Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Eléctrica

Proyecto Final: El Aspirante

EL5206-2 Laboratorio de Inteligencia Computacional y Robótica

Nombre: Sebastián Calderón
Profesor: Carlos Navarro
Martin Adams
Auxiliares: Ignacio Romero
Jorge Zambrano
Ayudante: Sebastián Herrera
Ayudantes de Laboratorio: Francisco Soto
Matías Carvajal
Mattías Prieto

Fecha de entrega: 13 de Julio de 2025
Santiago de Chile

Índice de Contenidos

Índice de Figuras

Índice de Tablas

Índice de Códigos

1. Introducción

En el contexto actual de la robótica, el desarrollo de robots autónomos capaces de desplazarse de manera eficiente en entornos desconocidos es un área de interés creciente. Un ejemplo cotidiano de esta tecnología son las aspiradoras robóticas, las cuales deben recorrer de forma sistemática los espacios disponibles para asegurar una cobertura completa del área a limpiar, evitando obstáculos y adaptándose a la geometría del entorno. Este proyecto se enmarca dentro de ese desafío, proponiendo la implementación de un sistema de navegación autónoma para una aspiradora robot en un ambiente simulado con Gazebo y ROS.

El objetivo principal del proyecto fue desarrollar un sistema capaz de realizar un proceso de **Coverage Path Planning**, es decir, la planificación de una trayectoria que permita al robot cubrir la totalidad del área libre de un entorno determinado. Para lograr esto, fue necesario integrar distintos módulos de percepción, localización y navegación, incluyendo algoritmos ampliamente utilizados en la robótica móvil como GMapping, AMCL y Move Base. El sistema propuesto consta de las siguientes etapas:

- **Mapeo del entorno:** Utilizando el algoritmo **GMapping**, se generó un mapa del entorno en formato `OccupancyGrid`, el cual fue construido a partir de los datos del sensor `LaserScan` y la odometría del robot.
- **Localización:** Se empleó el algoritmo **AMCL (Adaptive Monte Carlo Localization)** para estimar en tiempo real la pose del robot dentro del mapa generado, fusionando la información de los sensores con un modelo probabilístico basado en filtros de partículas.
- **Planificación y navegación:** A través del nodo **Move Base**, se gestionó la navegación autónoma, utilizando un planificador global para definir rutas y un planificador local basado en el *Dynamic Window Approach (DWA)* para evitar obstáculos dinámicos en tiempo real.
- **Generación de la trayectorias:** Se implementó un algoritmo propio de **Coverage Path Planning**, el cual reduce la resolución del mapa para simplificar el problema y genera una trayectoria zigzagueante que permite cubrir el área libre de forma eficiente. Los puntos generados son enviados secuencialmente a Move Base para su ejecución.

El presente informe se estructura de la siguiente manera:

- En el **Marco Teórico** se describen los conceptos fundamentales y los algoritmos involucrados en el sistema, incluyendo GMapping, Move Base, AMCL y la lógica del algoritmo de generación de trayectorias implementado.
- En la sección de **Tareas a Resolver** se definen los objetivos específicos del proyecto, detallando los requerimientos y desafíos abordados.

- En la **Metodología** se explica el proceso de desarrollo, describiendo la configuración del sistema, los archivos `launch`, y la implementación del algoritmo de Coverage Path Planning.
- La sección de **Resultados** presenta las trayectorias obtenidas y los mapas generados, diferenciando los casos de una habitación simple y una habitación de dos ambientes conectados.
- Finalmente, en el **Análisis de Resultados y Conclusiones** se discuten los problemas observados, las limitaciones del sistema actual y se proponen líneas de trabajo futuras.

Los resultados obtenidos muestran un desempeño satisfactorio del sistema en ambientes simples, logrando coberturas completas en una habitación rectangular. Sin embargo, al incrementar la complejidad del entorno, el sistema presentó dificultades para gestionar correctamente la transición entre ambientes separados por paredes y puertas. Estos desafíos abren la puerta a futuras mejoras en la lógica del algoritmo de cobertura y en la parametrización de los módulos de navegación.

2. Marco Teórico

En esta sección se revisan los conceptos fundamentales asociados al desarrollo del proyecto. Específicamente, se detallan tres componentes esenciales para lograr una navegación efectiva en entornos inicialmente desconocidos: el algoritmo GMapping para generación simultánea de mapas y localización (SLAM), el nodo Move Base para planificación y control de trayectorias, y el algoritmo Adaptive Monte Carlo Localization (AMCL) para estimar con precisión la posición del robot sobre el mapa generado.

2.1. GMapping

GMapping es un algoritmo de SLAM (*Simultaneous Localization and Mapping*) basado en filtros de partículas, comúnmente utilizado en robótica móvil para construir mapas de entornos inicialmente desconocidos, manteniendo simultáneamente una estimación precisa de la posición del robot. En este método, cada partícula representa una posible posición del robot y lleva asociada una hipótesis parcial del mapa. A medida que el robot se desplaza y obtiene nuevas lecturas sensoriales, estas partículas se actualizan mediante un filtro probabilístico, permitiendo mantener simultáneamente múltiples hipótesis del entorno y reduciendo así la incertidumbre acumulada por errores de odometría. GMapping implementa una variante optimizada del algoritmo *FastSLAM 2.0*, combinando:

- Un **filtro de partículas** que estima continuamente la trayectoria del robot.
- Conjuntos de **mapas locales** vinculados individualmente a cada partícula.
- Modelos probabilísticos del movimiento y de la percepción sensorial.

La probabilidad de que el robot se encuentre en una pose x_t en el instante t, dado el historial de observaciones $z_{1:t}$ y comandos de movimiento $u_{1:t}$, es calculada mediante la siguiente ecuación:

$$p(x_t|z_{1:t}, u_{1:t}) = \eta \cdot p(z_t|x_t) \cdot \int p(x_t|x_{t-1}, u_t) \cdot p(x_{t-1}|z_{1:t-1}, u_{1:t-1})\, dx_{t-1} \tag{1}$$

donde η es un factor de normalización, $p(z_t|x_t)$ representa el modelo sensorial y $p(x_t|x_{t-1}, u_t)$ representa el modelo de movimiento.

2.1.1. Rol de GMapping en el sistema

La integración de GMapping en este proyecto cumple tres funciones críticas:

- Genera un mapa tipo `OccupancyGrid`, base esencial para el posterior algoritmo de *Coverage Path Planning*.
- Proporciona un marco de referencia global estable (`map`) para la navegación.
- Permite la integración efectiva con AMCL para una localización precisa y continua del robot.

2.2. Move Base

El nodo *Move Base* constituye el componente central del sistema de navegación basado en ROS. Este nodo coordina tanto la planificación global como la ejecución local de las trayectorias del robot, operando como intermediario entre el algoritmo de *Coverage Path Planning*, encargado de generar los objetivos de navegación, y los comandos enviados al robot para la ejecución física de las trayectorias.

2.2.1. Arquitectura de Move Base

La arquitectura del nodo Move Base está formada por cuatro componentes principales integrados:

- **Global Planner**: encargado de generar rutas globales considerando el mapa estático proporcionado por GMapping.
- **Local Planner (Dynamic Window Approach - DWA)**: encargado de evitar obstáculos locales dinámicos mediante la actualización continua de las velocidades lineales y angulares del robot.
- **Global Costmap**: representación estática global del entorno, donde se identifican obstáculos permanentes y zonas transitables.
- **Local Costmap**: mapa dinámico local del entorno inmediato, que permite reaccionar rápidamente ante la aparición de obstáculos no previstos inicialmente.

2.2.2. Modelo Matemático del Local Planner (DWA)

El algoritmo de navegación local implementado es el *Dynamic Window Approach (DWA)*, que determina las velocidades óptimas (v, ω) utilizando la siguiente función de optimización:

$$G(v,\omega) = \alpha \cdot \text{heading}(v,\omega) + \beta \cdot \text{dist}(v,\omega) + \gamma \cdot \text{vel}(v,\omega) \tag{2}$$

donde $\text{heading}(v,\omega)$ evalúa la alineación del robot con la meta global, $\text{dist}(v,\omega)$ mide la distancia a los obstáculos cercanos, priorizando la seguridad en el desplazamiento, $\text{vel}(v,\omega)$ mide la eficacia del movimiento en dirección al objetivo planteado. En esta implementación, los parámetros clave para DWA se configuran como se muestra en la Tabla 1.

Tabla 1: Parámetros clave utilizados por el nodo Move Base en la navegación local del robot.

Parámetro	**Valor**
`max_vel_x`	0.26 m/s
`xy_goal_tolerance`	0.05 m
`inflation_radius`	0.5 m

2.3. Adaptive Monte Carlo Localization (AMCL)

AMCL es un algoritmo probabilístico de localización basado en filtros de partículas adaptativos, que permite estimar la posición (x, y, θ) del robot sobre un mapa previamente conocido. El algoritmo fusiona información proveniente de odometría y sensores láser, para entregar una estimación confiable y precisa de la pose del robot.

2.3.1. Monte Carlo Localization Estándar

El algoritmo clásico de localización Monte Carlo (MCL) sigue un esquema de Bayes recursivo, calculado mediante:

$$bel(x_t) = \eta \cdot p(z_t|x_t) \cdot \int p(x_t|x_{t-1}, u_t) \cdot bel(x_{t-1})\, dx_{t-1} \tag{3}$$

donde $bel(x_t)$ es la creencia probabilística de la pose actual del robot, $p(z_t|x_t)$ es el modelo del sensor láser y $p(x_t|x_{t-1}, u_t)$ es el modelo de movimiento de odometría.

Cabe destacar que el algoritmo opera en tres etapas principales:

1. **Predicción**: difusión de partículas según el modelo de movimiento.
2. **Corrección**: ponderación según compatibilidad con datos sensoriales.
3. **Remuestreo**: selección de nuevas partículas en función del peso.

2.3.2. Extensiones y mejoras de AMCL

AMCL extiende el método estándar de MCL incorporando:

- **Número de partículas adaptativo**, ajustado dinámicamente según la eficiencia de partículas:

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} (w_t^i)^2} \tag{4}$$

- **Modelos sensoriales mejorados**, incorporando distribuciones no gaussianas para mediciones láser, aumentando la robustez de la localización.

2.3.3. Integración con el sistema global

La integración con Move Base y Coverage Planner se realiza de manera coordinada:

- AMCL publica la transformación TF `map` → `odom`.
- Move Base utiliza esta transformación para calcular objetivos globales.
- Coverage Planner recibe la posición actual desde el tópico `/amcl_pose` para generar nuevos objetivos.

3. Tareas a Resolver

El objetivo principal de este proyecto es implementar un sistema de navegación autónoma para una aspiradora robot, basado en un algoritmo de *Coverage Path Planning*, en una simulación desarrollada con ROS y Gazebo.

Para lograr este objetivo, primero se desarrollará un ambiente simulado simple, correspondiente a una habitación rectangular en Gazebo, donde se probará inicialmente el comportamiento del robot. Luego, si se cumplen satisfactoriamente los objetivos iniciales, se ampliará el entorno agregando complejidad adicional mediante obstáculos y divisiones internas.

Las tareas específicas incluyen generar mapas del entorno utilizando el algoritmo *GMapping*, empleando los datos sensoriales del robot obtenidos con un sensor láser. A partir del mapa generado, se utilizara el algoritmo de localización probabilística AMCL, permitiendo al robot estimar continuamente su posición dentro del entorno.

Posteriormente, se utilizará el nodo *Move Base* para gestionar la planificación y control de trayectorias globales y locales del robot, considerando tanto los obstáculos estáticos detectados previamente en el mapa como posibles obstáculos dinámicos que puedan surgir durante la navegación.

Finalmente, se desarrollará e integrará un algoritmo de *Coverage Path Planning* que genere objetivos sucesivos o waypoints, asegurando que el robot cubra de manera eficiente y completa toda el área transitable. Este algoritmo se evaluará primero en una habitación sencilla y posteriormente en ambientes más complejos, analizando cómo influye el aumento de complejidad del entorno en la efectividad y desempeño del sistema completo.

4. Metodología

En esta sección se detalla el proceso seguido para desarrollar la implementación del proyecto de la aspiradora robot. Esta metodología se divide en dos etapas fundamentales: primero, la generación del mapa estático mediante el algoritmo GMapping; segundo, la configuración del entorno de navegación y la implementación del algoritmo de *Coverage Path Planning*, ambas etapas lanzadas mediante el archivo `aspirante_test.launch`.

4.1. Generación de mapas mediante GMapping

Antes de lanzar el sistema de navegación autónoma, fue necesario generar un mapa estático del entorno simulado. Para esto se utilizó el algoritmo GMapping. Este algoritmo fue ejecutado independientemente en una etapa previa, generando un mapa de tipo `OccupancyGrid`, indispensable para la navegación y localización posterior del robot.

Una vez obtenido un mapa confiable y preciso, este fue almacenado en formato YAML, quedando disponible para ser utilizado posteriormente por los nodos AMCL y Move Base en la navegación efectiva del robot.

4.2. Configuración del entorno de navegación

Para realizar la navegación autónoma efectiva, se construyó el archivo launch denominado `aspirante_test.launch`. Este archivo permitió integrar y configurar los algoritmos necesarios para la localización y navegación efectiva del robot en Gazebo, tomando como entrada el mapa generado previamente con GMapping. En particular, el archivo launch incorporó los siguientes nodos esenciales:

- **Nodo AMCL**: Fue configurado para llevar a cabo la localización precisa y continua del robot dentro del mapa generado previamente. AMCL utiliza los datos del sensor LaserScan junto con la odometría del robot para estimar en tiempo real la pose del mismo respecto al marco global (`map`), actualizando continuamente la transformación entre los marcos `map` y `odom`. En el Código 1 se muestra la línea donde se lanza el nodo AMCL utilizando los parámetros predeterminados del paquete `turtlebot3_navigation`.

Código 1: Lanzamiento del Nodo AMCL.

```
1 <!-- AMCL -->
2   <include file="$(find turtlebot3_navigation)/launch/amcl.launch"/>
```

- **Nodo Move Base**: Este nodo constituyó el núcleo del sistema de navegación y fue el encargado de ejecutar trayectorias hacia objetivos específicos. Su arquitectura está compuesta por un planificador global, que determina rutas hacia puntos distantes empleando el mapa estático previamente generado, y un planificador local basado en el método *Dynamic Window Approach (DWA)*, que ajusta constantemente las trayectorias para evitar obstáculos en tiempo real. En el Código 2 se muestra la configuración empleada, indicando los archivos que contienen los parámetros necesarios para el correcto funcionamiento del algoritmo.

Código 2: Lanzamiento del Nodo Move Base.

```
<!-- Move Base -->
  <node pkg="move_base" type="move_base" respawn="false" name="move_base">
    <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS"/>
    <rosparam file="$(find aspirante)/param/costmap_common_params.yaml"/>
    <rosparam file="$(find aspirante)/param/costmap_common_params.yaml"/>
    <rosparam file="$(find aspirante)/param/local_costmap_params.yaml"/>
    <rosparam file="$(find aspirante)/param/global_costmap_params.yaml"/>
    <rosparam file="$(find aspirante)/param/move_base_params.yaml"/>
    <rosparam file="$(find aspirante)/param/dwa_local_planner_params.yaml"/>
    <remap from="cmd_vel" to="$(arg cmd_vel_topic)"/>
    <remap from="odom" to="$(arg odom_topic)"/>
    <param name="DWAPlannerROS/min_vel_x" value="0.0" if="$(arg
    ↪ move_forward_only)" />
  </node>
```

- **Nodo Coverage Path Planning**: Finalmente, el archivo `aspirante_test.launch` se encarga de iniciar el nodo correspondiente al algoritmo de *Coverage Path Planning*, el cual utiliza la información proveniente de los nodos previamente iniciados (mapa generado por GMapping, pose estimada por AMCL, y navegación mediante Move Base) para construir y ejecutar trayectorias que cubran sistemáticamente el entorno. El lanzamiento de este nodo se muestra en el Código 3.

Código 3: Lanzamiento del Nodo Coverage Path Planning.

```
<!-- Coverage path planner -->
  <node pkg="aspirante" name="coverage_path_planning_test" type="
  ↪ coverage_path_planning_test.py" output="screen"/>
```

4.3. Algoritmo de Coverage Path Planning

Una vez establecido el entorno de navegación, se procedió a implementar el algoritmo responsable de generar trayectorias eficientes que permiten al robot recorrer toda la superficie libre de una habitación. Este algoritmo, conocido como *Coverage Path Planning*, fue desarrollado en el script `coverage_path_planning_test.py`, y su funcionamiento puede dividirse en cuatro etapas principales. A continuación, se explica la lógica general y el propósito de cada uno de los métodos presentes en la clase `CoveragePathPlanner`, destacando su rol dentro del sistema general.

4.3.1. Inicialización de Parámetros

El algoritmo comienza inicializando el nodo de ROS y configurando los parámetros esenciales, tales como el umbral de cercanía para considerar un objetivo como alcanzado, la resolución deseada del mapa, y el punto de inicio del algoritmo.

Además, se suscriben los tópicos `/map` y `/odom`, y se inicializa un cliente para enviar objetivos al nodo `move_base`. Una vez recibido el mapa y verificada la disponibilidad del servidor de navegación, se lanza el planificador de trayectorias mediante el método del Código 4.

`run_coverage_planner()`

Esta función principal orquesta la ejecución del algoritmo: procesa el mapa, genera la trayectoria de cobertura con BFS y ejecuta cada uno de los waypoints generados.

Código 4: Ejecución del algoritmo.

```
def run_coverage_planner(self):
    # Process map and plan path
    reduced_map = self.process_map()
    coverage_path = self.plan_path(reduced_map)

    # Execute path
    self.execute_path(coverage_path, reduced_map)
    rospy.loginfo("Coverage complete!")
```

4.3.2. Procesamiento inicial del mapa

El algoritmo recibe el mapa previamente generado por GMapping mediante el tópico `/map`. Para ajustar la precisión de los waypoints del algoritmo, este mapa es simplificado a una resolución especifica, agrupando bloques de celdas y marcándolos como ocupados si alguna celda en el bloque representa un obstáculo (ver Código 5).

process_map()

Transforma el OccupancyGrid en una versión reducida, dividiendo el entorno en bloques de mayor tamaño. Se consideran ocupados aquellos bloques que contengan al menos una celda con obstáculo, lo que permite reducir el espacio de búsqueda sin comprometer la seguridad de navegación.

Código 5: Procesamiento del mapa a menor resolución.

```
def process_map(self):
    resolution_ratio = self.GRID_RESOLUTION / self.current_map.info.resolution
    grid_ratio = int(round(resolution_ratio))

    # Reshape and reduce resolution
    original_height = self.current_map.info.height
    original_width = self.current_map.info.width
    original_data = np.array(self.current_map.data).reshape(original_height, original_width
     ↪ )

    # Create reduced map by checking blocks
    reduced_map = np.zeros((original_height // grid_ratio, original_width // grid_ratio),
     ↪ dtype=np.int8)

    for i in range(0, original_height, grid_ratio):
        for j in range(0, original_width, grid_ratio):
            block = original_data[i:i+grid_ratio, j:j+grid_ratio]

            # If any obstacle in the block
            if np.any(block > 0):
                reduced_map[i//grid_ratio, j//grid_ratio] = 1

    return reduced_map
```

4.3.3. Planificación de trayectoria

A partir del mapa simplificado, se asignan pesos a cada celda libre utilizando un algoritmo BFS (Breadth-first search). El robot selecciona celdas adyacentes no visitadas con mayor peso para construir una trayectoria que recorra de manera eficiente todas las zonas accesibles (ver Código 6).

plan_path(map_data)

Este método construye una grilla con coordenadas, pesos y tags de visitado. Luego, mediante BFS, se asignan prioridades a las celdas libres. El camino se genera iterativamente eligiendo la celda adyacente con mayor peso no visitada. Si no hay opciones, se elige la celda libre más cercana.

Código 6: Generación de trayectoria con BFS.

```
def plan_path(self, map_data):
    # Initialize grid
    height, width = map_data.shape
    grid = np.zeros((height, width), dtype=[('x', int), ('y', int),
                                            ('weight', int), ('visited', bool)])

    # Initialize grid cells
    for i in range(height):
        for j in range(width):
            grid[i,j] = (i, j, 0, bool(map_data[i,j]))

    # BFS from start position
    start_x, start_y = self.ORIGIN_X, self.ORIGIN_Y
    queue = [(start_x, start_y)]
    grid[start_x, start_y]['weight'] = 1

    # Assign weights using BFS
    while queue:
        x, y = queue.pop(0)
        for dx, dy in [(-1,0),(1,0),(0,-1),(0,1)]:
            nx, ny = x + dx, y + dy
            if 0 <= nx < height and 0 <= ny < width:
                if not grid[nx, ny]['visited'] and grid[nx, ny]['weight'] == 0:
                    grid[nx, ny]['weight'] = grid[x, y]['weight'] + 1
                    queue.append((nx, ny))

    # Generate coverage path
    path = []
    current = grid[start_x, start_y]
    current['visited'] = True
    path.append(current)

    while True:
        # Find adjacent unvisited cell with max weight
        next_cell = None
        max_weight = -1
        x, y = current['x'], current['y']

        for dx, dy in [(-1,0),(1,0),(0,-1),(0,1)]:
            nx, ny = x + dx, y + dy
            if 0 <= nx < height and 0 <= ny < width:
                if not grid[nx, ny]['visited'] and grid[nx, ny]['weight'] > max_weight:
                    max_weight = grid[nx, ny]['weight']
                    next_cell = grid[nx, ny]


```

```
        if next_cell is None:
            # Find closest unvisited cell
            min_dist = float('inf')
            for i in range(height):
                for j in range(width):
                    if not grid[i,j]['visited']:
                        dist = np.hypot(i-current['x'], j-current['y'])
                        if dist < min_dist:
                            min_dist = dist
                            next_cell = grid[i,j]
            if next_cell is None:
                break  # All cells visited

        next_cell['visited'] = True
        path.append(next_cell)
        current = next_cell

    rospy.loginfo(f"Generated path with {len(path)} waypoints")
    return path
```

4.3.4. Ejecución de la trayectoria generada

Cada punto de la trayectoria es transformado en coordenadas reales y enviado como objetivo al nodo **move_base**, encargado de ejecutar el movimiento utilizando el algoritmo DWA, ajustando en tiempo real ante la detección de obstáculos dinámicos (ver Código 7).

execute_path(path, map_data)

Recorre todos los puntos de la trayectoria. Por cada punto, genera un objetivo de navegación con `create_nav_goal()` y lo ejecuta mediante `navigate_to_goal()`, con hasta tres intentos por objetivo.

Código 7: Ejecución de la trayectoria generada.

```
def execute_path(self, path, map_data):
    for i, waypoint in enumerate(path):
        if rospy.is_shutdown():
            break

        # Create navigation goal
        goal = self.create_nav_goal(waypoint)
        rospy.loginfo(f"Navigating to [{waypoint['x']},{waypoint['y']}]")

        # Reaching waypoint with retries
        for attempt in range(3):
            if self.navigate_to_goal(goal):
                break
```

create_nav_goal(waypoint)

Traduce una celda del mapa simplificado a coordenadas reales del mundo, calcula la orientación hacia la meta usando trigonometría, y genera un mensaje MoveBaseGoal con esa información.

Código 8: Creación de objetivos de navegación.

```
def create_nav_goal(self, waypoint):
    goal = MoveBaseGoal()
    goal.target_pose.header.frame_id = "map"
    goal.target_pose.header.stamp = rospy.Time.now()

    # Convert grid to world coordinates
    goal.target_pose.pose.position.x = (waypoint['y'] - self.ORIGIN_Y) * self.
     ↪ GRID_RESOLUTION
    goal.target_pose.pose.position.y = (waypoint['x'] - self.ORIGIN_X) * self.
     ↪ GRID_RESOLUTION

    # Calculate orientation towards goal
    dx = goal.target_pose.pose.position.x - self.robot_pose.position.x
    dy = goal.target_pose.pose.position.y - self.robot_pose.position.y
    yaw = np.arctan2(dy, dx)
    q = quaternion_from_euler(0, 0, yaw)
    goal.target_pose.pose.orientation = Quaternion(*q)

    return goal
```

navigate_to_goal(goal)

Envía el objetivo al nodo **move_base**, monitorea su progreso y verifica si se alcanza. Se considera fallido si el tiempo supera los 30 segundos o si el estado del objetivo es ABORTED o REJECTED.

Código 9: Envío y monitoreo de objetivos.

```
def navigate_to_goal(self, goal):
    self.move_base.send_goal(goal)
    start_time = rospy.Time.now()

    while not rospy.is_shutdown():
        # Check if reached
        distance = np.hypot(
            self.robot_pose.position.x - goal.target_pose.pose.position.x,
            self.robot_pose.position.y - goal.target_pose.pose.position.y)


```

```
12          if distance < self.THRESHOLD:
13              rospy.loginfo("Waypoint reached")
14              return True
15
16          # Check timeout
17          if (rospy.Time.now() - start_time).to_sec() > 30.0:
18              rospy.logwarn("Navigation timeout")
19              self.move_base.cancel_goal()
20              return False
21
22          # Check for failures
23          state = self.move_base.get_state()
24          if state in [GoalStatus.ABORTED, GoalStatus.REJECTED, GoalStatus.PREEMPTED
        ↪ ]:
25              rospy.logwarn(f"Navigation failed with state {GoalStatus.to_string(state)}")
26              return False
27
28          rospy.sleep(0.1)
29
30      return False
```

En conjunto, este algoritmo genera una trayectoria tipo zigzag que cubre eficientemente el entorno, enviando objetivos secuenciales a través del cliente **move_base**. Gracias a la localización continua proporcionada por AMCL y al ajuste dinámico del planificador local, el robot puede recorrer el área deseada evitando colisiones y manteniendo su posición correctamente estimada dentro del mapa.

5. Resultados

En esta sección se presentan los principales resultados obtenidos tras la implementación del sistema de navegación del aspirante. Los experimentos se realizaron en un entorno simulado en Gazebo, donde el robot debía explorar, mapear y recorrer el área de manera eficiente siguiendo trayectorias construidas con el algoritmo de *Coverage Path Planning*.

5.1. Mapeo con GMapping

La primera etapa consistió en la generación del mapa del entorno mediante el algoritmo GMapping. Para esto, el robot recorrió una habitación simulada equipada con paredes y obstáculos, generando un `OccupancyGrid` que representa las áreas libres y ocupadas.

En la Figura 1 se observan tres etapas del mapeo del entorno: en la fila superior se muestra la evolución del `OccupancyGrid` generado en RViz, mientras que en la fila inferior se visualiza la posición del robot en el mundo simulado de Gazebo. A medida que el robot avanza por el entorno, va escaneando las paredes y obstáculos, completando progresivamente el mapa de ocupación. Este experimento fue realizado con el objetivo de validar el funcionamiento del algoritmo GMapping. Cabe destacar que ni este mapa generado ni este entorno fueron utilizados en las pruebas del algoritmo de *Coverage Path Planning*, ya que dichas pruebas se realizaron en un mundo de Gazebo distinto.

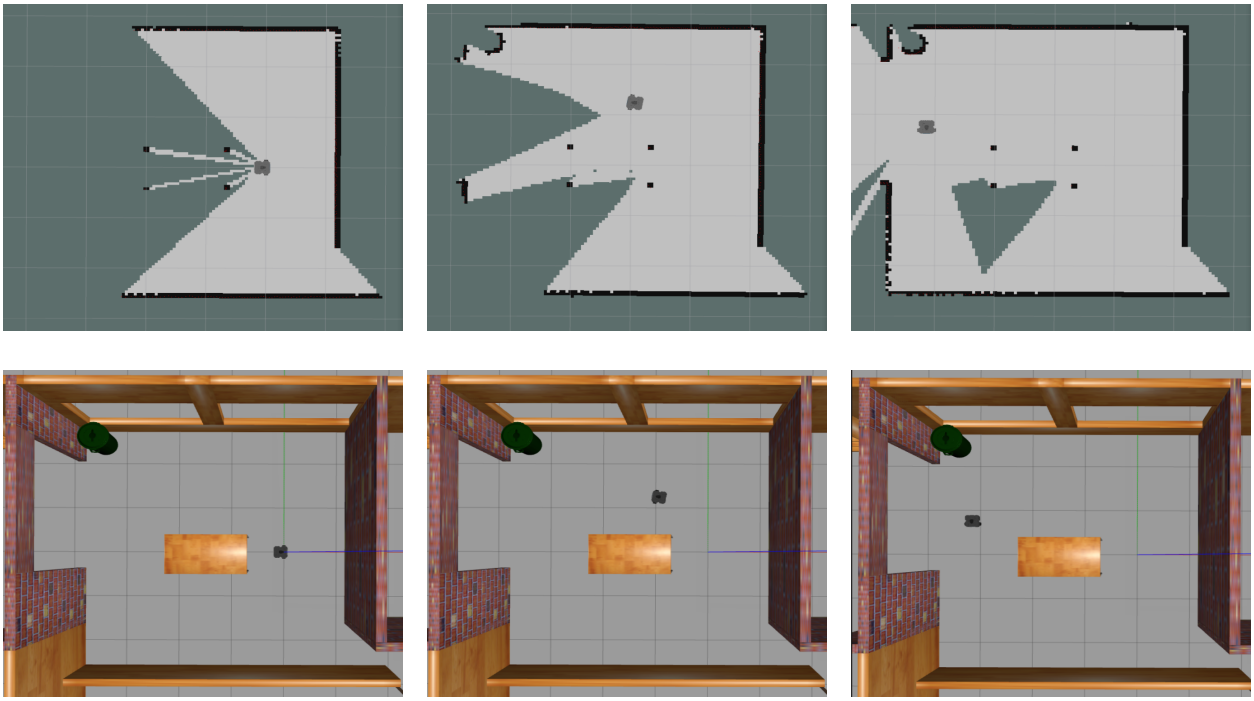

Figura 1: Proceso de obtencion de mapas utilizando GMapping.

Para realizar las pruebas del algoritmo de *Coverage Path Planning*, se crearon dos mundos en Gazebo, los cuales fueron mapeados utilizando GMapping para obtener mapas compatibles con los algoritmos de navegación y localización. Los mapas generados se muestran en la Figura 2. En dicha figura se observa la configuración de ambos entornos simulados: a la izquierda se presenta un mundo rectangular simple compuesto por una única habitación, mientras que a la derecha se muestra un entorno más complejo, donde paredes adicionales separan dos ambientes conectados por un pasillo. Este último escenario representa un mayor desafío para el algoritmo de *Coverage Path Planning*, debido a la necesidad de planificar rutas que conecten y cubran múltiples espacios.

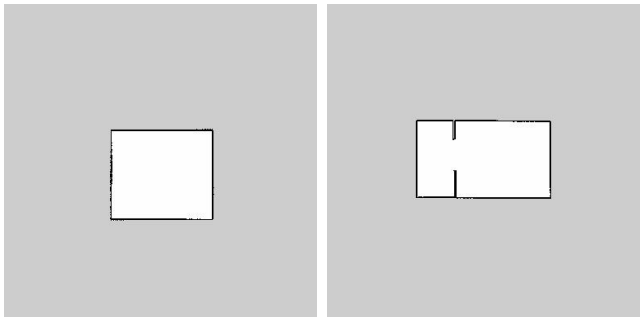

Figura 2: Mapas utilizados con el algoritmo de Coverage Path Planning.

5.2. Trayectorias Coverage Path Planning

Una vez generado el mapa, se procedió a ejecutar el algoritmo de *Coverage Path Planning*, el cual genera una trayectoria que permite al robot recorrer sistemáticamente toda el área libre, simulando el comportamiento de una aspiradora robot.

Habitación Rectangular

A continuación se muestran algunas de las trayectorias que se obtuvieron en el proceso de configuración y afinación de detalles del algoritmo. En la Figura 3, se observa como la trayectoria generada no cumple de forma satisfactoria con el objetivo del proyecto, ya que no logra cubrir completamente el área libre. Se observa un comportamiento desordenado, donde el robot genera movimientos en zig-zag amplios que no permiten un barrido eficiente, que no alcanza las esquinas y bordes del ambiente, lo cual es crucial para un sistema de aspiradora robot.

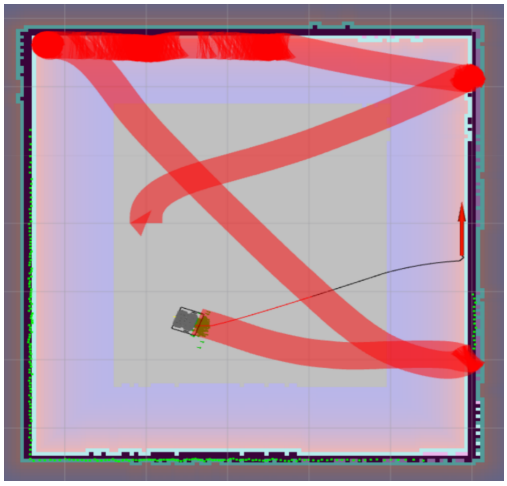

Figura 3: Trayectoria fallida.

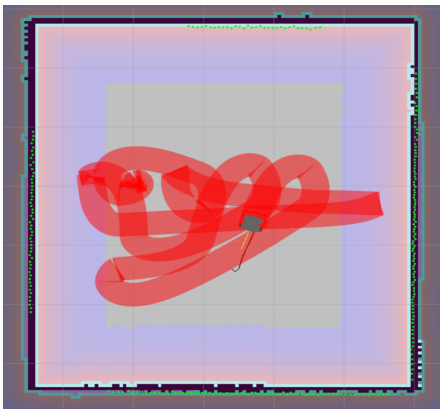

Figura 4: Trayectoria fallida.

Por otro lado, en la Figura 4 la trayectoria logra recorrer diversas zonas de la habitación, el patrón de movimiento no es óptimo ni ordenado. Se observan trayectorias sobrepuestas y giros innecesarios, lo que genera una cobertura redundante en ciertas áreas y deja sin explorar otras zonas cercanas a los bordes. Este comportamiento evidencia problemas en la planificación eficiente de la cobertura, afectando directamente la funcionalidad de la aspiradora robot. La meta del proyecto es lograr una cobertura sistemática y completa del espacio, evitando trayectorias desordenadas como la observada.

Siguiendo con el proceso de configuracion del algoritmo de *Coverage Path Planning*, se obtiene la trayectoria de la Figura 5. En este caso, la trayectoria presenta un patrón en zig-zag más estructurado en comparación con pruebas anteriores; sin embargo, aún se observan inconsistencias. En particular, persisten zonas del mapa que no son visitadas, especialmente cercanas a los bordes, lo que evidencia que la cobertura no es completa. Estas deficiencias muestran que, si bien el sistema mejora respecto a intentos anteriores, aún no cumple completamente con los objetivos del proyecto, que requieren una cobertura sistemática, sin omitir áreas y sin recorrer múltiples veces la misma zona.

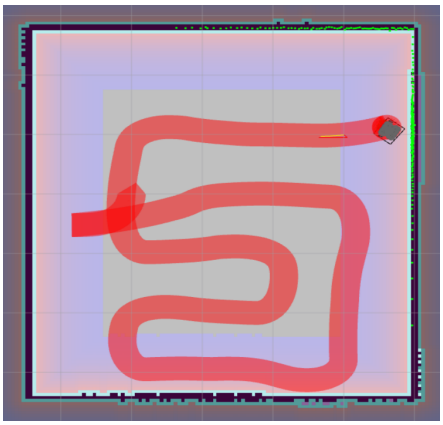

Figura 5: Trayectoria fallida.

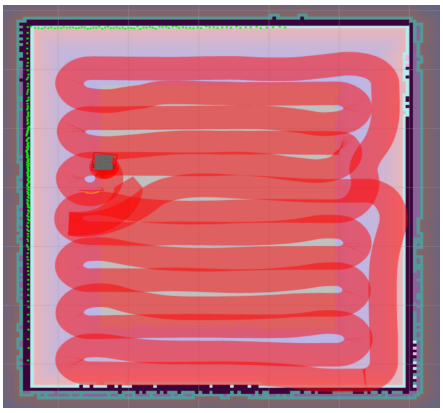

Figura 6: Trayectoria exitosa.

Tras un proceso iterativo de ajuste, se determinó la combinación óptima de parámetros para el algoritmo de Coverage Path Planning, el resultado se presenta en la Figura 6. En esta ejecución se observa un comportamiento adecuado y acorde a los objetivos del proyecto, ya que el robot recorre de forma ordenada y sistemática todo el espacio libre de la habitación siguiendo un patrón en zig-zag sin solapamientos innecesarios. La trayectoria cubre eficientemente toda la superficie, respetando los límites de las paredes y evitando colisiones. La ruta resultante es coherente con el comportamiento esperado de una aspiradora robot.

Habitación de dos ambientes

Una vez obtenidos resultados satisfactorios en la habitación de un solo ambiente, se decidió probar el algoritmo de *Coverage Path Planning* en un entorno de mayor complejidad. Para esto, se utilizó un mundo de Gazebo compuesto por dos habitaciones conectadas por un pasillo. Esta configuración añade un desafío adicional al sistema, ya que requiere que el robot sea capaz de moverse entre distintas zonas conectadas pero parcialmente separadas por obstáculos.

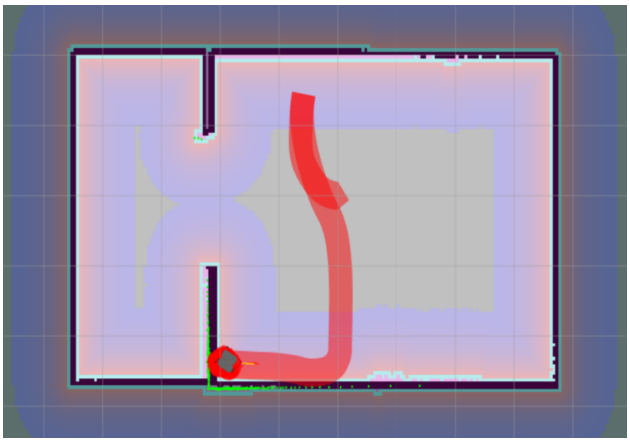

Figura 7: Trayectoria fallida en un entorno de dos ambientes.

La Figura 7 muestra uno de los intentos de cobertura en este entorno. Se observa cómo el robot inicia el recorrido en una de las habitaciones y comienza a recorrer la zona libre disponible. Sin embargo, en esta ejecución, el robot cubre parcialmente el espacio de la primera habitación sin lograr transitar hacia la segunda zona.

En un segundo intento, ilustrado en la Figura 8, el robot logra recorrer parcialmente el primer ambiente y luego se aproxima a la conexión entre las dos habitaciones. Sin embargo, la trayectoria presenta cambios de dirección abruptos y un comportamiento errático que impide la cobertura completa de ambas zonas. A pesar de que el robot planifica una trayectoria que evita la pared que separa los dos ambientes, termina realizando movimientos en forma de curva y regresa a la zona inicial, sin lograr cubrir eficientemente el segundo ambiente.

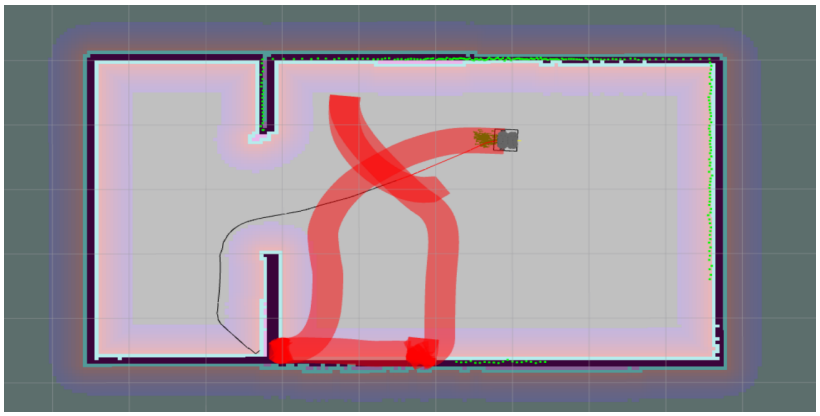

Figura 8: Trayectoria fallida en un entorno de dos ambientes.

6. Análisis de Resultados

En esta sección se analizan las trayectorias obtenidas tras la implementación del sistema de planificación de trayectorias, comparando los resultados obtenidos en cada escenario simulado. Los errores y aciertos observados se asocian tanto a la lógica de planificación del algoritmo de *Coverage Path Planning* como a la configuración de los parámetros del sistema, tales como la resolución de la grilla y los parámetros de Move Base.

6.1. Habitación Rectangular

En la Figura 3 se observa una trayectoria fallida, donde el robot no logra cubrir completamente el espacio libre. Se evidencia un patrón desordenado de movimiento, con desplazamientos zigzagueantes amplios y poco eficientes. Este comportamiento puede asociarse principalmente a un problema en la lógica inicial del algoritmo de *Coverage Path Planning*, particularmente en la forma en que se asignan prioridades de visita a las celdas o en la construcción del recorrido. El algoritmo generaba redundancia en los movimientos y no contemplaba correctamente las esquinas y bordes, lo cual es crucial para una cobertura eficiente en el contexto de una aspiradora robot.

Además, se debe considerar el rol del `Global Planner` de Move Base en este comportamiento. Aunque el planificador global está diseñado para calcular rutas óptimas hacia puntos distantes del mapa, en este caso el sistema recibe múltiples objetivos secuenciales generados por el algoritmo. La acumulación de rutas hacia puntos cercanos y consecutivos puede causar que el `Global Planner` genere trayectorias subóptimas o innecesariamente amplias, afectando la eficiencia del movimiento. Esto explica los desplazamientos zigzagueantes extensos y las trayectorias poco naturales observadas en la figura.

En la Figura 4, el robot logra recorrer más zonas de la habitación, pero la trayectoria sigue mostrando sobreposiciones y giros innecesarios. Este comportamiento desordenado puede deberse a un problema en las iteraciones del algoritmo de Coverage Path Planning, donde la selección de los siguientes puntos a visitar se vuelve ineficiente al avanzar el tiempo. La trayectoria tiende a desordenarse progresivamente, acumulando errores y provocando coberturas redundantes o incompletas, afectando directamente el desempeño del sistema.

En la Figura 5 se observa una mejora en la organización del recorrido, mostrando un patrón zigzagueante más estructurado. Sin embargo, el robot sigue dejando áreas sin cubrir, especialmente en las cercanías de los bordes. Esto se debe a la resolución de la grilla utilizada en el algoritmo, la cual era demasiado baja. Al tener celdas de mayor tamaño, el sistema considera como cubiertas zonas que en realidad no fueron recorridas completamente por el robot. A pesar de esta deficiencia, se aprecia que el algoritmo comenzó a funcionar de forma más ordenada en comparación con las pruebas anteriores.

La Figura 6 muestra el resultado exitoso tras el ajuste de la resolución de la grilla. Al disminuir el tamaño de las celdas, se logra una cobertura más detallada del espacio, y el robot recorre sistemáticamente todas las áreas libres siguiendo un patrón zigzagueante eficiente. Esta trayectoria representa un comportamiento satisfactorio y coherente con los objetivos del proyecto, simulando correctamente el funcionamiento de una aspiradora robot.

6.2. Habitación de Dos Ambientes

Luego de obtener resultados positivos en la habitación simple, se probó el algoritmo en un entorno más complejo con dos habitaciones conectadas por un pasillo. En la Figura 7, se observa cómo el robot recorre parcialmente la primera habitación, pero no logra transitar hacia la segunda. El problema se debe a que el próximo objetivo planificado por el algoritmo se encontraba justo sobre una pared separadora, lo cual hacía imposible que el robot alcanzara ese punto. Esta situación generaba colisiones repetidas contra la pared, impidiendo que el robot completara la cobertura de ambos espacios.

En la Figura 8, se realizó un ajuste en el umbral de satisfacción del algoritmo para que el robot considerara como alcanzado el punto problemático cerca de la pared, incluso sin llegar exactamente al objetivo. Esto permitió que el robot retrocediera tras chocar y planificara una ruta hacia la segunda habitación. Sin embargo, el robot no siguió efectivamente la trayectoria hacia la otra habitación, lo que podría deberse a un problema en los parámetros de Move Base o a errores en la implementación del algoritmo de *Coverage Path Planning*. Es posible que la representación de la grilla utilizada no identificara correctamente la conexión entre las dos habitaciones, lo que afectó la capacidad del sistema para navegar entre ambientes conectados.

7. Conclusiones

La implementación de una aspiradora robot en un entorno simulado permitió explorar e integrar diversos conceptos fundamentales de la robótica móvil. A lo largo del desarrollo del proyecto, se adquirió experiencia práctica en la construcción de mundos virtuales mediante Gazebo y en la implementación de algoritmos avanzados de navegación como GMapping, Move Base y AMCL. Cada uno de estos componentes resultó esencial para la operación del sistema:

- **GMapping** permitió la creación de un mapa del entorno desconocido, generando un `OccupancyGrid` que luego fue utilizado tanto para la planificación global como para el algoritmo de cobertura.
- **AMCL** se encargó de la localización continua y precisa del robot, combinando la odometría y los sensores láser, facilitando la navegación en tiempo real.
- **Move Base** coordinó la ejecución de las trayectorias mediante un planificador global y uno local, permitiendo que el robot avanzara hacia los objetivos generados por el algoritmo de Coverage Path Planning, evitando obstáculos dinámicos.

Respecto a los resultados obtenidos, el sistema mostró un desempeño satisfactorio en escenarios simples. En la habitación rectangular, tras ajustes progresivos en la resolución de la grilla, se logró una cobertura eficiente y ordenada del espacio, simulando adecuadamente el comportamiento esperado de una aspiradora robot. La trayectoria resultante presentó un patrón zigzagueante sistemático, cumpliendo con los objetivos planteados inicialmente.

Sin embargo, al aumentar la complejidad del entorno, los resultados dejaron de ser completamente satisfactorios. En la habitación de dos ambientes, el robot mostró dificultades para planificar rutas adecuadas al enfrentarse a paredes interiores y puertas estrechas. En particular, se observó que el algoritmo tendía a generar objetivos en zonas inalcanzables, como puntos ubicados directamente sobre paredes, lo que provocaba colisiones y bloqueos en la planificación. Además, la conexión entre las habitaciones no fue correctamente interpretada por la grilla utilizada en el Coverage Planner, lo que afectó la transición entre espacios.

Estas limitaciones evidencian la necesidad de realizar mejoras en el sistema. Entre las dificultades principales encontradas durante el desarrollo se destacan:

- La sincronización adecuada de los sensores en `rviz` con el entorno simulado en Gazebo, particularmente al configurar los tópicos de `LaserScan`, odometría y transformaciones TF.
- La definición correcta de la lógica detrás del algoritmo de *Coverage Path Planning*, incluyendo el diseño de la grilla y la estrategia para asignar pesos y recorrer el mapa.

- La configuración de los parámetros del sistema, especialmente aquellos asociados al `move_base`, como el tamaño de los costmaps, el umbral de satisfacción de objetivos y los parámetros de seguridad.

7.1. Trabajos Futuros

Como próximos pasos, se proponen las siguientes líneas de trabajo para mejorar el sistema desarrollado:

- **Mejorar la representación de la grilla**: Utilizar una resolución adaptativa o implementar un método de generación de grillas que considere de forma explícita las puertas y pasillos, para facilitar la navegación entre ambientes conectados.
- **Incorporar técnicas de planificación global más robustas**: Integrar algoritmos como *Wavefront* o *Voronoi Graphs*, que permiten una cobertura más estructurada y adaptada a entornos complejos.
- **Ajuste fino de parámetros de Move Base**: Afinar los parámetros del planificador global y local para mejorar la transición entre habitaciones y la evitación de obstáculos cercanos.
- **Incluir gestión de fallos**: Implementar mecanismos de replanificación en caso de bloqueos o fallos en la navegación, lo que permitiría aumentar la resiliencia del sistema frente a escenarios complejos.
- **Explorar algoritmos alternativos de cobertura**: Considerar métodos como *Spanning Tree Coverage* o *Boustrophedon Cellular Decomposition* para entornos con obstáculos interiores o particiones.

En resumen, el proyecto permitió construir una base sólida para el desarrollo de un sistema de cobertura en robótica móvil, integrando módulos de mapeo, localización y navegación. Aunque se obtuvieron resultados positivos en entornos simples, los desafíos presentados en escenarios complejos abren nuevas oportunidades para continuar mejorando y extendiendo el sistema, con aplicaciones reales en la robótica de servicio.