Sistema de Navegación Autónoma con ROS2 y Turtlebot3

Jefferson Paul Caiza Jami

Pablo Granell Robles

1. Objetivos Principales

El trabajo se estructura en tres tareas que crean un sistema de navegación autónomo completo:

1. **Mapeo del entorno (Tarea 0):** Crear un mapa de la casa simulada utilizando SLAM
2. **Servicio de comandos de navegación (Tarea 1):** Desarrollar un servicio ROS2 que permita comandar al robot a que haga patrullaje por la casa o que vuelva a la salida
3. **Búsqueda del tesoro con patrullaje (Tarea 2):** Implementar un comportamiento autónomo de búsqueda y navegación hacia un "tesoro" basado en la información recibida

2. Estructura del Sistema

2.1 Organización del Proyecto

```
Entregable-ARO/
|- src/                                 # Código fuente
|   |- servicio_comandos/               # Paquete Tarea 1
|   |   |- servicio_comandos/
|   |       |- servidor_comandos.py
|   |       |- cliente_comandos.py
|   |- busqueda_tesoro/                 # Paquete Tarea 2
|   |   |- busqueda_tesoro/
|   |   |   |- busqueda_nodo.py
|   |   |- config/
|   |       |- turtlebot3_params.yaml   # Parámetros del robot
|   |- minimal_interfaces/              # Interfaces personalizadas
|   |   |-srv/
|   |       |- ComandoNavegacion.srv     # Definición del servicio
|- mapas/                               # Mapas generados
|   |- casa_map.pgm
|   |- casa_map.yaml
|- scripts/                             # Scripts de automatización y auxiliares
    |- setup_env.sh                     # Configuración del entorno
    |- paso1_gazebo.sh                  # Lanzamiento de Gazebo
    |- paso2_slam.sh                    # Lanzamiento de SLAM
    |- paso3_nav.sh                     # Lanzamiento de Nav2
    |- paso4_rviz.sh                    # Lanzamiento de RViz
    |- paso5_mapa.sh                    # Guardado del mapa
```

```
|- gazebo_cliente.sh              # Lanzamiento de la interfaz de gazebo
|- ejecutar_todo.sh               # Script completo para Tarea 1
|- ejecutar_todo2.sh              # Script completo para Tarea 2
```

3. Tarea 0: Mapeo de la Casa

3.1 Objetivo

Generar un mapa 2D del entorno simulado de la casa utilizando SLAM y Nav2 para ayudar a navegarla. Luego editar el mapa si fuera necesario para limpiarlo y dejarlo preparado para las tareas posteriores.

3.2 Proceso de Mapeo

Ejecución de Gazebo Primero necesitamos ejecutar el servidor de Gazebo con el mundo de la casa usando el comando `./scripts/paso1_gazebo.sh`

Ejecución del SLAM Luego necesitamos iniciar el proceso de SLAM para empezar a mapear el entorno y que el robot se pueda ubicar con `./scripts/paso2_slam.sh`

Ejecución de Nav2 Luego tenemos que iniciar el stack de navegación Nav2 para controlar el robot durante la exploración y que así sea más sencillo recorrer la casa con `./scripts/paso3_nav.sh`

Ejecución de Rviz Finalmente, lanzamos RViz para visualizar el proceso de mapeo en tiempo real y asi poder seleccionar los puntos de exploración con `./scripts/paso4_rviz.sh`

Nos aseguraremos de que el robot explore toda la casa siguiendo estas pautas:

- Cobertura completa de todas las habitaciones
- Múltiples pasadas por varias habitaciones para reducir el ruido

Guardado del Mapa

Una vez completada la exploración tenemos que guardar el mapa generado con `./scripts/-paso5_mapa.sh`

Esto genera dos archivos: - **casa_map.pgm:** Imagen del mapa en formato PGM (escala de grises) - **casa_map.yaml:** Metadatos del mapa con parámetros de configuración

3.3 Parámetros y mapa Final

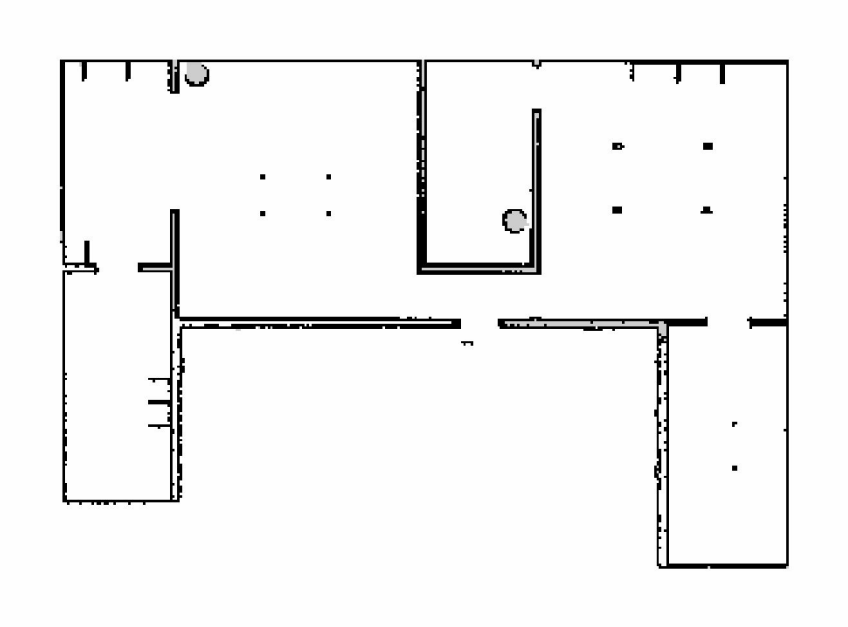

Figure 1: Mapa generado de la casa

```yaml
image: casa_map.pgm
mode: trinary
resolution: 0.05        # 5 cm por píxel
origin: [-8.79, -6.42, 0]  # Posición del origen en metros
occupied_thresh: 0.65   # Umbral para celdas ocupadas
free_thresh: 0.25       # Umbral para celdas libres
```

3.4 Problemas Encontrados y Soluciones

Problema 1: Mapa inicial muy pequeño

- **Causa:** Las zonas de fuera no se exploraban
- **Solución:** Ajustes con GIMP para ampliar el tamaño del mapa

Problema 2: Paredes no completas y zonas grises

- **Causa:** Ruido en el SLAM
- **Solución:** Ajuste manual en GIMP para limpiar el mapa y asegurar paredes completas

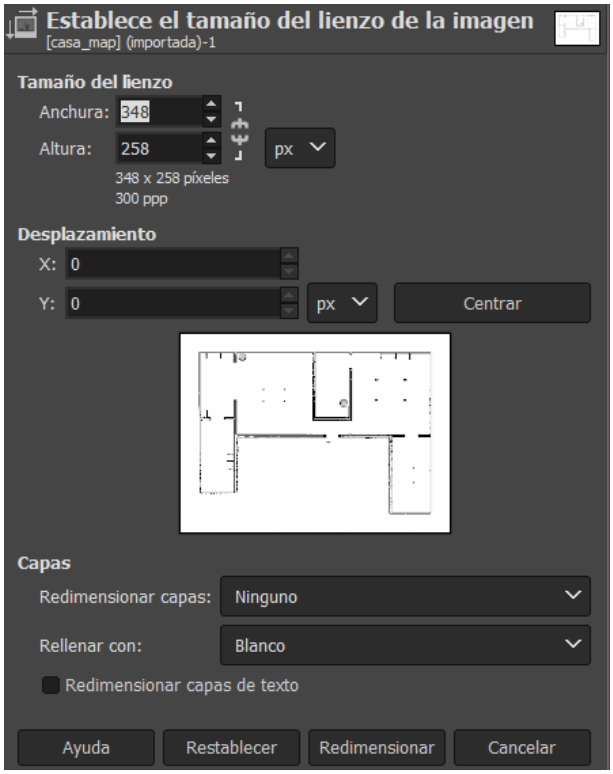

Figure 2: Comando utilizado en GIMP para aumentar el tamaño del mapa

4. Tarea 1: Servicio de Comandos de Navegación

4.1 Objetivo

Desarrollar un servicio ROS2 que permita enviar comandos de navegación al robot, abstrayendo la complejidad de Nav2. La implementacion incluye el comando de patrullaje y retorno a la salida.

4.2 Definición del Servicio

Se creó una interfaz personalizada `ComandoNavegacion.srv`:

```
# Solicitud del servicio
string comando  # "Patrullar" o "GoToExit"

# Respuesta del servicio
bool exito
string mensaje
```

4.3 Implementación del Servidor

Comandos Implementados

1. **Salida:** Retorno al origen
    - Navega a un punto establecido justo fuera de la puerta

- Punto de referencia conocido

2. **Patrullar:** Patrullaje continuo por waypoints predefinidos

   - Ciclo infinito por puntos predefinidos
   - Se detiene cuando el cliente deja de mandar el comando

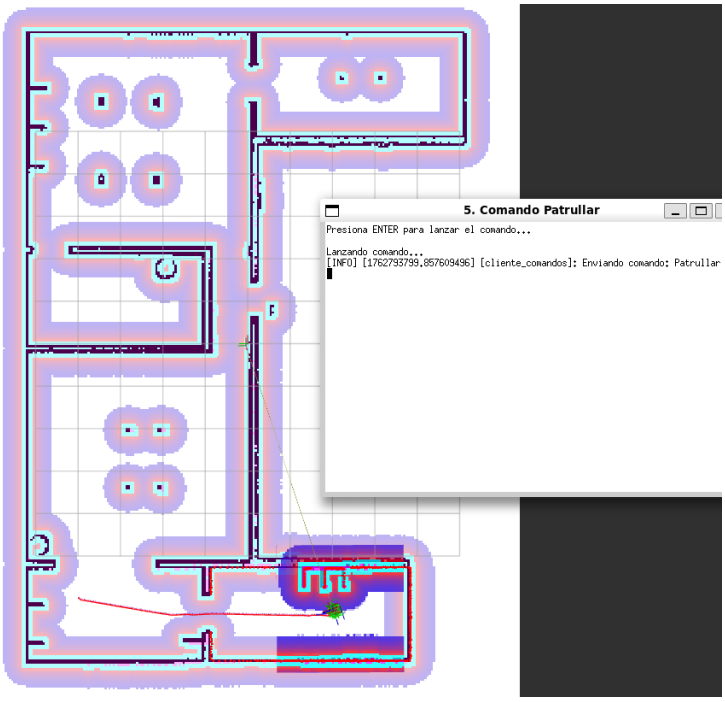

Figure 3: Mapa con Nav2 funcionando

Arbol de decisiones de los comandos

```
def comando_callback(self, request, response):
    ...
    # Arbol de decisiones para enrutar los comandos a funciones específicas
    if request.comando == "Patrullar":
        response.exito, response.mensaje = self.patrullar()
    elif request.comando == "GoToExit":
        response.exito, response.mensaje = self.ir_a_salida()
    ...
```

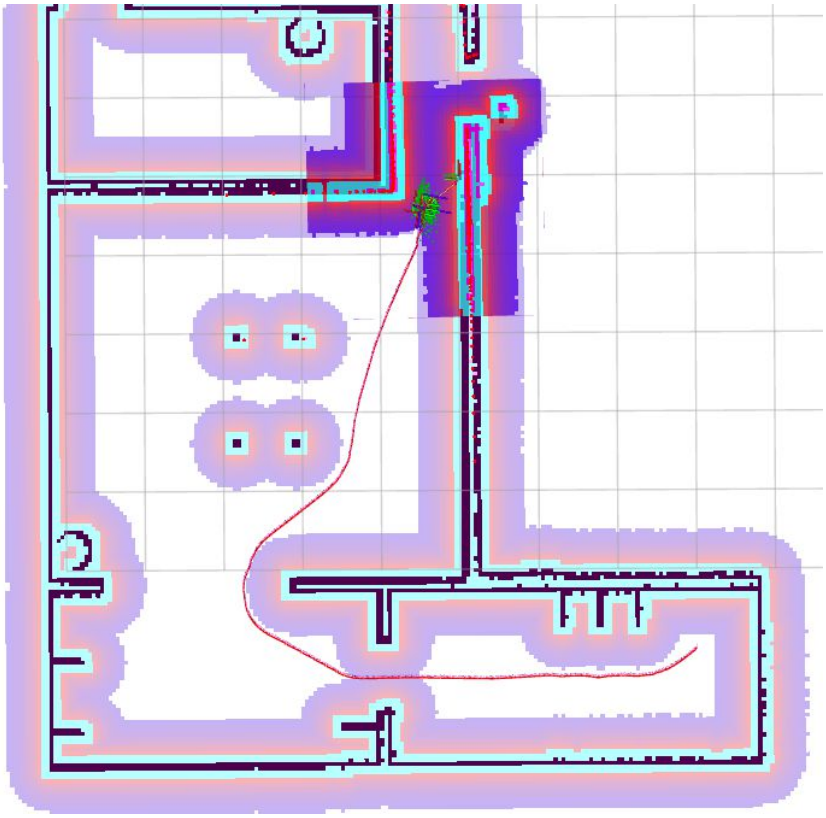

Figure 4: Comando Salida

4.4 Problemas Encontrados y Soluciones

Problema 1: Comando Patrullar paraba tras una iteración

- **Causa:** No se gestionaba el ciclo continuo
- **Solución:** Implementar flag `self.patrullando` y bucle infinito

Problema 2: Puntos de patrullaje y zonas complejas

- **Causa:** Puntos con obstáculos que tienden a producir que el robot no encuentre un camino hacia el objetivo
- **Solución:** Ajuste manual de las coordenadas de los puntos tras pruebas

5. Tarea 2: Búsqueda del Tesoro con Patrullaje

5.1 Objetivo

Desarrollar un nodo autónomo que busca un “tesoro” por dentro o fuera de la casa e intenta llegar a él en 90 segundos. Para ello lee la informacion del nodo_tesoro y calcula donde estará el tesoro para localizarlo y acercarse a el.

Resumen del Sistema

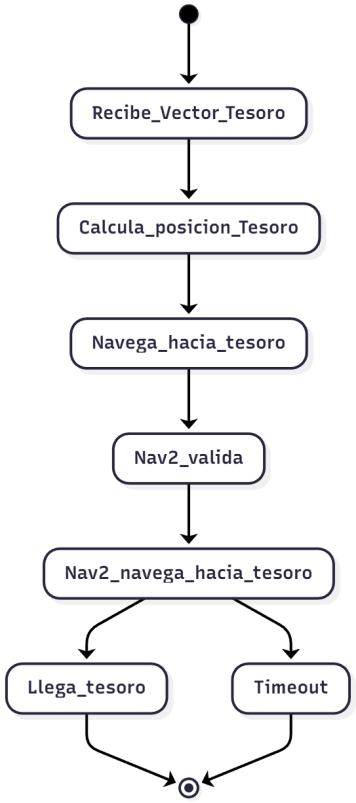

Figure 5: Diagrama de estados

5.2 Estrategia de Búsqueda

Algoritmo de Detección

El tesoro se calcula con los siguientes pasos:

- Se calcula la posicion actual del robot
- Se recibe el vector de distancia del tesoro y se guarda
- Se suma el vector de distancia a la posición del robot para obtener la posición del tesoro

La posicion de tesoro aproximada será la posición del robot más el vector de distancia al tesoro

```
Iniciando publicador del tesoro...
[INFO] [1762795060.166892327] [tesoro_nodo]: Inicializando Nodo Tesoro
[INFO] [1762795062.163717799] [tesoro_nodo]: Distancia al tesoro: X=2.599357, Y=
5.000000, total=5.635304
[INFO] [1762795063.157433460] [tesoro_nodo]: Distancia al tesoro: X=2.599321, Y=
5.000000, total=5.635288
[INFO] [1762795064.150964509] [tesoro_nodo]: Distancia al tesoro: X=2.532276, Y=
5.002222, total=5.606661
[INFO] [1762795065.145552369] [tesoro_nodo]: Distancia al tesoro: X=2.302714, Y=5.106750, total=5.6
01909
[INFO] [1762795066.143213582] [tesoro_nodo]: Distancia al tesoro: X=2.094867, Y=5.250351, total=5.6
52845
[INFO] [1762795067.138109568] [tesoro_nodo]: Distancia al tesoro: X=1.868707, Y=5.357402, total=5.673960
[INFO] [1762795068.132845979] [tesoro_nodo]: Distancia al tesoro: X=1.628404, Y=5.421140, total=5.660429
[INFO] [1762795069.130454352] [tesoro_nodo]: Distancia al tesoro: X=1.381272, Y=5.452943, total=5.625166
[INFO] [1762795070.149442087] [tesoro_nodo]: Distancia al tesoro: X=1.122503, Y=5.475437, total=5.589314
[INFO] [1762795071.165326974] [tesoro_nodo]: Distancia al tesoro: X=0.866369, Y=5.512582, total=5.580247
```

Figure 6: Recepcion de Coordenadas del tesoro

```
Iniciando nodo de búsqueda autónoma...
[INFO] [1762795061.610073735] [busqueda_tesoro_node]: Búsqueda activada
[INFO] [1762795063.614235455] [busqueda_tesoro_node]: Tesoro calculado en: (0.60
, 4.50)
[INFO] [1762795063.615346184] [busqueda_tesoro_node]: Goal enviado: (0.60, 4.50)
[INFO] [1762795113.300601583] [busqueda_tesoro_node]: Tesoro encontrado. Tiempo:
 49.69s
[INFO] [1762795113.301545730] [busqueda_tesoro_node]: Búsqueda desactivada
```

Figure 7: Envio de coordenadas del tesoro

5.3 Acercamiento al Tesoro

Una vez se recibe informacion sobre el tesoro:

1. **Cálculo de posición global:** Utiliza SLAM para localizar el robot en el mapa
2. **Navegación hacia el tesoro:** Envía un goal a Nav2 con la posición estimada del tesoro
3. **Monitoreo continuo:** Nav2 navega autónomamente hacia el tesoro, evitando los obstáculos y planificando automáticamente la ruta.
4. **Tesoro:** El tesoro se considera encontrado al estar a < 0.5m del mismo.

5.6 Problemas Encontrados y Soluciones

Problema 1: Problemas si el tesoro estaba dentro de la casa

- **Causa:** El tesoro puede aparecer dentro de la casa y puede haber paredes entre el robot y el tesoro
- **Solución:** Utilizar Nav2 en vez de mover directamente el robot para asi utilizar la planificación de rutas de Nav2

Problema 2: Robot no podia navegar por fuera de la casa

- **Causa:** No habia suficientes caracteristicas en el mapa para que el SLAM funcionara bien
- **Solución:** Modificar los ajustes del SLAM para mejorar la localización con menos características

en exteriores.

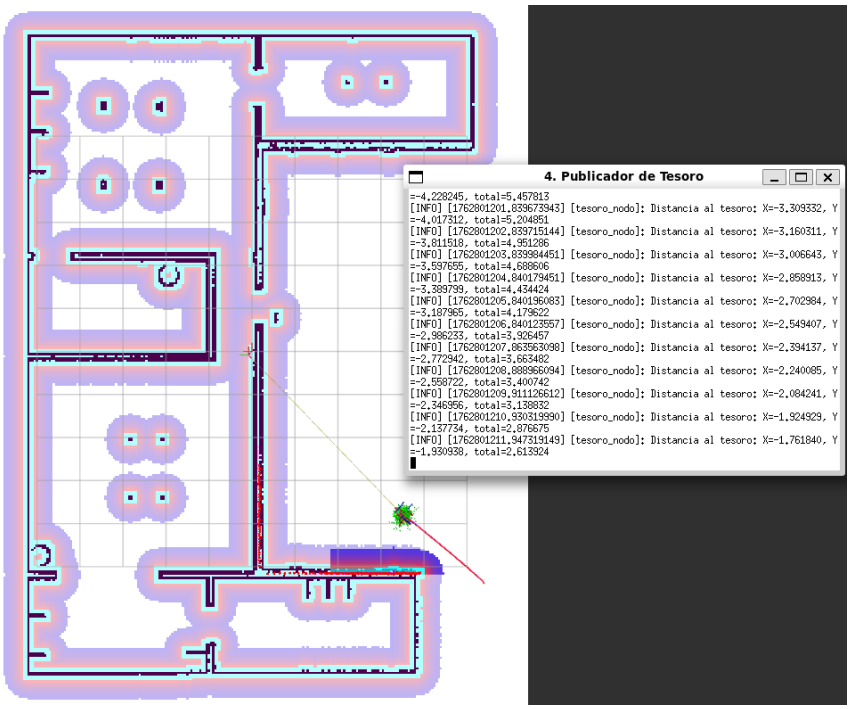

Figure 8: Detección del tesoro en RViz

6. Scripts de Automatización

Para facilitar la ejecución y pruebas del sistema, se desarrollaron varios scripts bash que automatizan el lanzamiento de componentes.

6.1 Scripts Principales

`setup_env.sh`: Configura el entorno ROS2 y se utiliza de manera auxiliar en todos los demas scripts.

`paso1_gazebo.sh`: Lanza Gazebo con el mundo de la casa con `./scripts/paso1_gazebo.sh`

`paso2_slam.sh`: Inicia SLAM para mapeo con `./scripts/paso2_slam.sh`

`paso3_nav.sh`: Lanza el stack de navegación Nav2 con `./scripts/paso3_nav.sh`

`paso4_rviz.sh`: Lanza RViz para visualización con `./scripts/paso4_rviz.sh`

`paso5_mapa.sh`: Guarda el mapa generado con `./scripts/paso5_mapa.sh`

`ejecutar_todo.sh`: Script maestro que lanza todo el sistema necesario para la Tarea 1. Compila ademas el workspace antes de lanzar todo. Para asegurarse de que el servidor de Gazebo esté listo antes de iniciar SLAM y Nav2, incluye una barrera que se sobrepasa pulsando ENTER.

`ejecutar_todo2.sh` Script maestro que lanza todo el sistema necesario para la Tarea 2. Compila ademas

el workspace antes de lanzar todo. Para asegurarse de que el servidor de Gazebo esté listo antes de iniciar SLAM y Nav2, incluye una barrera que se sobrepasa pulsando ENTER.

7. Configuración y Parámetros

7.1 Configuración del Robot

`turtlebot3_params.yaml`

```yaml
robot:
  # Dimensiones físicas del waffle
  radius: 0.220
  wheel_base: 0.287

  # Límites de velocidad
  max_linear_vel: 0.26
  max_angular_vel: 1.82

  # Parametros de aceleración
  linear_acc: 2.5
  angular_acc: 3.2

  # Parametros del sensor láser
  laser:
    min_range: 0.12
    max_range: 3.5
    min_angle: -3.14159
    max_angle: 3.14159
    samples: 360
```

7.2 Parámetros de Navegación

```yaml
nav2:
  # Planificador de ruta
  planner:
    plugin: "nav2_navfn_planner/NavfnPlanner"
    tolerance: 0.5
    use_astar: false

  # Controlador de trayectoria
  controller:
    plugin: "nav2_regulated_pure_pursuit_controller"
    desired_linear_vel: 0.25
    max_linear_accel: 2.5
    lookahead_dist: 0.6
```

```
# Los mecanismos de recuperación cuando el robot se queda atascado
recovery:
  - spin
  - backup
  - wait
```

9. Conclusiones

El proyecto ha cumplido satisfactoriamente todos los objetivos planteados.

En la Tarea 0, se generó un mapa de la casa utilizando SLAM, que sirvió como base para las tareas posteriores. En la Tarea 1, se implementó un servicio de comandos que permitió al robot realizar patrullajes y regresar a la salida de manera autónoma. Finalmente, en la Tarea 2, se desarrolló un sistema de búsqueda del tesoro que demostró la capacidad del robot para navegar y localizar objetivos basándose en información recibida.

Todas estas implementacions funcionan de manera resistente, adaptandose al entorno simulado y gestionando errores comunes.