o superior | Simulador 3D con integración ROS2 vía ZeroMQ o plugin ROS. |
| **Python** | 3.10+ | Para ejecución de launch files y utilidades. |
| **CMake** | 3.16 o superior | Herramienta de construcción del paquete. |
| **colcon** | Última estable | Para compilar el workspace de ROS2. |
| **RViz2** | Incluido en ROS2 | Para visualización de odometría, trayectorias y sensores. |

**Dependencias adicionales:**

- geometry_msgs, nav_msgs, sensor_msgs, std_msgs: mensajes estándar.
- rclcpp, rclpy: librerías ROS2 para C++ y Python.
- Plugin de integración entre CoppeliaSim y ROS2 (normalmente vía script LUA o simExtROS2).

# Instrucciones de Instalación

*Esta guía parte de la base de que el usuario ya tiene instalado Ubuntu 22.04, ROS2 Humble y CoppeliaSim configurado para su uso con ROS2.*

## 1. Descargar el proyecto

Descargar el archivo comprimido del proyecto:

```
wget <enlace_al_zip> -O project_rom.zip
```

*(Reemplazar <enlace_al_zip> por el enlace real del repositorio o plataforma de distribución)*

## 2. Extraer dentro del workspace

Extraer el contenido directamente en la carpeta src/ del workspace ROS2 (en este caso RM_prac):

```
cd ~/RM_prac/src
unzip ~/Descargas/project_rom.zip
```

*Debe crearse una carpeta project_rom/ con todo el código fuente y estructura del paquete.*

## 3. Compilar el proyecto

Volver a la raíz del workspace y compilar:

```
cd ~/RM_prac
colcon build --packages-select project_rom
unzip ~/Descargas/project_rom.zip
```

Fuente del entorno:

```
source install/setup.bash
```

Si hay errores por dependencias faltantes, asegúrate de haber instalado previamente los paquetes estándar de mensajes (ros-humble-geometry-msgs, ros-humble-nav-msgs, etc.).

# Ejecución del Programa

Una vez compilado correctamente el paquete, puedes ejecutar el sistema completo utilizando el archivo de lanzamiento principal.

## 1. Fuente del entorno

Antes de ejecutar cualquier comando, asegúrate de tener el entorno configurado:


## 2. Lanzar el sistema

Ejecuta el siguiente comando para lanzar:


Esto lanzará automáticamente:

- La escena sonar_scene.ttt en **CoppeliaSim** con el entorno y robot.
- Los tres nodos principales: waypoint_node, purepursuit_node, y obstacle_avoider_node.
- La carga de la trayectoria definida en config/waypoints.yaml.
- La visualización en **RViz**, si está incluida en el launch.

*El robot comenzará a seguir la trayectoria definida, adaptando su movimiento si detecta obstáculos en su camino mediante los sensores de ultrasonidos.*