UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DESARROLLO DE UN SISTEMA DE *COVERAGE NAVIGATION* PARA UN ROBOT MÓVIL NO HOLONÓMICO

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

MICHELLE PAZ VALENZUELA CORTES

PROFESOR GUÍA:
JAVIER RUIZ DEL SOLAR SAN MARTÍN

PROFESOR CO-GUÍA:
FRANCISCO LEIVA CASTRO

COMISIÓN:
ANDRÉS CABA RUTTE

Este trabajo ha sido parcialmente financiado por Proyecto FONDECYT 1201170 y Proyecto ANID-PIA AFB230001

SANTIAGO DE CHILE
2024

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: MICHELLE PAZ VALENZUELA CORTES
FECHA: 2024
PROF. GUÍA: JAVIER RUIZ DEL SOLAR SAN MARTÍN

DESARROLLO DE UN SISTEMA DE *COVERAGE NAVIGATION* PARA UN ROBOT MÓVIL NO HOLONÓMICO

En el área de la robótica móvil, el concepto de *coverage navigation* se refiere a la tarea de cubrir un área o volumen. Esta tarea es un requerimiento esencial en múltiples aplicaciones, como por ejemplo desminado de un área, arado de un campo e incluso tareas cotidianas como la limpieza de un espacio. En esta memoria, se diseñó e implementó un sistema de navegación autónoma de cobertura para bases móviles con restricciones holonómicas, esto es, bases móviles que tienen restricciones de velocidad y por ende no pueden ejecutar ciertos movimientos. Para abordar el problema se separó la tarea en dos partes: el cálculo de rutas de cobertura mediante el uso de algoritmos de *coverage path planning* y el problema de navegación clásica. Este último problema consiste en resolver la tarea de navegación autónoma que busca que un robot pueda desplazarse por sí mismo de un punto a otro.

El enfoque de la solución consiste en el desarrollo de un árbol de comportamiento que permita el cálculo y ejecución de rutas que permitan cubrir el área de interés. La ruta es obtenida a través de un algoritmo de *coverage path planning* y la ejecución del plan de navegación se realiza mediante un *stack* de navegación. Es importante mencionar que el sistema requiere de una representación métrica a priori del espacio a cubrir, dígase, un mapa. Dado esto, ante cambios en el espacio el sistema podría no cubrir efectivamente el área, por lo cual el árbol debe incorporar mecanismos que permitan al sistema adaptarse para cumplir la tarea de cobertura de manera robusta.

El sistema desarrollado fue evaluado en simulación y desplegado en el mundo real, obteniéndose resultados positivos. El sistema desarrollado demuestra una capacidad efectiva para cubrir distintos ambientes, siendo robusto a imprevistos como obstáculos no identificados en el mapa del ambiente, pudiendo evadirlos y generar replanificaciones que permitan dar cobertura a las áreas que no se pudieron cubrir debido a estos.

Agradecimientos

En primer lugar, agradezco sinceramente a mi profesor guía, Javier Ruiz del Solar, por brindarme la oportunidad de explorar este fascinante tema. Su apoyo y confianza en mi capacidad fueron cruciales en este proceso.

A Francisco Leiva, co-guía de este trabajo, le agradezco inmensamente por su compromiso. Su ayuda en la realización de experimentos y las enriquecedoras discusiones sobre diversos aspectos de la memoria fueron esenciales para el éxito de este trabajo.

Un reconocimiento especial para Pablo Alfessi, quien diseñó y construyó los montajes para los robots y también me otorgo su apoyo en la realización de las pruebas en el mundo real.

Quiero extender mi agradecimiento de manera general a todos mis amigos y compañeros que me han acompañado a lo largo de mi estancia en la universidad. Su amistad y colaboración han sido un motor constante de motivación que me ha llevado hasta este punto.

Adicionalmente, quiero expresar mi agradecimiento al laboratorio de robótica de la universidad de Chile ya que fue allí donde mi fascinación por la robótica se despertó, marcando el inicio de mi camino en este campo.

A todos ellos, mi más sincero agradecimiento por su contribución directa o indirecta a este proyecto y por haber sido parte fundamental de mi trayectoria académica.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Contexto

La robótica es una disciplina que integra diversas áreas del conocimiento como ingeniería mecánica, ingeniería eléctrica y ciencias de la computación, para diseñar, programar, construir y operar robots, es decir, máquinas que son capaces de realizar tareas predeterminadas de manera autónoma.

En la actualidad, los robots están presentes en muchos sectores productivos. Es común observar a estas máquinas en el área de manufactura([1], [2], [3]), en agricultura([4], [5], [6]) e incluso en el área de medicina ([7], [8], [9]). Más aún, podemos encontrar robots en nuestro diario vivir. Un claro ejemplo de esto son las aspiradoras automáticas (e.g. una *roomba*). Estos dispositivos tienen como propósito limpiar un área, para lo cual deben cubrirla navegando a través de ella. En robótica, específicamente en robótica móvil, a esta tarea de cubrir un área o volumen se le denomina *coverage navigation* [10].

Para poder llevar a cabo la cobertura de un área, un robot necesita poder resolver primero el problema de moverse de manera autónoma desde un punto a otro. Este problema corresponde al problema de navegación clásico y ha sido ampliamente abordado en la literatura debido a su fundamental importancia en prácticamente toda aplicación robótica, incluyendo el problema de la cobertura de un área. Un enfoque ampliamente usado para abordar este problema es la descomposición de este en múltiples sub-problemas. Un ejemplo de descomposición corresponden a mapeo y localización, planificación de rutas y planificación local. El mapeo y la localización permiten al robot conocer su posición en el espacio. El mapa corresponde a una representación del ambiente en el cual se encuentra, esta puede ser de distinto tipo como una representación métrica o una topológica, y el sistema de localización permite estimar cual es la pose, posición y orientación, del robot en el mapa. Por otra parte, para la planificación de rutas se tienen dos componentes, un planificador de rutas global que calcula una ruta a partir de información ya conocida (mapa) y un planificador local que permite ejecutar y adaptar la ruta a medida que esta es ejecutada. Esto último es necesario ya que en general los ambientes donde se desempeñan los robots están en constante cambio y por ende es necesario que el robot pueda por ejemplo evadir obstáculos que no conocía previamente. A este conjunto de componentes se le denomina *stack* de navegación.

Para poder integrar la componente de cobertura en el problema de navegación robótica se utilizan algoritmos de *coverage path planning* (CPP), los cuales nos permiten calcular rutas que permiten al robot cubrir un área de interés. En un *stack* de navegación clásico, estos algoritmos actuarían de manera similar a un planificador global.

Es importante mencionar que los planificadores pueden tener algún tipo de dependencia respecto a la cinemática del robot. Esto se debe a que según la cinemática del robot este puede o no ejecutar ciertos movimientos y por tanto el planificador puede tener las distintas restricciones existentes en cuenta. Según las restricciones que se tenga un robot puede ser holonómico o no holonómico. Un robot holonómico es aquel que puede moverse libremente en el plano (x,y) debido a que sus grados de libertad y controlabilidad son iguales. Por el contrario un robot no holonómico no es capaz de moverse libremente en el plano debido a que sus grados de libertad y controlabilidad se encuentran limitados por restricciones de velocidad. Un ejemplo de vehículo móvil no holonómico son los autos, estos poseen restricciones de velocidad que no le permiten realizar ciertos movimientos como desplazamientos laterales. Es importante mencionar que el que un robot sea no holonómico no tiene como consecuencia que no pueda alcanzar una cierta pose en el espacio. Siguiendo con el ejemplo del automóvil, si esto fuera cierto significaría que es imposible estacionar un este en paralelo, lo cual es en efecto posible pero complejo debido a que se deben ejecutar diversas maniobras para poder colocar el automóvil en esa pose. Lo anterior permite ilustrar la importancia de tener en consideración el tipo de base móvil que se utilizará en una aplicación robótica debido a que los comandos de velocidad que se le entregan al robot deben ser apropiados para que la base pueda alcanzar una pose objetivo.

El desarrollo de un sistema de navegación de cobertura o *coverage navigation* resulta interesante, ya que permite abordar el problema de cubrir un área que es un requisito esencial para la realización de diversas labores. Desde tareas cotidianas, como la limpieza de una habitación o el corte del césped a problemáticas mas complejas y desafiantes como el desminado de áreas peligrosas, cosecha y arado de campos agrícolas, así como exploración y reconocimiento de sitios. El desarrollo del sistema permite que un robot aborde de manera autónoma la tarea de cubrir un área específica de manera completa y sistemática lo cual permite no solo mejora la eficiencia con la cual la tarea se realiza, sino que también reduce los riesgos asociados a actividades peligrosas, como el desminado.

En esta memoria se desarrollará un sistema de navegación de cobertura o *coverage navigation* para ambientes de exterior. Al tomar este enfoque se debe tener en consideración las distintas implicaciones que surgen al exponer al robot a un ambiente de exterior. Las diferencias fundamentales entre entornos de interior y exterior requieren ser abordadas en el diseño de la solución. Los robots que operan en exteriores se ven expuestos a condiciones ambientales más hostiles, enfrentando terrenos irregulares, espacios amplios y poco estructurados, así como condiciones climáticas adversas como viento, lluvia y polvo, entre otros desafíos.

El sistema que se desarrollará estará destinado a bases móviles no holonómicas, específicamente a aquellas de tipo *skid-steered*. Es importante tener en cuenta esta elección, ya que, como se mencionó anteriormente, el tipo de base móvil puede influir en el desarrollo del sistema de navegación al estar sujeta a restricciones de velocidad particulares.

El sistema propuesto contará con un módulo de decisiones de alto nivel basado en el uso de *behavior trees* [11], que dicta el comportamiento del robot a través del control de un *stack* de

navegación. Este behavior tree permitiría, por ejemplo, identificar instancias donde es necesario realizar una re-planificación global con un algoritmo de *coverage path planning*, para cubrir una región que no pudo ser cubierta previamente debido a la presencia de un obstáculo dinámico que fue evadido por el planificador local. El *stack* de navegación incluye la componente de localización, planificación de rutas del robot (mediante un algoritmo de CPP), y de planificación local.

1.2. Objetivos

1.2.1. Objetivo general

El objetivo de este trabajo es desarrollar un sistema de *coverage navigation* para un robot no holonómico en ambientes de exterior.

1.2.2. Objetivos específicos

Para poder conseguir el objetivo principal de esta memoria se han planteado los siguientes objetivos específicos:

- Implementar y/o integrar diferentes algoritmos de *coverage path planning* en simulación.
- Evaluar los algoritmos de *coverage path planning* considerados eligiendo los de mejor desempeño.
- Integrar los algoritmos en un *stack* de navegación que incluya un sistema de localización y planificación local.
- Adaptar y probar en el robot no holonómico objetivo, y optimizar desempeño en simulaciones.
- Desplegar y evaluar el sistema de *coverage navigation* en el mundo real.

1.3. Alcances de este trabajo

El sistema de navegación de cobertura desarrollado se validó en una base de tipo *skid-steered*, particularmente se utilizó la base móvil Panther. El uso en otro tipo de base, por ejemplo omnidireccional o diferencial, tiene como implicancia que el sistema deberá ser adaptado en los componentes que corresponda, por ejemplo, el planificador local.

Adicionalmente, este sistema se desarrolló para ambientes de exterior por lo cual podrían requerirse modificaciones en el sistema para un apropiado uso en ambientes de *indoor* los cuales tienen características de distinta naturaleza, siendo común la presencia de humanos, paredes que delimitan claramente el espacio, entre otros.

Finalmente, este sistema fue desplegado en ambientes de exterior controlados, es decir, estos eran acotados y predecibles, a diferencia de lo que podría suceder al desplegar en un ambiente mas hostil como un terreno agrícola o un sitio de construcción. Además, los entornos usados tanto en simulación como en el mundo real ya eran conocidos previamente por el sistema mediante un mapa métrico del espacio a cubrir. Por tanto, el despliegue del sistema en ambientes donde no se cuenta a priori con un mapa y se deba utilizar alguna técnica de SLAM (*Simultaneous Localization And Mapping*) quedan excluidos de este trabajo.

1.4. Estructura del documento

El presente documento tiene la estructura que se presenta a continuación:

- Capitulo 2: Se exponen los conceptos base requeridos para poder entender el problema a resolver, por ejemplo, que se entiende por navegación robótica y por *coverage navigation*.
- Capitulo 3: se muestra el estado del arte de soluciones y aplicaciones relacionadas a *coverage navigation*. Se presentan aplicaciones donde se ha desarrollado un sistema de este tipo y se presentan trabajos desarrollados donde se han implementado algoritmos de *coverage path planning* los cuales son esenciales para el funcionamiento del sistema.
- Capitulo 4: Se presenta el diseño e implementación de la solución la cual es dividida en dos partes: *stack* de navegación y *behavior tree*.
- Capitulo 5: Se exponen los resultados obtenidos tras someter a distintas evaluaciones la solución desarrollada. Se desarrolla un *benchmark* para comparar los algoritmos de CPP a utilizar y se evalúa el sistema tanto en simulación como en el mundo real.
- Capitulo 6: Se presentan las conclusiones que derivan del trabajo desarrollado y evaluado en esta memoria, además se presentan posibilidades de mejora para el sistema.

Capítulo 2

Marco teórico

En el presente capítulo se describirán distintos conceptos clave y su importancia para lograr entender y abordar correctamente el problema a resolver. El capítulo comienza explicando la navegación robótica, concepto clave para resolver la tarea de *coverage navigation*, el cual corresponde a un problema particular de navegación. También se incluye el concepto de robot *skid-steered*, cuya importancia radica en la influencia de la cinemática del robot para poder resolver el problema de navegación y por ende también el problema de cobertura. Finalmente se define el concepto de *behavior tree* el cual es clave para entender la solución que se plantea en el Capítulo 4.

2.1. Navegación robótica

En robótica, la tarea de navegación se refiere a la capacidad de un robot de poder desplazarse desde una pose a otra de manera autónoma. Este tópico ha sido ampliamente abordado en la literatura debido a su vital importancia en el desarrollo de aplicaciones autónomas, ya que que para poder desarrollar una tarea es muy común que un robot requiera desplazarse a distintos lugares.

El enfoque clásico con el cual este problema es abordado se basa en la división de la tarea en distintos elementos, los cuales se encargan de distintos sub-problemas que son necesarios para poder resolver la tarea de navegación. Este enfoque comprende la representación del espacio mediante un mapa, la estimación de la ubicación del robot respecto al mapa y los planificadores global y local que permiten la planificación de trayectorias y evasión obstáculos a largo y corto plazo respectivamente.

2.1.1. Componentes de un *stack* de navegación

Un *stack* de navegación corresponde a una agrupación de componentes que trabajan en conjunto para permitir a un robot moverse de manera autónoma. Los componentes de un *stack* son principalmente fuentes de información (sensores), un sistema de mapeo, un sistema de localización, un sistema de control, un planificador global y un planificador local.

Los sensores son un elemento clave en un *stack* de navegación puesto que proporcionan al robot información de su entorno en tiempo real. Esta información es requerida por el sistema de

localización que utiliza estos datos para estimar la pose del robot en el mapa y por el planificador local que utiliza esta información para detectar cambios en el entorno, como obstáculos dinámicos, y realiza ajustes en la ruta. Algunos ejemplos de sensores ampliamente usados son LiDARs 2D y 3D, radares, cámaras RGB, RGB-D y estéreo, etc.

Para que el robot sea capaz de estimar su pose en el mundo se necesitan dos cosas: un mapa, el cual tomar como referencia, y un sistema (algoritmo) de localización para poder estimar su pose en el mapa. Dada la existencia o inexistencia de un mapa (a priori) del entorno, existen dos casos: localización y mapeo simultáneo, o solo localización (tras la realización de un mapeo, o dado un mapa preexistente). Para el caso en que no existe una representación previa del entorno, i.e. no hay un mapa, es posible mediante la utilización de algoritmos de SLAM (*Simultaneous Localization And Mapping)*, como *Fast*SLAM [12], ORBSLAM [13], RTAB-Map [14], entre otros, realizar ambas tareas al mismo tiempo. Para el caso en el que se tiene un mapa, es posible estimar la pose del robot mediante algoritmos que emplean filtros de Kalman (e.g [15, 16, 17, 18, 19]) o filtros de partículas[20, 21, 22, 23, 24] como parte del algoritmo de localización, entre otras posibilidades.

Para poder ir desde una pose inicial hasta una pose destino, el robot necesita encontrar una ruta. De esta tarea se encarga el planificador global. Para lograr esto utiliza la información del entorno contenida en el mapa y algún algoritmo de planificación como A* [25] o Dijkstra [26], con el cual encontrar una ruta óptima.

Es importante mencionar que al seguir la trayectoria propuesta por el planificador global pueden ocurrir imprevistos, como la aparición de obstáculos no identificados previamente, esto es, que no aparecen en el mapa del entorno. Para poder sortear estas dificultades y llegar al objetivo de navegación sin colisionar, se utiliza un planificador local que, mediante la información proveniente de los sensores del robot, puede observar los cambios y controlar al robot usando esta información como retroalimentación.

Es fundamental reconocer que el planificador local se encuentra estrechamente vinculado al tipo de base móvil del robot, debido a que según el tipo de base móvil habrán ciertas maniobras que el robot podrá o no ejecutar, como se verá en la Sección 2.3. La naturaleza de la plataforma móvil impacta directamente en las capacidades de movimiento del robot. Por lo tanto, el planificador local debe considerar las características y restricciones propias del tipo de la base móvil al generar instrucciones. Esto es crucial para garantizar que los comandos emitidos sean compatibles con las capacidades particulares de la base móvil.

2.2. *Coverage navigation*

El problema de *Coverage navigation* consiste en cubrir un área o volumen de interés de manera autónoma. A grandes rasgos, esta tarea es similar a la tarea de navegación, solo que en lugar de haber un único objetivo de navegación, aquí lo que se busca es poder cubrir toda un área. *Coverage navigation* es una parte esencial en la ejecución de múltiples aplicaciones, por ejemplo, en tareas de exploración [27], arado de campos [28], cosecha [4] [29], cortadoras de césped [30], desminado [31], limpieza [32] [33], entre muchas otras. Si bien la cobertura puede darse también en tres dimensiones (como por ejemplo, [34, 35, 36, 37]), el trabajo abordado en esta memoria se limitará a la cobertura en dos dimensiones, es decir, cubrir un plano.

Para lograr llevar a cabo la tarea de cobertura de un espacio se utiliza un *stack* de navegación con los mismos componentes mencionados anteriormente (ver Sección 2.1.1). La diferencia en este caso se encuentra en que no se quiere una trayectoria para llegar a un destino especifico, sino que se requiere del cálculo de una trayectoria que permita cubrir el espacio. Para lograr esto, se utilizan algoritmos de *coverage path planning* que vendrían a tomar el lugar de planificador global en el *stack* de navegación.

2.2.1. *Coverage path planning*

Teniendo en cuenta la amplia variedad de aplicaciones en las que se requiere *coverage navigation* para llevar a cabo tareas específicas, no es sorprendente que haya un extenso estudio de algoritmos de *coverage path planning*, ya que son esenciales para ejecutar dichas aplicaciones de manera efectiva. Este tipo de algoritmos pueden ser clasificados según el enfoque con el cual dividen el espacio a cubrir. A continuación se listan las distintas clasificaciones según Choset [38]:

- **Métodos de descomposición en celdas exacta:**
  Los algoritmos pertenecientes a esta categoría tratan el problema de cobertura en tres pasos: división del espacio en celdas, búsqueda de ruta de visita óptima de las celdas y cubrimiento de celdas.

  Este tipo de algoritmo busca dividir el espacio libre de obstáculos en celdas de distinto tamaño y forma, tal que estas celdas no se superpongan y al unirse conformen el espacio libre original. Un ejemplo de celdas obtenidas con este tipo de descomposición se puede apreciar en la Figura 2.1a. Cada una de las celda usualmente es cubierta con movimientos simples, usualmente movimientos en zig-zag.

  Para decidir en qué orden las celdas generadas se visitan se utilizan grafos de adyacencia [39] para representar el espacio dividido. Luego, el problema se reduce a encontrar el camino más corto para visitar cada celda, lo cual lo convierte en el clásico problema del vendedor viajero [40]. Este camino puede calcularse mediante algún algoritmo como *Depth-first Search* [41].

  Algoritmos que pertenecen a esta categoría son descomposición trapezoidal [42], descomposición Boustrophedon [43] y descomposición con funciones Morse [44].
- **Métodos de descomposición aproximada:**
  Los algoritmos de cobertura que utilizan este modo de descomposición dividen el espacio en celdas uniformes de igual tamaño y forma, como se aprecia en el ejemplo de la Figura 2.1c. Es usual que las celdas sean de forma cuadrada y de tamaño similar al robot a utilizar ya que usualmente los algoritmos que utilizan esta descomposición asumen que el robot ha cubierto la celda una vez entra a ella. Sin embargo, es posible elegir otras formas, como triángulos y un distinto tamaño de resolución.

  Existe una amplia variedad de algoritmos que utilizan este tipo de representación como base. Esto se debe a que es sencillo crear un mapa de grilla y es sencillo marcar las áreas cubiertas. También es importante destacar que las representaciones en grilla son costosas, pues existe un crecimiento exponencial en términos de memoria al agrandar el tamaño del mapa. Lo anterior se debe a que la resolución se mantiene constante sin importar la complejidad del área de interés.

  Algunos ejemplos de algoritmos que pertenecen a esta categoría son *Backtracking Spiral Algorithm* (BSA) [45], *Spanning Tree Covering* (STC) [46] y *Wavefront Algorithm* [47].

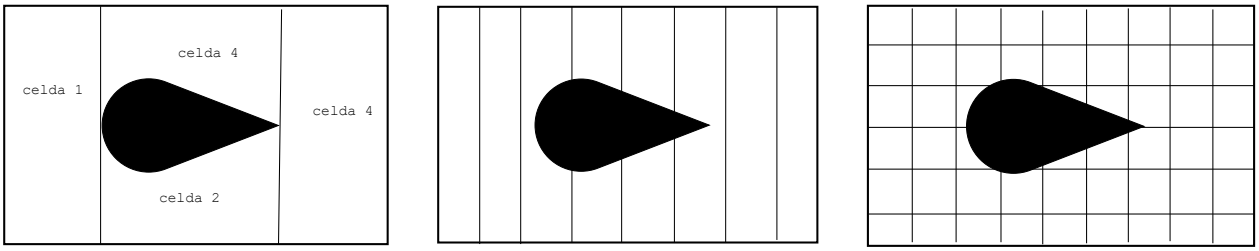


Figura 2.1: Ejemplos de celdas obtenidas según distintos tipos de descomposición. En la imagen, el espacio en negro representa presencia de obstáculo y el espacio en blanco representa espacio libre que el robot puede cubrir. (a) Celdas obtenidas al realizar una descomposición exacta. (b) Celdas obtenidas al realizar una descomposición semi exacta. (c) Celdas obtenidas al realizar una descomposición aproximada.

- **Métodos de descomposición semi exacta:**
  Este tipo de algoritmos divide el espacio de forma semi aproximada mediante el uso de celdas de ancho fijo y forma variable. Para realizar la descomposición, el área se divide con líneas equidistantes de manera que cada división será una celda con bordes inferior y superior arbitrario, como se muestra en la Figura 2.1b. Este tipo de enfoque fue utilizado por Lumensky en [48] y Hert en [49].
- **Métodos Heurísticos o aleatorizados:**
  El enfoque utilizado en este tipo de algoritmos se basa en el uso de una regla general, que se caracteriza por el uso de experiencias pasadas. Estos pueden ser de tipo aleatorios o estar gobernados por algún tipo de heurística usualmente bio-inspirada [50]. Es importante mencionar que este tipo de algoritmo no garantiza una cobertura completa del área, y además puede tomar mucho tiempo en lograr la cobertura. Dado lo anterior, este tipo de enfoque no es comúnmente usado en aplicaciones reales y sus resultados se limitan mas bien a simulaciones.
- **Métodos multi-robots:**
  Este tipo de algoritmo busca dividir la tarea de cubrir un área utilizando varios robots simultáneamente. El uso de más de un robot no solo disminuye el tiempo en que la tarea se completa, sino que también ayuda a amortiguar el error, ya que los robots pueden apoyarse mutuamente en caso que algún miembro experimente un fallo.

  Los algoritmos pertenecientes a esta categoría usualmente extienden las ideas de los algoritmos de *coverage path planning* para un robot a múltiples robots mediante el uso de alguna estrategia para dividir el trabajo. También existen algoritmos que tratan el problema de múltiples robots directamente. Algunos ejemplos donde se ha visto el enfoque de utilizar múltiples robots son [51, 52, 53, 54, 55].

2.3. Robot *skid-steered*

Un robot está compuesto de diversas partes que le permiten ejecutar distintas funciones, por ejemplo brazos y *gripper* para manipular objetos, cara para expresar emociones, sensores como LiDARs y cámaras que le permiten obtener información de su entorno, base móvil para desplazarse, entre otros. Particularmente, para las bases móviles, hay de distintos tipos, por ejemplo se tienen bases móviles con ruedas y bases cuadrúpedas e incluso bípedas.

Las robots móviles con ruedas pueden ser clasificados en dos grupos: robots holonómicos y robots no holonómicos. Los robot holonómicos son aquellos en las que el número de grados de controlabilidad es mayor o igual al número de grados de libertad. Por este motivo son capaces de moverse libremente en cualquier dirección, ya que son capaces de controlar los tres grados de libertad que caracterizan al movimiento en un plano, dígase desplazamiento en x e y y rotación en torno a z. Un ejemplo de robot holonómico son los de tipo omnidireccionales. Por otra parte, los robots no holonómicos están sujetos a restricciones cinemáticas, por lo cual el número de grados de controlabilidad es menor al número de grados de libertad, lo que implica que estos no pueden ejecutar cualquier tipo de movimiento en el plano. En este grupo podemos encontrar varios tipos de robots móviles, por ejemplo, los de tipo *skid-steered* y los de tipo diferencial [56].

Los robot *skid-steered* se caracterizan por tener pares de ruedas u orugas paralelas ubicadas en cada lado del robot. Estas ruedas son controladas de manera independiente. Para poder desplazarse se varía la velocidad de las ruedas de forma tal que para ir en línea recta las ruedas de ambos lados deben ir a la misma velocidad, y para girar es necesario que la velocidad de las ruedas de lados opuestos sea diferente [57].

Este tipo de robot móvil posee una alta maniobrabilidad y tracción, lo cual los hace apropiados para aplicaciones en terrenos irregulares o complejos. Su alta maniobrabilidad se explica por su radio de curvatura cero, lo que se traduce en que el robot es capaz de girar sobre su propio eje y por ende puede maniobrar en lugares con espacios estrechos. Es importante mencionar que al realizar maniobras de giro usualmente existe derrape, esto es, un desplazamiento no controlado de las ruedas.

Por otra parte, la alta tracción de este tipo de plataforma se debe a que usualmente las ruedas están acopladas directamente al motor, por lo cual no existen elementos intermedios, como engranajes, que causen pérdidas. Además, en general este tipo de robot suele utilizar ruedas u orugas con una amplia superficie de contacto, por lo cual hay mayor área de fricción con el suelo [58].

Las bases tipo *skid-steered* comparten similitudes con las bases de tipo diferencial en términos de su capacidad de movimiento y maniobrabilidad. Ambos tipos de bases son capaces de girar en torno a su propio eje, variando de manera independiente la velocidad de sus ruedas o pares de ruedas opuestas. Sin embargo, a pesar de haber similitudes, existen diferencias notables a nivel cinemático ya que las bases *skid-steered* presentan derrape al realizar maniobras de giro. Dado esto, es posible utilizar los sistemas de planificación de trayectorias existentes para robots diferenciales en robots *skid-steered* realizando algunas adaptaciones para considerar las diferencias existentes entre estos tipos de bases [59].

2.4. *Behavior tree*

Un árbol de comportamiento o *Behavior tree* es una forma de estructurar el cambio entre distintas tareas que conforman una actividad. Este tipo de estructura es altamente eficiente y permite crear sistemas complejos modulares y reactivos. En esta sección serán descritos los árboles de comportamiento o *behavior trees* según como se detalla en [60].

Formalmente, un Árbol de Comportamiento se define como un árbol dirigido con raíz. En esta estructura, los nodos del árbol están conectados por arcos, y la dirección de estos arcos define el

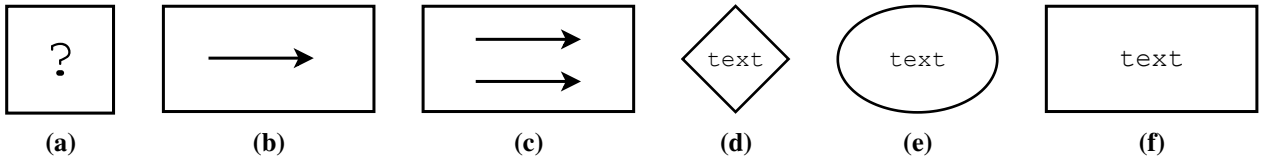


Figura 2.2: Simbología nodos de un *behavior tree*. (a) Alternativa. (b) Secuencia. (c) Paralelo. (d) Decorador. (e) Condición. (f) Acción.

flujo de ejecución. Los nodos internos del árbol de comportamiento, llamados nodos de flujo de control, dirigen la lógica del comportamiento, mientras que los nodos hoja, denominados nodos de ejecución, representan acciones específicas. La presencia de una raíz implica que el flujo del árbol es originado en un nodo padre. Este nodo raíz genera señales denominadas *ticks*, los cuales permiten la ejecución de los nodos hijos.

En la formulación clásica de este tipo de arboles se definen cuatro tipos de nodos de flujo de control. Estos corresponden a nodos de tipo secuencia, paralelo, decorador y alternativa. Por su parte, para los nodos de ejecución se tienen dos tipos: condición y acción. Un nodo, sea de control o ejecución, puede retornar tres estados diferentes: éxito, fracaso y en ejecución. Cada uno de estos nodos tiene un símbolo asociado que los representa. Estos pueden verse en la Figura 2.2.

Un nodo de secuencia propaga el *tick* a sus hijos de manera secuencial, empezando con el hijo más a la izquierda. Si este retorna éxito, el *tick* se propaga al siguiente hijo a la derecha y se sigue así de manera recursiva. Este nodo retornará éxito si y solo si la ejecución de todos los hijos fue exitosa. Si uno de los nodos hijos retorna fracaso, el nodo adquirirá este mismo estado y el resto de los nodos hijos no se ejecutarán. En el caso en que uno de los hijos retorne en ejecución, el nodo de secuencia retornará el mismo estado y la ejecución del resto de los hijos estará suspendida hasta que el nodo hijo en cuestión retorne éxito o fracaso.

Los nodos de tipo paralelo propagan el *tick* de manera simultánea a todos sus nodos hijos. La ejecución será exitosa si y solo si un determinado número de nodos retorna éxito. Este número (*threshold*) es definido por el usuario. En caso de que el número de hijos que retorna fracaso sea mayor al número total de hijos menos el valor del *threshold*, el nodo retornará fracaso. En otro caso, su estado será en ejecución.

En el caso de los nodos de tipo decorador se tienen distintas posibilidades de comportamiento. Estos nodos tienen un único hijo sobre el cual realizan manipulaciones sobre el estado que devuelve. El comportamiento del nodo es definido por el usuario. Por ejemplo, un nodo decorador puede invertir el estado del nodo hijo, retornar éxito cuando el nodo hijo esta en ejecución y fracaso en cualquier otro caso, etc.

Los nodos de alternativa propagan el *tick* de igual forma que los nodos de tipo secuencial. La diferencia se encuentra en las condiciones bajos las cuales se devuelve éxito, fracaso o en ejecución. Este nodo devolverá fracaso si y solo si todos sus hijos retornan fracaso. En caso contrario devolverá el mismo estado que el nodo hijo que este en evaluación, lo cual puede ser éxito o en ejecución.

Para los nodos de ejecución, en el caso de los nodos de acción se tiene que estos definen los comportamientos del árbol. Estos corresponden a acciones ejecutables, por ejemplo, agarrar un

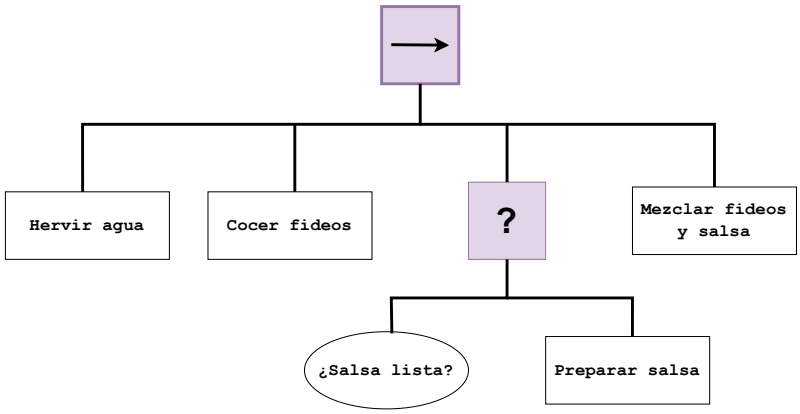


Figura 2.3: Ejemplo de un *behavior tree* simple.

objeto, mover un objeto, depositar un objeto, etc. Por otro lado, los nodos de tipo condición evalúan el estado de una condición definida por el usuario, por ejemplo ¿tengo el objeto? ¿la puerta está abierta? ¿estoy en la cocina?, etc.

La forma más sencilla de entender como se estructura un *behavior tree* es mediante un ejemplo. Tomemos como base la actividad cotidiana de hacer fideos con salsa. Para lograr hacer el platillo es necesario realizar una serie de pasos, los cuales en un árbol de comportamiento se estructurarían en un nodo de tipo secuencia. Los pasos a seguir, de manera simplificada, serían hervir agua, luego cocinar los fideos, preparar la salsa y finalmente mezclar ambas cosas. Para ir un poco mas allá en el diseño del árbol, podemos agregar un nodo alternativa en el paso de la salsa, donde nos preguntemos si la hemos dejado esta lista previamente, y en caso de no ser así, prepararla. El árbol resultante puede observarse en la Figura 2.3. De esta misma manera se pueden ir detallando más y más cada paso, obteniéndose arboles con mayor cantidad y tipos de nodo.

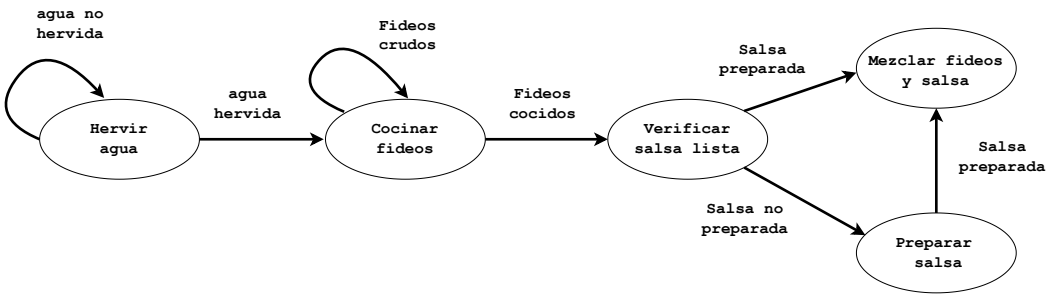


Figura 2.4: Versión del *Behavior tree* de la Figura 2.3 como máquina de estados.

Otra herramienta con la cual se estructura la ejecución de tareas son las máquinas de estados. El enfoque utilizado para abordar el problema en este caso es mediante el uso de estados discretos y transiciones. Aunque las máquinas de estados han sido ampliamente adoptadas, el uso de árboles de comportamiento presenta notables ventajas en comparación con el uso de máquinas de estados, ya que estas estructuras permiten una mayor escalabilidad al ser más sencillas de gestionar. Es importante mencionar que ambos enfoques son ampliamente utilizados, y siempre es posible convertir una máquina de estado en su equivalente árbol de comportamiento y viceversa. Por ejemplo, en la Figura 2.4 se presenta la versión en máquina de estados del *behavior tree* de la Figura 2.3.

Capítulo 3

Estado del arte

A través de este capítulo se exponen algunos ejemplos recientes de aplicaciones donde se ha desarrollado un sistema de *coverage navigation*, esto nos permite dar cuenta claramente de la relevancia que tiene resolver este problema ya que como se observará tiene aplicaciones en distintos campos. Posteriormente, se detallan implementaciones de código abierto de algoritmos de *coverage path planning* que servirán como base para el desarrollo de la solución propuesta en el Capítulo 4.

3.1. Aplicaciones actuales de *coverage navigation*

Dar cobertura a un determinado espacio es un requerimiento que surge en una variedad de campos y aplicaciones. Es posible encontrar este requerimiento en tareas que tienen como objetivo la limpieza de un lugar ([61, 62, 63]), en tareas de exploración o reconocimiento de áreas([64, 65]), en tareas relacionadas a agricultura ([4, 5, 6]), tareas de inspección ([66, 67, 68]), entre otras.

Se han visto diversos desarrollos recientes en el área de agricultura. En [4] se desarrolló un sistema de *coverage navigation* para automatizar, mediante el uso de robots, la cosecha de kiwis. En este trabajo se utilizó un enfoque basado en aprendizaje reforzado para el cálculo de rutas óptimas que permitan cubrir el espacio de recolección. La estrategia utilizada se basó en modelar el problema de cobertura como el clásico problema del vendedor viajero, utilizando descomposición aproximada del espacio mediante división en grilla. En [69], se desarrolló un sistema de cobertura para un tractor de labranza que se desempeñaría en campos de arroz. En este trabajo, el sistema divide el problema de cobertura en interno y externo. El problema de cobertura externo corresponde a rodear los bordes del área de interés y la cobertura interna corresponde al resto del área que queda al interior de los bordes. Adicionalmente se implementan estrategias para la maniobrabilidad del tractor en esquinas. Esto fue necesario para mejorar el desempeño del sistema ya que la máquina posee maniobrabilidad limitada, lo cual le dificultaba cubrir eficazmente zonas con bordes. En [70] se implementó un sistema de cobertura similar al anteriormente descrito. La aplicación se desarrolló para rodillos de arcilla autónomos que se desempeñan en campos de cultivo de sal.

Además de agricultura, se han visto esfuerzos por desarrollar este tipo de sistemas en otros campos de aplicación. Tal es el caso de lo que se expone en [71] donde la aplicación desarrollada tiene como objetivo realizar exploración de suelo submarino. En este caso el ambiente en el cual

será expuesto el vehículo, el cual es de tipo submarino, es desconocido y altamente complejo. Para enfrentar las adversas condiciones bajo la cual se quiere desplegar el sistema se desarrolla un algoritmo que recalcula constantemente la ruta de cobertura, lo cual permite asegurar un nivel de cobertura alto y óptimo, en el sentido de tiempo en que toma realizar la cobertura.

Siguiendo con aplicaciones en ambientes marítimos, en [64] se propuso un marco de planificación rutas de cobertura para embarcaciones que utiliza aprendizaje reforzado profundo para mejorar la eficiencia en la búsqueda y rescate de personas sobrevivientes. Este enfoque combina modelos de predicción de trayectorias de deriva, un modelo de entorno de área de búsqueda jerárquico y una búsqueda de cobertura basada en aprendizaje reforzado profundo, lo cual permite a los agentes del barco priorizar regiones de alta probabilidad de encontrar sobrevivientes, evitando la cobertura repetida y mejorando significativamente la probabilidad de rescate dentro de un marco de tiempo limitado.

Otro tipo de ambiente donde se han visto desarrollos de *coverage navigation* es en ambientes aéreos como se observó en [72]. En este trabajo se desarrolló un sistema de navegación de cobertura para drones cuyo propósito es el mapeo de un área. Para esto, el dron debe recolectar fotografías con las cuales se busca reconstruir el terreno de interés. Para lograr esto se requiere de un plan de cobertura que le indique al robot que secuencia de poses seguir para lograr el objetivo. Para resolver el problema de la ruta de cobertura, este trabajo propone como solución dividir el área en celdas de igual tamaño y forma, asignando cada celda como un *waypoint* de navegación. El orden de visita de estas poses se decide mediante la resolución del problema del vendedor viajero, es decir, cada *waypoint* representa una ciudad a visitar (ver algoritmo del problema del vendedor viajero usando grillas, Subsección 3.2.1). Adicionalmente, para ahorrar en términos de tiempo de ejecución, lo cual es crítico debido a la batería limitada de los robot tipo dron, se propone un método de suavizado de plan, lo cual permite obtener planes más simples y por ende más rápidos y de fácil ejecución para el robot.

3.2. Implementaciones (código abierto) de algoritmos de CPP

A pesar del hecho de que el problema de *coverage navigation* ha sido ampliamente estudiado y hay una gran cantidad de publicaciones al respecto, no hay una gran cantidad de implementaciones de código abierto a disposición [73]. A continuación se listan algunas de las implementaciones *open source* disponibles:

- `CoveragePlanning`[1]
- `full_coverage_path_planner`[2]
- `polygon_coverage_planning` [3]
- `bousthophedon_planner`[4]
- `coverage.planning`[5]

[1] `https://github.com/RJJxp/CoveragePlanning`
[2] `https://github.com/nobleo/full_coverage_path_planner`
[3] `https://github.com/irvingvasquez/ocpp`
[4] `https://github.com/Greenzie/boustrophedon_planner`
[5] `https://github.com/Ipiano/coverage-planning`

- `Fields2Cover`[6]
- `ipa_coverage_planning`[7]

En esta lista, podemos destacar al repositorio `ipa_coverage_planning`, mas específicamente, al paquete `ipa_room_exploration`, cuyo código proporciona seis planificadores de rutas de cobertura, los cuales corresponden a algoritmos que han sido ampliamente usados en la literatura. Dado esto, se decide utilizar esta implementación en el desarrollo de este trabajo, con el fin de poder integrar y evaluar distintos algoritmos al sistema de *coverage navigation*.

3.2.1. Algoritmos contenidos en `ipa_room_exploration`

El código contenido en el repositorio `ipa_coverage_planning` deriva de una aplicación desarrollada en [74] , donde se busca desarrollar un robot que pueda desempeñar la tarea de limpieza en un entorno de oficina. Como se señaló anteriormente, en este trabajo se presentan seis algoritmos que permiten el cálculo de rutas de cobertura a partir de un mapa. Estos algoritmos son descritos a continuación:

- **Descomposición Boustrophedon** [43]
  Este algoritmo corresponde a la familia de algoritmos de descomposición exacta. El algoritmo consta de tres etapas para poder realizar el cálculo de la ruta. En primer lugar, el espacio de interés (mapa) es dividido en celdas, las cuales pueden tener forma y tamaño arbitrario y que cumplen con particionar de manera exacta el espacio libre de tal manera que al juntar las celdas el espacio se puede reconstruir perfectamente. El cálculo de las celdas se realiza definiendo una dirección de barrido y una dirección de corte como se muestra en la Figura 3.1. Para este caso, la dirección de barrido recorre horizontalmente la imagen y la línea de corte recorre la imagen verticalmente. Al ir atravesando el espacio a través de la línea de barrido es posible encontrarse con obstáculos. Cuando la línea de barrido intercepta un obstáculo de tal forma que en este punto de intersección tiene una normal perpendicular a la dirección de corte, se dice se ha encontrado un punto crítico. Este punto genera la apertura y/o cierre de una celda, como se muestra en la Figura 3.1, según si el espacio se está dividiendo o juntando.

  Tras el cálculo de las celdas, se debe determinar el orden óptimo de visita de estas, es decir el camino más corto. Para realizar este cálculo el problema es convertido al problema del vendedor viajero. En este caso cada celda representa un pueblo que se debe visitar. Este cálculo se puede realizar mediante algoritmos heurísticos [75] como *Nearest Insertion Heuristic* [41], algoritmos genéticos [76, 77, 78] e incluso mediante el uso de fuerza bruta, esto es, computando todas las posibles soluciones y eligiendo la mejor.

  Finalmente, cada celda a visitar es cubierta con movimientos simples en zig-zag como los que se observan en las Figuras 5.1b y 5.1h.
- **Problema del vendedor viajero usando grillas**
  Este algoritmo pertenece a la familia de algoritmos de descomposición aproximada. Su enfoque se basa en convertir el problema de la planificación de rutas de cobertura en el clásico problema del vendedor viajero. Para esto, este planificador divide el espacio en una cuadrícula de celdas de tamaño uniforme como la que se observa en la Figura 3.2. El tamaño de las celdas esta determinado por el tamaño del *footprint* de la base móvil a utilizar. Este corres-

[6] `https://github.com/Fields2Cover/Fields2Cover`
[7] `https://github.com/ipa320/ipa_coverage_planning/tree/noetic_dev`

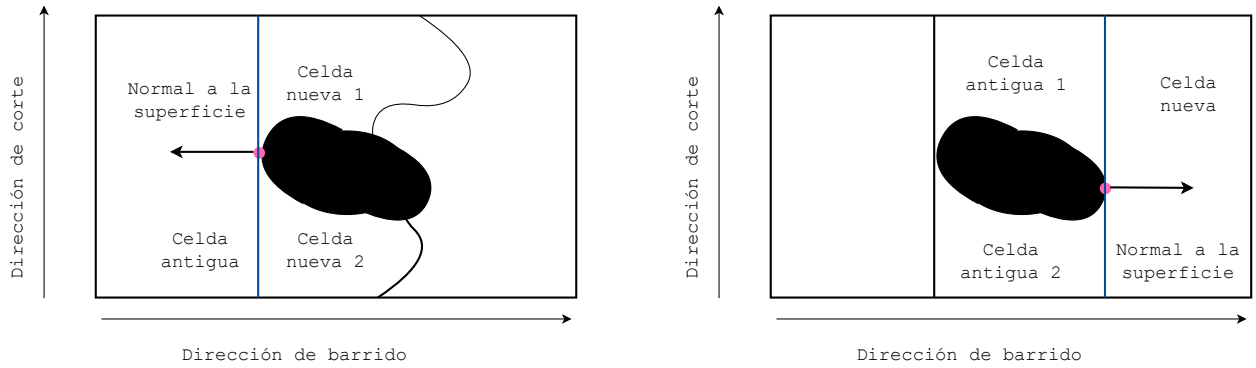


Figura 3.1: Ejemplo descomposición celular mediante Boustrophedon. (a) Evento de abertura de celdas. (b) Evento de cierre de celdas. En ambas figuras se señala el punto crítico que da origen a los eventos, como se aprecia la normal a la superficie en ese punto es perpendicular a la dirección de corte.

ponde al cuadrado máximo que encierra al robot, en los ejes x e y, como se ejemplifica en la Figura 4.2.

Posteriormente se debe determinar la ruta mas corta que permita visitar todas las celdas libres de obstáculos lo cual es equivalente a resolver el problema del viajero. Este problema es bien conocido y hay distintos enfoques para resolverlo como el uso de algoritmos genéticos, algoritmos heurísticos, entre otros [79].

Finalmente, teniendo el orden de visita de las celdas, se busca la ruta mas corta entre pares de celdas consecutivas mediante el uso del algoritmo A* sobre el mapa original del espacio.

- **Redes neuronales bio-inspiradas** [80]
  El enfoque tomado por este algoritmo es del tipo descomposición aproximada. El espacio es descompuesto en una cuadrícula de resolución adecuada al tamaño del robot a utilizar. Para determinar la ruta que permite cubrir todas las celdas se utiliza un enfoque bio-inspirado en la dinámica de activación de las neuronas. Luego, cada celda de la cuadrícula corresponde a una neurona con ocho vecinos adyacentes, como se muestra en la Figura 3.2. En cada iteración se calcula el potencial de cada una de las ocho celdas vecinas según la Ecuación (3.1), donde p_n corresponde a la siguiente neurona a visitar, x_j el vecino j-ésimo, c es una constante y $\Delta\theta_j$ representa el diferencia angular entre el vecino j-ésimo y la posición actual. La neurona elegida para visitar a continuación será aquella con mayor valor de potencial. El algoritmo sigue iterando hasta cubrir todas las celdas que componen el espacio libre. Para evitar atascos o zonas inalcanzables en lugares como esquinas, se establece que las celdas pueden ser visitadas más de una vez.

$$p_n = max\left\{x_j + c\left(1 - \frac{\Delta\theta_j}{\pi}\right)\right\} \tag{3.1}$$

- **Minimización local de energía basada en grilla** [81]
  Este algoritmo pertenece a la familia de algoritmos de descomposición aproximada. El enfoque se basa en dividir el espacio en una cuadrícula de tamaño apropiado para las dimensiones del *footprint* robot. Este algoritmo asume que la posición inicial de la base móvil es una de las esquinas del espacio. En cada iteración el robot se mueve una celda de distancia como

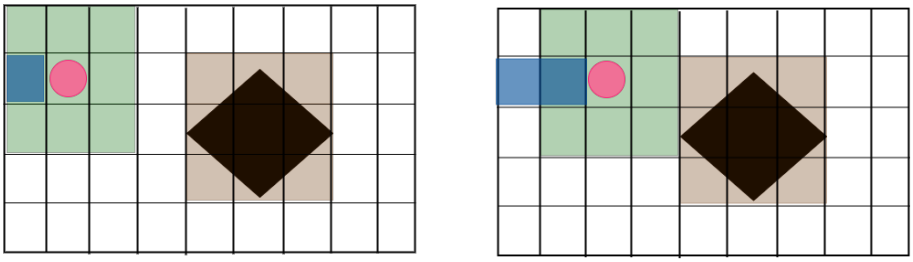


Figura 3.2: Representación del funcionamiento de los algoritmos de Redes neuronales bio-inspiradas y Minimización local de energía basada en grilla en dos instantes de tiempo consecutivos, t y $t+1$. A la izquierda se observa la posición en t del robot, a la derecha la posición en $t+1$. El circulo rojo representa el robot, los cuadros verdes el vecindario actual. Los cuadros marcados en café representan celdas marcadas como ocupadas con obstáculo, las celdas azules son celdas por donde ya paso el robot y las celdas blancas representan espacio por cubrir.

máximo de tal manera que la siguiente posición queda limitada a una de los ocho celdas adyacentes en la actual posición. Para elegir la siguiente celda, n, a visitar, se calcula para los ocho vecinos de la celda actual, p, su energía según la Ecuación (3.2), la cual incorpora en el costo la distancia traslacional (Ecuación (3.3)), la distancia rotacional (Ecuación (3.4)) y un término atractivo $N(n)$ (Ecuación (3.5)) para que el robot prefiera aquellas celdas que se encuentran cercanas a aquellas que ya han sido cubiertas, donde L es el conjunto de todas las celdas ya procesadas y Nb_8(n) el conjunto de los ocho vecinos de la celda actual.

La siguiente celda a visitar será aquella que tenga menor valor de energía. El algoritmo sigue iterando de la forma descrita hasta que todas las celdas han quedado cubiertas. Es importante mencionar que se permite revisitar celdas para evitar que el robot quede atascado en una zona durante el cálculo de la ruta.

A modo general, el funcionamiento de este algoritmo es similar al de Redes neuronales bio-inspiradas, habiendo cambios en la función objetivo con la cual se decide que celda se visitara a continuación [81].

$$E(p,n) = d_t(p,n) + d_r(p,n) + N(n) \tag{3.2}$$

$$d_t(p,n) = \frac{\sqrt{(p_x - n_x)^2 + (p_y - n_y)^2}}{l} \tag{3.3}$$

$$d_r = \frac{|p_\theta - n_\theta|}{\pi/2} \tag{3.4}$$

$$N(n) = 4 - \sum_{k \in Nb_8(n)} \frac{|k \cap L|}{2} \tag{3.5}$$

- **Planificación de trayectoria de cobertura basada en líneas de contorno**
  A diferencia de los otros algoritmos de `ipa_room_exploration`, este método no utiliza ningún tipo de descomposición celular. Su funcionamiento se basa en el calculo del esqueleto del grafo de Voronoi del mapa a cubrir. En primer lugar este algoritmo realiza una descomposición del mapa en celdas de Voronoi como se describe en [82]. Luego, para cada celda se calcula la máxima distancia al obstáculo más cercano a la celda o al borde del mapa. Utilizando esta distancia, se calculan distintos niveles al interior de la celda en función del radio

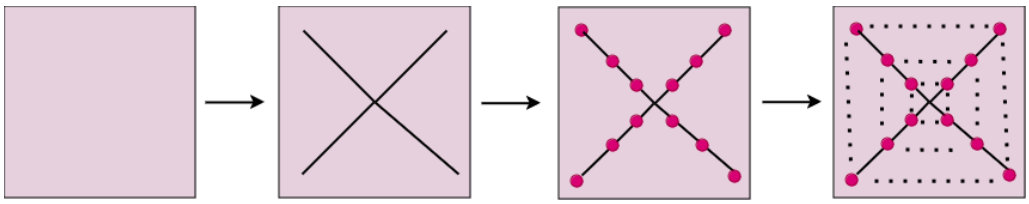


Figura 3.3: Ejemplo de cálculo de una ruta de cobertura con algoritmo basado en líneas de contorno. Para un mapa dado se cálcula el esqueleto del grafo de Voronoi, en este caso se obtiene como resultado una figura similar a una cruz. Luego, sobre estas lineas que se generan a partir del esqueleto, se calculan los distintos niveles por lo cuales el robot debe posicionarse para poder cubrir el área. Estas posiciones conforman el plan de navegación.

de cobertura del robot, r_c, como se muestra la Ecuación (3.6) donde i representa el nivel i-ésimo tal que $c_i < d_{max}$. Esto da lugar a que el plan calculado siga las líneas del contorno del mapa. Un ejemplo sencillo que permite entender el funcionamiento del algoritmo es el que se presenta en la Figura 3.3.

$$c_i = r_c + 2 \cdot i \cdot r_c \tag{3.6}$$

- **Planificación de trayectoria de cobertura con colocación de sensores convexos** [83]

  El enfoque tomado por este algoritmo se basa en ver el problema de las rutas de cobertura como el problema de la galería de arte, el cual consiste en resolver el problema de colocar la menor cantidad de observadores (cámaras, guardias, sensores), tal que un área pueda ser vista en su totalidad al mismo tiempo. En este algoritmo se asume que el rango de alcance o FoV (Field of View) de los sensores se limita al tamaño del robot.

  Lo anterior permite encontrar el número de poses mínimas que permiten cubrir el área. Posteriormente, se debe calcular la ruta mas corta que debe recorrer el robot para pasar por todas las poses lo cual puede ser visto como el problema del vendedor viajero. De esta manera, mediante algún algoritmo que permita resolver el problema del vendedor viajero se obtiene la ruta de cobertura. Lo anterior se puede observar en la Figura 3.4 donde se muestra a la izquierda todas las posiciones donde se debe colocar sensores con el FoV dado y a la derecha se muestra la ruta mas corta por la cual se viaja por todas esas poses.

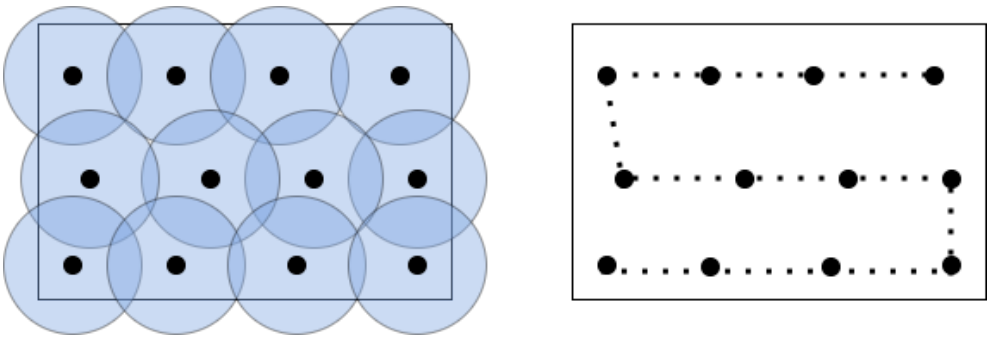


Figura 3.4: Ejemplo de calculo de ruta de cobertura con algoritmo con colocación de sensores convexos. El círculo en azul representa el robot. Los círculos negros representan el punto central de la circunferencia y las poses que se calculan. La figura a la izquierda representa la solución al problema de la galería de arte para el mapa rectangular dado. La imagen a la derecha representa la solución al problema del vendedor viajero usando las poses que se obtienen a partir de haber resuelto el problema de la galería de arte sobre el mapa.

Capítulo 4

Diseño e implementación

Para que una base móvil pueda llevar a cabo la tarea de *coverage navigation* es necesario el desarrollo y configuración de una serie de módulos que dividen la tarea y en conjunto permiten su realización. Estos módulos corresponden a la información sensorial del robot, el sistema de localización, los planificadores de ruta local y global y el *behavior tree*. En este capítulo se aborda tanto el diseño como implementación de estos componentes. A nivel macro, el sistema de dividió en dos elementos: un *stack* de navegación y un árbol de comportamiento. El *stack* tiene como propósito permitir la navegación autónoma del robot y abarca la información sensorial, un sistema de localización y planificación global y local que permiten el cálculo de rutas y evasión de obstáculos. Por otro lado, el *behavior tree* se encarga de la toma de decisiones de más alto nivel, como estrategias de replanificación y de recuperación en ruta, así como de coordinar la ejecución de distintas tareas.

4.1. *Stack* de navegación

4.1.1. Diseño de componentes

El diseño de solución propuesto para el *stack* de navegación se basa en el enfoque clásico con el cual se aborda el problema de navegación autónoma en robótica. Este enfoque consiste en dividir el problema en varios subproblemas, tales como mapeo, localización, planificación de rutas y evasión de obstáculos. Para el caso particular de *coverage navigation*, la planificación de rutas toma un significado distinto al de la navegación clásica. En este último caso se busca permitir al robot desplazarse de manera autónoma hasta una pose objetivo pero para el caso de *coverage navigation* se busca que el robot pueda cubrir de manera autónoma un área de interés, por lo cual no existe un único objetivo de navegación.

El enfoque de diseño propuesto adquiere sentido al realizar una abstracción sobre el significado que se le da a la acción de cubrir una zona. Si el plan de cobertura es considerado como una sucesión ordenada de poses a las cuales navegar, el problema de *coverage navigation* puede ser entendido como la tarea de navegar hasta estas poses de manera secuencial. Esta perspectiva permite aplicar de manera efectiva los enfoques de navegación clásicos al problema de la navegación de cobertura. Con esto, se propone el *stack* de navegación que se observa en la Figura 4.1, el cual muestra los distintos componentes que integran al sistema y el modo en que se relacionan entre ellos.

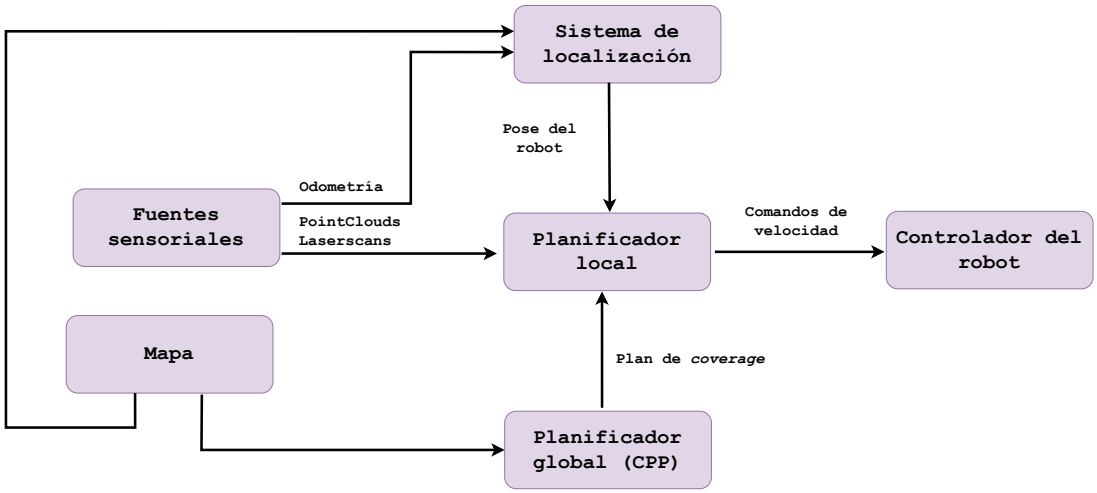


Figura 4.1: *Stack* de navegación para *coverage navigation* a nivel conceptual.

El *stack* propuesto consta de seis componentes. Para que el sistema pueda lograr que el robot se mueva de tal manera que cubra el área de interés, en primer lugar se requiere de una representación métrica del espacio sobre el cual trabajará el robot. Este mapa es de vital importancia, ya que otros componentes del sistema lo requieren para poder ejecutar sus tareas. Sobre este mapa, el planificador global, que corresponde a un algoritmo de *coverage path planning* (CPP), calcula una ruta que permite realizar la cobertura, y este plan es posteriormente entregado al planificador local el cual lo traduce a comandos de velocidad que se entregan al controlador de la base móvil, que finalmente se encarga de ejecutarlos y por ende de mover al robot. Adicionalmente, el planificador local se encarga de la evasión de obstáculos que puedan aparecer en el camino, y de imprevistos que puedan surgir, es decir, se encarga de que la base móvil navegue de manera segura sin colisionar. Para poder realizar estas funciones, el planificador local generalmente requiere saber en todo momento en que lugar se encuentra el robot, para lo cual se utiliza un sistema de localización, y requiere de información de su entorno en tiempo real, para lo cual se pueden utilizar distintos sensores y sistemas de percepción como sensores LiDAR que pueden entregar información en forma de nube de puntos (como lo que se observa en la Figura 5.4) y en forma de scan láser (como se observa en la Figura 5.5).

Es importante tener en consideración que al realizar evasión de obstáculos o maniobras de recuperación[1], el planificador local puede desviarse de la ruta de cobertura calculada por el planificador global. Lo anterior puede tener como consecuencia que queden áreas sin cubrir, por lo cual se debe incorporar un mecanismo para resolver este tipo de situaciones. Este problema será abordado por el *behavior tree* y se describe en detalle en la Sección 4.2.

4.1.2. Implementación

Se trabajaron dos implementaciones con distinto enfoque en torno a la planificación local para llevar a cabo el *stack* de navegación propuesto en la sección de diseño. La primera alternativa se basa en el uso de `move_base`, un paquete de ROS que proporciona una interfaz para configurar y ejecutar el *stack* de navegación de ROS. La segunda implementación se basa en el uso de un

[1]Conjunto de acciones o movimientos realizados con el objetivo de corregir la trayectoria o posición cuando se encuentra en una situación inesperada o no deseada.

planificador local basado en aprendizaje reforzado, desarrollado por AMTC, el cual corresponde a una variación del planificador desarrollado en [84]. Es importante mencionar que, a diferencia del software proporcionado por ROS y `move_base`, el planificador local de AMTC, no es de código abierto.

Para llevar a cabo el cálculo de rutas de cobertura, se utilizó `ipa_room_exploration`. Como se detalla en la Sección 3.2.1, este paquete integra seis algoritmos de *coverage path planning*, los cuales tienen diversos principios de funcionamiento. El formato de los planes generados por cualquiera de estos algoritmos consiste en una serie de poses de navegación secuenciales. Lo anterior es concordante con lo propuesto en la sección de diseño (ver Sección 4.1.1), donde se abstrae el plan de cobertura a una serie de *waypoints* (poses objetivo) a los cuales navegar.

La implementación de este paquete incluye un *action server* de ROS[2] que permite enviar solicitudes para calcular planes de cobertura. Estas solicitudes deben contener información respecto al área a cubrir y al robot. En relación al área que se quiere cubrir, se debe proporcionar un mapa binario en formato de imagen (PNG, JPG, PGM, etc.), junto a la especificación de su resolución y su origen. Dado esto, para esta aplicación se asume la existencia de un mapa del área a cubrir. Este mapa puede ser obtenido previamente mediante el uso de algún algoritmo de SLAM, por ejemplo *gmapping* [85]. En relación al robot, se deben especificar dos parámetros clave: su radio de cobertura, que corresponde al radio del máximo círculo que puede existir dentro del cuerpo del robot, y su radio de tamaño, que corresponde al radio del mínimo círculo que contiene a todas las partes del cuerpo del robot. Estos conceptos, además del *footprint* del robot, se ilustran en la Figura 4.2.

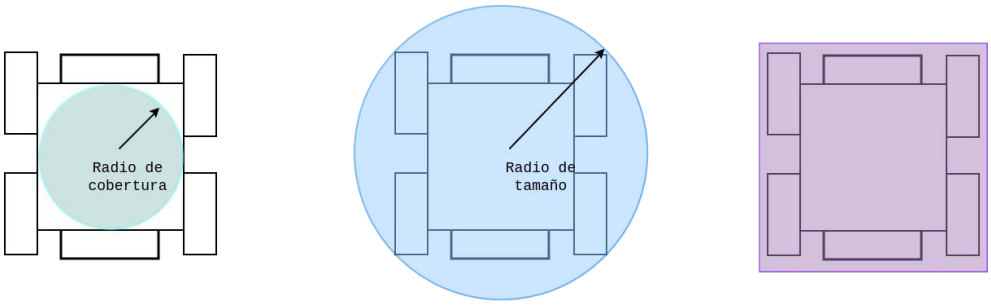


Figura 4.2: Parámetros del robot: radio de cobertura (izquierda), radio de tamaño (centro) y *footprint* del robot (derecha).

Luego, la implementación del sistema se basa en la construcción de un *stack* de navegación clásico que permite a la base móvil navegar de manera autónoma hasta una pose objetivo en el espacio y de un *behavior tree* que permite el cálculo y ejecución de las rutas de cobertura. Para la generación de rutas de cobertura se utilizará la implementación de `ipa_room_exploration` y el *stack* de navegación permitirá su seguimiento usando objetivos de navegación a poses secuenciales muestreadas del plan de cobertura. Adicionalmente, el *behavior tree* debe integrar alguna estrategia de recuperación que permita realizar replanificaciones en ruta.

[2]Información en detalle de qué es y como funciona un *action server* de ROS puede encontrase en `http://wiki.ros.org/actionlib`.

Implementación basada en uso de `move_base`

`move_base` corresponde a un paquete de ROS que provee una interfaz para la interacción, configuración y ejecución de los distintos componentes que conforman un *stack* de navegación. Los distintos componentes de este *stack* y sus interacciones pueden observarse en la Figura 4.3. Mediante este paquete es posible habilitar la navegación autónoma de un robot, de manera relativamente sencilla, instalando y configurando los sub-paquetes necesarios. Es importante destacar que la implementación dada por ROS no es robot dependiente, en el sentido de que es ajustable mediante parámetros, por lo cual puede ser usada con una gran variedad de bases móviles tras se configurada correctamente.

Al igual que para el *stack* propuesto, `move_base` requiere un sistema de localización, un planificador local y uno global, información sensorial y un mapa del espacio. Pero también requiere componentes adicionales, como mapas de costo y acciones de recuperación. El paquete `move_base` permite elegir qué se ejecutará en cada componente o nodo, esto es, se pueden elegir distintas implementaciones. Por ejemplo, para el planificador global se puede utilizar A*, Dijkstra; para los mapas de costo se pueden elegir distintas configuraciones, etc. Más aún, es posible para cada componente utilizar una implementación propia siguiendo los requerimientos necesarios para que dicha implementación funcione como un "*plugin*".

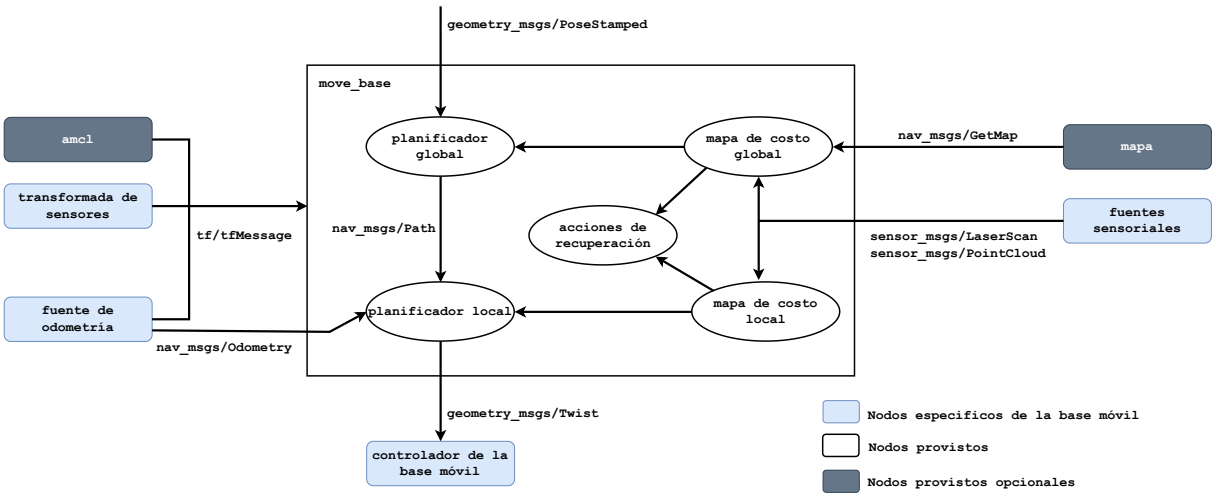


Figura 4.3: *Stack* de navegación clásico implementado por *move_base*. Adaptado de `move_base`.[3]

Particularmente, para esta implementación se utilizó el paquete `map_server` para publicar el mapa del área donde se trabajará; como sistema de localización se utilizó la implementación que posee ROS de AMCL; para el planificador local se trabajó con la implementación de DWA (*Dynamic Window Approach*) que provee ROS; para el caso del planificador global se utilizó `NavFn`; y para los mapas de costo se configuró una capa de inflación y una capa de obstáculos. Para cada módulo se ajustaron los distintos parámetros existentes para que el *stack* se desempeñe correctamente con la base móvil objetivo.

[3] `http://wiki.ros.org/move_base`

Implementación basada en el uso de planificador local entrenado con aprendizaje reforzado

Un enfoque con el cual también se ha abordado el problema de navegación autónoma en robots es la utilización de aprendizaje reforzado [86, 87, 84]. La implementación del *stack* de navegación propuesta en esta sección se basa en la utilización de un planificador local basado en aprendizaje reforzado que fue desarrollado por AMTC en [84], y posteriormente adaptado para funcionar con robots *skid-steered*, diferentes al Pioneer 3DX usado en el trabajo inicial. Este planificador aborda el problema de alcanzar un objetivo de navegación mientras evade posibles obstáculos, valiéndose únicamente de información sensorial inmediata. Particularmente, la información sensorial proveniente de sensores debe estar en formato de nube de puntos en dos dimensiones, por lo cual debe haber un nodo de pre procesamiento de esta información para entregar los datos al planificador en el formato requerido (por ejemplo, si la información proviene de múltiples LiDARs la información de rango debe ser fusionada y procesada). El *stack* que define este enfoque se muestra en la Figura 4.4.

Para integrar el planificador local en cuestión, se desarrolló un *action server* de ROS para poder abordar la comunicación con el resto de los componentes del sistema y entregarle objetivos de navegación. Para el sistema de localización se utilizó AMCL y para la publicación del mapa se utilizó el paquete `map_server`. Es importante notar que al igual que en el caso de `move_base`, el *stack* sólo puede ejecutar un objetivo de navegación a la vez, por lo cual para ejecutar la ruta de cobertura se debe ir indicando de manera secuencial los *waypoints* al *stack*.

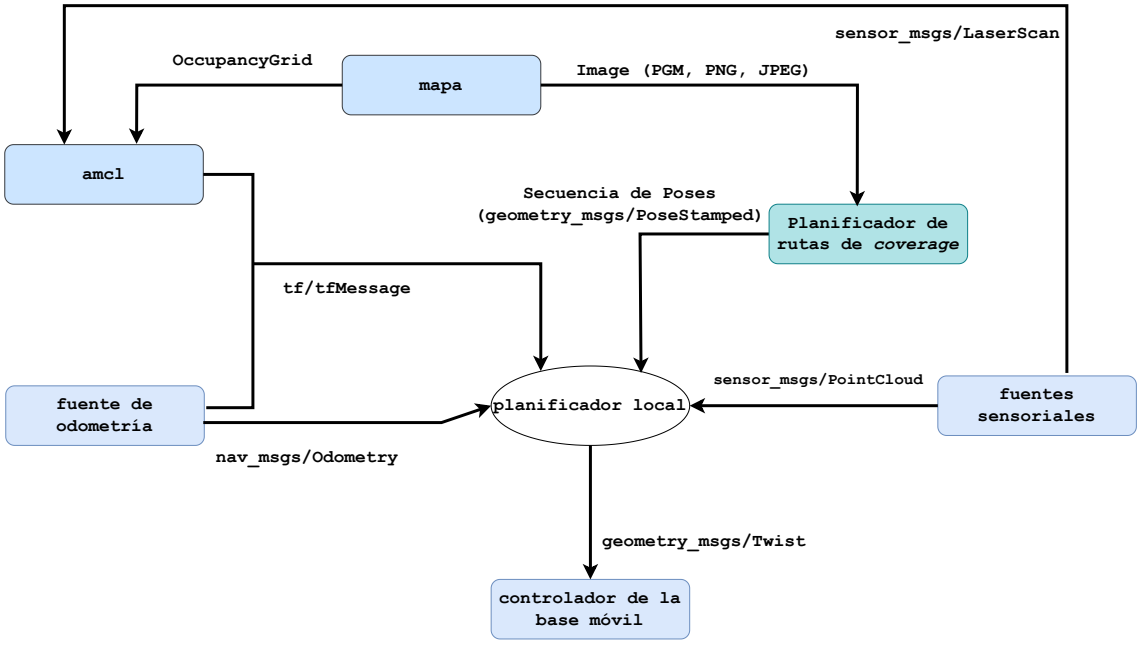


Figura 4.4: *Stack* de navegación implementado basado en el planificador local entrenado con aprendizaje reforzado de AMTC.

Para este enfoque, se tiene que la implementación es parcialmente robot dependiente ya que el planificador debe ser entrenado para las dimensiones de la base móvil objetivo. Sin embargo, el planificador puede ser utilizado con una variedad de bases móviles. Para el resto de los componentes del *stack*, se tiene que estos no presentan dependencias respecto al robot utilizado y son fácilmente configurables para el caso de uso mediante el ajuste de parámetros. Una observación

adicional importante que debe ser realizada, es que este planificador basado en aprendizaje reforzado, a diferencia de DWA, solo comanda velocidades lineales positivas, es decir, el robot no puede moverse en reversa. Adicionalmente, llega a posiciones y no a poses objetivo, es decir, no considera una orientación especifica objetivo al momento de dirigirse a un *waypoint*.

4.2. *Behavior tree*

4.2.1. Diseño de nodos

Durante la ejecución de la ruta de cobertura, el robot puede enfrentarse a diversos imprevistos, especialmente en un entorno real, donde factores como obstáculos dinámicos, cambios en las condiciones del terreno y otros contratiempos pueden forzar al robot a desviarse del plan establecido previamente. Estas situaciones imprevistas pueden resultar en que queden áreas sin cubrir, por lo cual es necesario agregar una componente reactiva al sistema que permita al robot adaptarse dinámicamente a las diversas circunstancias que pueda enfrentar. En este contexto, realizar replanificaciones a lo largo de la ejecución se vuelve imperativo, ya que proporciona al robot la capacidad de ajustar su trayectoria, asegurando así una cobertura completa y eficiente incluso en presencia de contratiempos.

La necesidad de realizar replanificaciones durante la ejecución del plan conlleva un desafío significativo, pues el cálculo de rutas de cobertura es computacionalmente costoso. Se ha demostrado que este problema, es decir, encontrar una ruta óptima que permita cubrir un área, es de tipo NP-Hard [88], lo cual implica que a medida que el tamaño del problema aumenta, este se vuelve significativamente mas costoso de resolver. Lo anterior implica que el cálculo de rutas puede tomar tiempos de ejecución significativos lo cual limita la reactividad que el sistema pueda tener.

Con lo anterior en consideración, se propone una arquitectura para el *behavior tree* que tiene como objetivo disminuir los tiempos de cálculo de las rutas de cobertura. Esta reducción puede lograrse al abordar el problema desde la perspectiva de la reducción del tamaño del área de cobertura, lo que equivale a disminuir el área sobre la cual se calcula la ruta de *coverage*. Luego, el árbol de comportamiento permite lograr esto procesando el mapa del área a cubrir por "parches". Como resultado de esta estrategia, el cálculo de las rutas de cobertura se vuelve considerablemente más rápido (a medida que se divide en más secciones el mapa). Esta mejora en la eficiencia permite al sistema ser reactivo ante imprevistos, ya que tiene la capacidad de realizar replanificaciones de forma rápida.

Adicionalmente, se toma en consideración el hecho de que no necesariamente se desea cubrir toda el área libre en un mapa, sino que puede que solo un área sea de interés. Por ejemplo, en un complejo de oficinas o una casa solo algunas zonas podrían requerir ser limpiadas; en un ambiente de exterior solo ciertas zonas de un patio o calle podrían ser de interés cubrir, etc. Luego, se define que el mapa sobre el cual el árbol realizará la cobertura será solo la de interés. Por esto, se debe realizar sobre el mapa del espacio un cercado para indicar el área de trabajo, como se muestra en la Figura 5.11 (izquierda).

La estructura del árbol de comportamiento propuesto es la que se muestra en la Figura 4.5. Al realizar *ticks* al árbol, el flujo de acción comienza intentando cargar el mapa del área que se quiere cubrir y dividiéndolo en N secciones iguales, donde N es un parámetro que define el usuario. Com-

pletada esta actividad, el *tick* se propaga a la derecha, donde la acción a ejecutar es cubrir el área. En esta etapa se tiene un nodo decorador que tiene como política devolver éxito solo cuando ya todas las secciones del mapa han sido marcadas como cubiertas. Para cada sección del mapa, se calcula una ruta que permita cubrirla y luego esta se ejecuta. Al concluir la ejecución del plan se vuelve a intentar calcular una ruta para la misma sección. Si ya no queda nada por cubrir, entonces no habrá un plan que ejecutar y se marcará dicha sección como completada dando paso al procesamiento de la siguiente sección. En caso contrario, si se encuentra una ruta, esta es ejecutada y se sigue esta lógica hasta que todas las secciones del mapa hayan sido completadas. El comportamiento lógico del *behavior tree* es el que se detalla en el Pseudocódigo 1 y es técnicamente equivalente a lo que se muestra en la Figura 4.5.

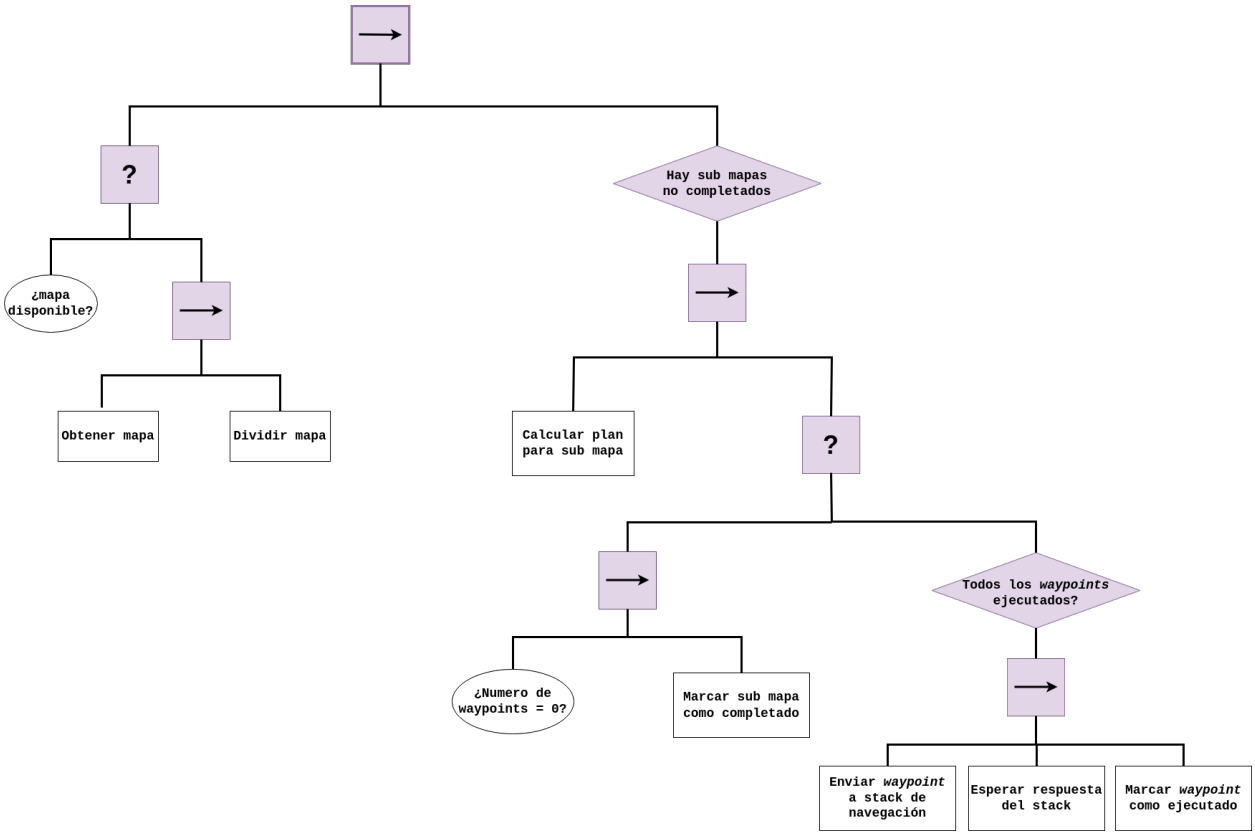


Figura 4.5: Árbol de comportamiento de la aplicación.

4.2.2. Implementación

Para llevar a cabo el diseño propuesto para el *behavior tree* es necesario que el robot pueda recordar que partes del mapa ya ha cubierto. Para lograr esto se desarrolla un nodo externo al árbol que permite al robot conocer esta información. Este nodo se implementa en ROS y su funcionamiento se basa en marcar en el mapa como ocupado los lugares que el robot ya cubrió proyectando el *footprint* del robot sobre el mapa para un instante de tiempo dado. Esta información es publicada en un tópico al cual el árbol de comportamiento se suscribe.

Además de proporcionar información sobre el área cubierta, el nodo auxiliar también publica en un tópico el progreso de la tarea, expresado como un valor en el rango de 0 a 1, donde 1 representa el 100 por ciento de la tarea completada. Esta información de progreso es valiosa para la supervisión

Pseudocódigo 1: Representación del *Behavior tree* como algoritmo, conceptual.

```
Obtener mapa del entorno M.
Dividir mapa en m secciones.
for sección 1...m do
    while sección no cubierta do
        Calcular plan de cobertura C sobre sección.
        if C no nulo then
            for waypoint w en C do
                Enviar w a stack de navegación.
                Esperar respuesta.
        else
            Marcar sección como cubierta.
```

y evaluación del rendimiento del robot, permitiendo una visualización clara del estado de la tarea en tiempo real.

Para la implementación del *behavior tree* se utilizó el lenguaje de programación Python. Particularmente se utilizaron los paquetes `rospy` y `pytrees`. La librería `rospy` corresponde a un cliente de ROS: permite la comunicación con los distintos componentes de la solución, como el *stack* de navegación y el planificador de rutas de cobertura de `ipa_room_exploration`, mediante la creación suscriptores, publicadores, clientes, entre otros. Utilizando este paquete se integró el árbol de comportamiento al sistema del robot. Por otro lado, para implementar el árbol, dígase, su estructura, se utilizó `pytrees`, que ofrece una base para la construcción de arboles de comportamiento, proporcionando los nodos de control de flujo y un marco de trabajo para desarrollar los distintos comportamientos requeridos.

Para la implementación del nodo que obtiene el mapa se utilizó un suscriptor de ROS al tópico que publica estos datos, e información relevante respecto a este, como su resolución y coordenadas de origen. Para el nodo de cálculo de rutas, de cobertura se implementó un *action client* de ROS que se comunica con el servidor del paquete `ipa_room_exploration`. Para la ejecución de la ruta, se implementaron dos *action client*, uno para la comunicación con `move_base`, y otro para la comunicación con el planificador entrenado con aprendizaje reforzado.

Capítulo 5

Evaluación experimental

La evaluación del sistema implementado se divide en varias etapas que buscan evaluar el desempeño de la aplicación desarrollada bajo distintas condiciones. En una primera etapa se busca poder caracterizar los algoritmos provistos en el paquete `ipa_room_exploration` y establecer una comparación en cuanto a rendimiento entre ellos mediante el uso de métricas de interés. En una segunda etapa se busca evaluar el desempeño del sistema completo (*behavior tree* y *stacks* de navegación), para lo cual se realizan pruebas en ambientes de distinta complejidad en simulación, mediante el uso de GAZEBO. Finalmente, se busca evaluar el desempeño del sistema en el mundo real, para lo cual se realizaron pruebas desplegando la aplicación en una base móvil de tipo *skid-steered* en ambientes de exterior.

5.1. Benchmark algoritmos de *coverage path planning*

El propósito de esta evaluación es comparar los diferentes algoritmos de *coverage path planning* que se proveen en el paquete `ipa_room_exploration` mediante el cálculo de diversas métricas de interés. La evaluación se realiza sobre un subconjunto de mapas de la base de datos *House expo* [89], la cual incluye 35.000 mapas correspondientes a entornos *indoor*. Para la selección de este subconjunto, se aplicó un filtro basado en el tamaño, excluyendo todos los mapas con una resolución mayor a 700 x 700 píxeles, resultando en aproximadamente 1.000 mapas. Esta elección se fundamenta en la observación de que el cálculo de rutas de *coverage* se vuelve considerablemente más costoso para mapas de dimensiones mayores. Se espera que esta evaluación permita caracterizar a los algoritmos en términos de rendimiento y permita conocer que diferencias hay entre los planes que entregan.

5.1.1. Métricas

Las métricas de evaluación aplicadas son las que se describen a continuación:

- **Tiempo de cálculo** (T_c) **:** Corresponde al tiempo que le lleva al algoritmo realizar el cálculo del plan. Los tiempos de cálculo son una medida importante para el sistema, pues mientras menor sea este valor, más rápido se podrá realizar replanificaciones. Luego, es deseable obtener tiempos de cálculo bajos. Esta métrica se calcula como se muestra en la Ecuación 5.1.

$$T_{\mathrm{c}} = T_{\mathrm{f}} - T_0 \tag{5.1}$$

Donde, T_0 corresponde al instante de tiempo en el cual se solicita el cálculo de una ruta de cobertura para un cierto mapa y T_{f} corresponde al instante de tiempo cuando la respuesta por parte del planificador llega. Luego, el tiempo de cálculo del plan de cobertura corresponde a la diferencia entre estos dos instantes.

- **Porcentaje de *coverage* predicho** ($C_{\%}$) **:** Corresponde a cuanto porcentaje del área logra cubrir el plan entregado. Este cálculo se realiza asumiendo que el robot sigue el plan íntegramente, sin salirse de él, por lo cual corresponde al estimado de lo que se obtendría en el caso ideal [1]. El porcentaje de cobertura nos indica que tan efectivamente el algoritmo cubre un área. El cálculo de esta métrica se realiza computando la razón entre los píxeles libres inicialmente, P_{inicial}, y los que se marcaron como cubiertos, P_{final}. Se marcan como cubiertos a aquellos píxeles que son alcanzados por el *footprint* robot para alguna pose del plan. Matemáticamente, se tiene lo que se muestra en la Ecuación 5.2.

$$C_{\%} = \frac{P_{\text{final}}}{P_{\text{inicial}}} \cdot 100 \tag{5.2}$$

- **Largo del plan (P_{len}:)** Corresponde a cuanta distancia en total el robot deberá recorrer si ejecuta el plan íntegramente. Un menor largo del plan puede significar que la ruta es "más óptima"[2] debido, a que hay menos lugares cubiertos más de una vez, pero también podría significar, por ejemplo, que la ruta es muy corta y no cubre todo el espacio. Además, el largo de la ruta influye en la ejecución del sistema, (*behavior tree*), ya que mientras más distancia se deba recorrer más se demorará el robot en completar la tarea. Para calcular el largo del plan se calculó la sumatoria entre la distancia A* entre *waypoints* consecutivos, $W_{\text{i+1}}, W_{\text{i}}$. La distancia A*, $d_{A^\star}$, corresponde a la ruta más corta posible entre dos puntos, calculada con el algoritmo de búsqueda A*. Sea N la cantidad de *waypoints* totales del plan de cobertura, para el cálculo de esta métrica se tiene lo que se muestra en la Ecuación 5.3:

$$P_{\text{len}} = \sum d_{A^\star}(W_{\text{i+1}}, W_{\text{i}}) \tag{5.3}$$

- **Numero de vueltas (θ_{T}):** Corresponde a cuantos giros debería de realizar el robot para poder completar el plan de cobertura. Para dos poses consecutivas, se calcula la diferencia entre los ángulos de estas poses, $\theta_{\text{i}}, \theta_{\text{i+1}}$, luego cada uno de estos valores es sumado para calcular el total de ángulos que debe recorrer el robot para ejecutar el plan, como se muestra en la Ecuación 5.4.

  Esta información es de interés debido a que la base objetivo de la aplicación es de tipo *skid-steered*, la cual se caracteriza por presentar deslizamiento al girar, lo cual puede inducir, por ejemplo, errores de localización que pueden afectar negativamente el *performance* del sistema. Adicionalmente, un número elevado para esta métrica podría significar que el plan entregado es ruidoso, esto es, que hay desviaciones en la orientación entre poses sucesivas que son pequeñas. Esto es poco deseable, ya que obliga al robot a ejecutar maniobras de giro innecesarias. Esto es particularmente perjudicial para bases con capacidades de movimiento limitadas.

$$\theta_{\text{T}} = \sum |\theta_{\text{i}}| - |\theta_{\text{i+1}}| \tag{5.4}$$

[1] Durante la ejecución del plan, en simulación o realidad, puede suceder que el plan obtenido no sea ejecutado a la perfección. Por este motivo, asumir que el plan se sigue estrictamente corresponde a una predicción.

[2] Se habla de optimalidad en el sentido de encontrar la ruta más corta que permita cubrir toda el área de interés.

- **Densidad de *waypoints* (ρ_{path}):** Se busca medir que tan saturado en *waypoints* es el plan de cobertura entregado. Para esto, esta métrica calcula cual es el factor promedio de distancia entre poses, este factor indica cada cuantos diámetros del robot hay una pose. Esto se obtiene calculando el promedio de la distancia entre poses consecutivas, $W_{\text{i+1}}, W_{\text{i}}$, y luego se normaliza este valor usando el diámetro del robot, $2R_{\text{c}}$,como se muestra en la Ecuación 5.5.

  La densidad de poses del plan de cobertura es una característica interesante a conocer, pues como se verá más adelante en esta sección, tiene influencia en el desempeño del sistema. Una mayor densidad de *waypoints* ayuda a que la ejecución del plan se apegue más a este debido a que hay una menor distancia entre poses consecutivas, y por lo tanto el planificador local tiene menos margen para apartarse de la ruta original. Por otra parte, una menor cantidad de poses permite que la evasión de obstáculos sea más efectiva, debido a que el planificador local tiene mayor libertad para apartarse del plan. No obstante, lo anterior puede dar lugar a zonas sin cubrir.

$$\rho_{\text{path}} = \frac{\sum d(W_{\text{i+1}}, W_{\text{i}})}{N_{\text{waypoints}}} \frac{1}{2R_{\text{c}}} \tag{5.5}$$

5.1.2. Resultados

A continuación se presentan los resultados obtenidos en el *benchmark*. La Tabla 5.2 muestra el resumen de los resultados obtenidos. Para cada mapa se calculó una ruta de cobertura utilizando los distintos algoritmos de *coverage path planning* descritos previamente en la Subsección 3.2.1. Para cada uno de estos mapas y planes de cobertura se calcularon las métricas descritas anteriormente en la Subsección 5.1.1.

Adicionalmente, se incorporaron algunos resultados cualitativos, los cuales se pueden observar en las Figuras 5.1 y 5.2. Estos resultados permiten observar de manera visual las diferencias entre los planes que entregan los distintos algoritmos de *coverage path planning*.

A cada algoritmo se le fue asociado un número para facilitar la referencia a estos en las Secciones 5.1.3 y 5.3.3 donde se presentan los resultados y su correspondiente análisis. La asignación hecha entre algoritmo y numeración puede observarse en la Tabla 5.1.

Tabla 5.1: Asignación algoritmo y número.

Algoritmo	**Correspondencia**
Problema del viajero usando grillas	1
Descomposición Boustrophedon	2
Redes neuronales bio-inspiradas	3
Planificación basada en colocación de sensores convexos	4
Minimización local de energía basado en grillas	5
Planificación basada en líneas de contorno	6

Tabla 5.2: Resumen resultados *benchmark*.

Algoritmo	$C_{\%}$ **[%]**	$\mathbf{T}_c$ **[seg]**	P_{len} **[m]**	θ_T **[rad]**	$\rho_{\textbf{path}}$
Problema del viajero usando grillas	98.93 ± 1.29	3.47 ± 3.82	56.045 ± 35.47	68.01 ± 31.01	1.30 ± 0.1
Descomposición Boustrophedon	86.56 ± 5.57	1.34 ± 0.73	54.25 ± 35.24	29.89 ± 12.21	0.19 ± 0.02
Redes neuronales bio-inspiradas	96.04 ± 2.65	0.03 ± 0.03	205.12 ± 174.65	57.94 ± 52.51	1.46 ± 0.09
Planificación basada en colocación de sensores convexos	98.93 ± 0.01	3.37 ± 3.75	61.80 ± 33.82	67.39 ± 30.82	1.30 ± 0.07
Minimización local de energía basada en grilla	96.27 ± 2.54	0.008 ± 0.004	75.88 ± 49.12	38.51 ± 14.69	1.45 ± 0.10
Planificación basada en lineas de contorno	62.72 ± 11.16	0.06 ± 0.02	69.35 ± 36.55	52.01 ± 19.12	2.22 ± 0.85

5.1.3. Análisis

Los resultados obtenidos en el *benchmark* han permitido observar que los algoritmos con mejores resultados en términos de *coverage* son los algoritmos 1 y 4, seguidos de cerca por los algoritmos 5 y 3. Los algoritmos con resultados mas bajos para la métrica de *coverage* predicho son el algoritmo 2 y 6. Lo anterior dista con los resultados vistos en el *benchmark* que se realiza en el trabajo original de los autores de la implementación [74], donde para los seis algoritmos se obtuvieron valores de cobertura altos. Esta diferencia en los resultados puede explicarse por la manera en que se realiza el calculo. En [74] el calculo de la cobertura del plan se realiza simulando la ruta que seguiría el robot al ejecutar el plan de cobertura, lo cual para los algoritmos de baja densidad de poses, como el algoritmo 7, marca una diferencia significativa debido a que en este trabajo el calculo se realiza solo con las poses que entrega el plan sin realizar una proyección del camino que seguiría el robot mediante algún algoritmo como A*.

Los tiempos de calculo obtenidos permiten observar el desempeño de los algoritmos. Se observo que el algoritmo 5 presenta un tiempo de calculo notoriamente menor al resto de los algoritmos, siendo el de mas rápido computo. A continuación, se tiene que los algoritmos 3 y 6 son los de mas bajo tiempo de calculo. Para estos algoritmos se tiene un orden de magnitud de diferencia respecto al algoritmo 1, siendo aun así tiempos de calculo bajos en relación a las dimensiones del mapa. Por ultimo, se tiene que los algoritmos 2, 4 y 1 son los de mayor tiempo de calculo, siendo notoriamente mas lentos. Es de fundamental importancia tener en consideración la velocidad con la que los algoritmos resuelven el problema de encontrar la ruta de cobertura pues este es un problema de tipo *NP-hard* lo cual implica que a medida que el tamaño del problema aumenta, en este caso el tamaño del mapa, los tiempos de resolución crecen de manera exponencial. En este caso, los mapas estaban limitados a 700x700 píxeles, teniendo cada mapa una resolución de 0.015 lo cual implica que el máximo tamaño de un mapa, en metros, es de 10.5 x 10.5 lo que es relativamente pequeño para un área de exterior. Respecto a los resultados que se obtuvieron en [74], se tiene que estos son concordantes respecto a esta métrica.

El numero de giros o maniobras que debe ejecutar el robot para completar afecta el rendimiento durante la ejecución. Esto se debe a que ejecutar maniobras de giro es significativamente mas costoso que ir en linea recta, tanto en términos de tiempo que toma realizar el movimiento como en términos de energía. De los resultados del *benchmark*, se observo que el algoritmo que requiere de una menor cantidad de maniobras de giro para cubrir un área es el algoritmo 2, observándose una diferencia significativa respecto la mayoría de los métodos. Como se observa en las Figuras 5.1 y 5.2, los planes obtenidos con el algoritmo 2 son mas sencillos, en el sentido de que el espacio se

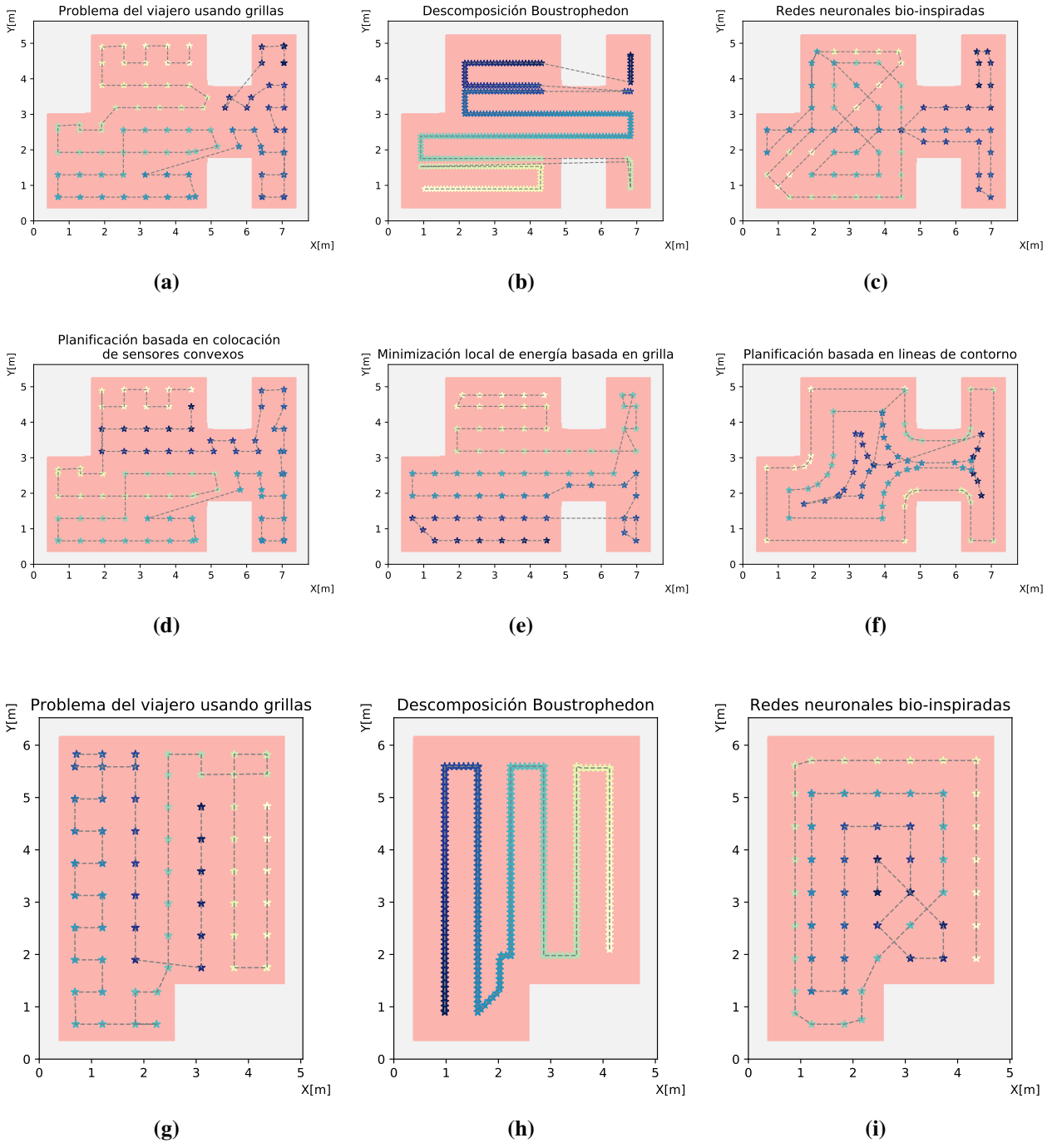


Figura 5.1: Resultados cualitativos *benchmark*. Los píxeles en color representan el espacio a cubrir. Los *waypoints* obtenidos se marcan con estrellas cuyo color indica la temporalidad, esto es, la primera pose es la que tiene asignado el color mas claro y la ultima pose es aquella que tiene asignado el color mas oscuro. Los *waypoints* se unen mediante lineas punteadas para visualizar el flujo del plan.

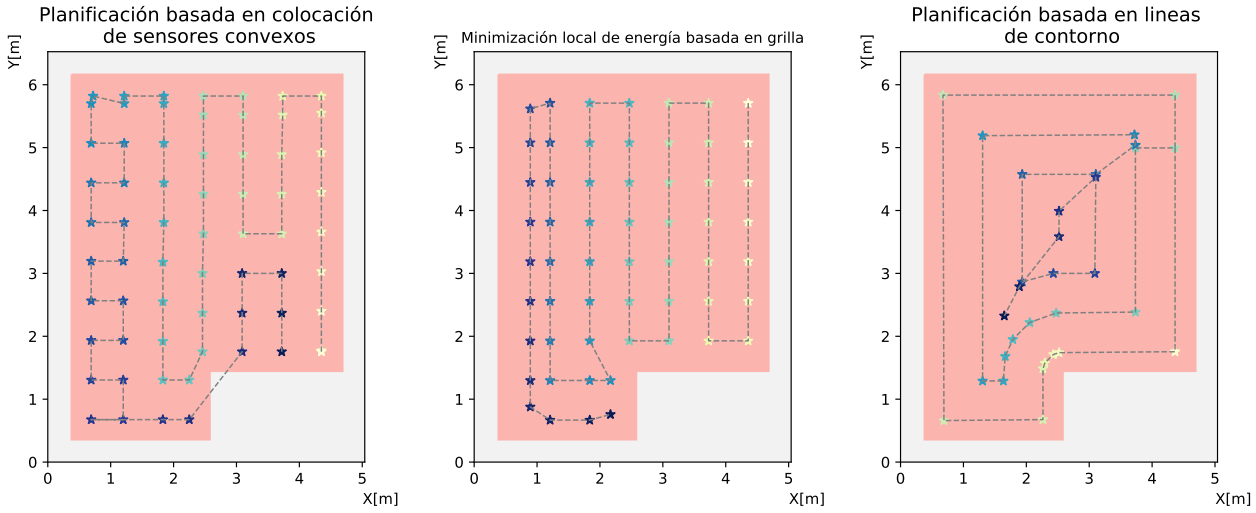


Figura 5.2: Resultados cualitativos obtenidos en *benchmark*, continuación.

cubre principalmente con lineas rectas. Para el resto de los algoritmos se puede observar que hay una mayor cantidad de giros, los planes son mas complejos, teniéndose una considerable mayor cantidad de giros. Los resultados obtenidos para esta métrica son concordantes con los de [74], donde se calcula una métrica similar pero solo considerando las diferencias en ángulo que sean superiores a un cierto umbral, lo cual en este trabajo no se considera ya que todas estas diferencias angulares, incluso las pequeñas, influyen en el desempeño de la ejecución del plan.

El largo del plan es una variable que afecta directamente en tiempo de ejecución del plan. De la Tabla 5.2, observamos que el planificador que obtiene planes mas cortos es el 3, seguido de cerca por el algoritmo 1. El algoritmo que presenta planes mas largos, notoriamente, es el 3. Es importante notar que un plan mas corto puede no necesariamente ser lo mejor, esto debido a que podría darse el caso de que el algoritmo entrega planes que entregan una cobertura deficiente, lo cual implica que el plan entregado no es óptimo en el sentido de ser el plan mas corto que permite cubrir una cierta área. Esto ultimo es lo que sucede en el caso del algoritmo 2. Si bien el largo del plan es corto, este no cubre de la mejor manera el área ya que se obtuvieron coberturas bajas respecto a los otros algoritmos. Podemos observar cualitativamente lo que implica tener un plan mas largo o corto en las las Figuras 5.1 y 5.2, donde observamos que para el algoritmo 3 el plan tiende a ser redundante, con muchos giros y para el algoritmo 2 observamos planes simples en zig-zag pero que no permiten cubrir perfectamente el área.

Respecto a la densidad de *waypoints* de los planes de cobertura, se observa que el algoritmo 2 es el que presenta una mayor densidad de poses, estando estas distanciadas, en promedio, a una distancia de 0.2 veces el diámetro del robot. En contraste, para el algoritmo de planificación basada en lineas de contorno se tiene la menor densidad de poses, estando estas bastante alejadas entre si, se observa que en promedio la distancia entre poses consecutivas es de 2 veces el radio del robot. Lo anterior permite explicar la discrepancia entre el porcentaje de *coverage* obtenido en [74] y en la Tabla 5.2 ya que en este trabajo para calcular la cobertura que un plan entrega se utilizan las poses únicamente y no se realiza una proyección del camino que seguiría el robot si ejecutara

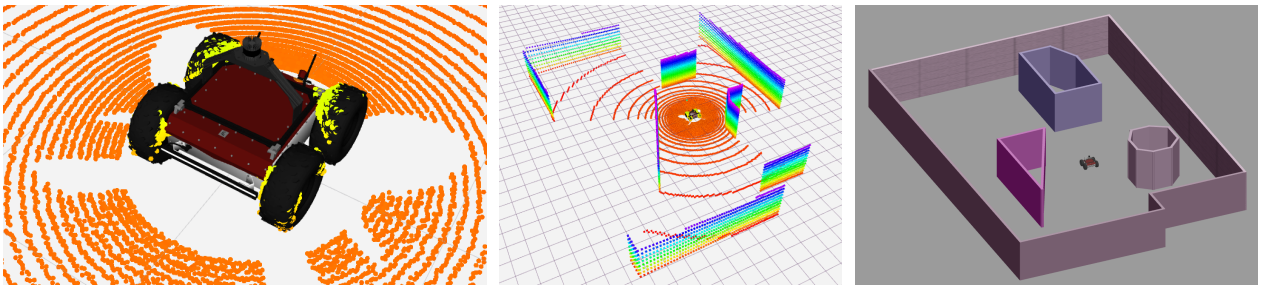
el plan. Esto puede observarse cualitativamente comparando los resultados para el algoritmos 6 y los otros algoritmos que se observan en las Figuras 5.1 y 5.2. Para el resto de los algoritmos, se observa que en promedio la distancia entre poses es de 1.4 veces el diámetro del robot. El conocer esta caracteristicas de los algoritmos es sumamnete relevante debido a que si los *waypoints* se encuentran muy separados, el *stack* de navegación tiene mas probabilidades de salirse del plan original, como se vera en la Sección 5.3.3.

Como resultado principal del *benchmark*, se observa que el algoritmo 5 es el que presenta mejores características a nivel general. Aún con esto, para la mayoría de los algoritmos se obtuvo un buen desempeño por lo cual vale la pena someterlos a evaluación en simulación para observar que diferencias se aprecian al realizar la ejecución del plan de cobertura y observar que o como factores como largo del plan o densidad de *waypoints* afectan a la ejecución. Esto es realizado en la Sección 5.3.3 donde se trabaja primeramente con un robot y ambientes simulados en `Gazebo` y luego estos son puestos aprueba en el mundo real.

5.2. Diseño experimental

5.2.1. Preprocesamiento sensorial

Para poder realizar pruebas tanto en simulación como en el mundo real, es necesario que la base móvil, cualquiera sea esta, cuente con alguna fuente de información que le permita adquirir información de su entorno. Con este fin, se utiliza un Ouster OS0 el cual corresponde a un LiDAR 3D con campo de visión de 360° horizontal y 90° de campo de visión vertical. Como salida, este sensor entrega una nube de puntos como la que se observa en la Figura 5.3.

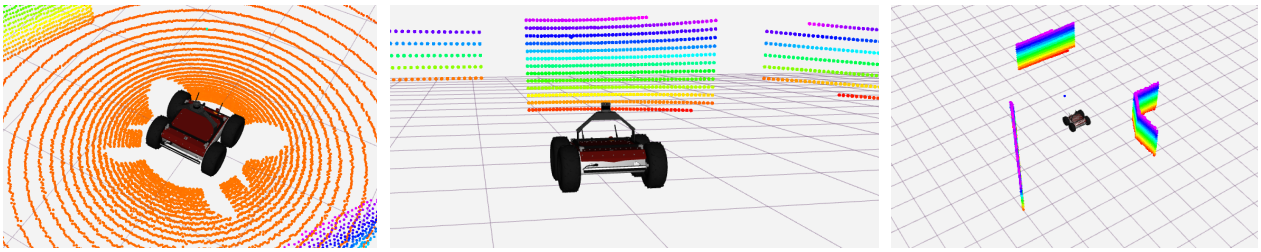


(a) **(b)** **(c)**

Figura 5.3: Data capturada por el sensor Ouster OS0, en simulación. De izquierda a derecha: (a) vista completa de data obtenida, (b) acercamiento de data obtenida, (c) mundo utilizado. Como se observa, el sensor detecta el suelo y partes del robot por lo cual se hace necesario filtrar estos puntos. El sensor al tener un FoV de 90° en vertical puede detectar objetos que están por encima del robot como el techo de un espacio cerrado o la copa de un árbol.

Para utilizar la información que proporciona el sensor se implemento un nodo de ROS en C++ que aplica una serie de filtros que permiten eliminar aquellos datos de la nube de puntos que no son de interés. Estos filtros corresponden a un filtro de caja para poder eliminar los puntos que corresponden al cuerpo del robot mismo, como se observa en la Figura 5.4a, un filtro de recorte en el eje z para eliminar los puntos correspondientes al suelo y aquellos que se encuentran muy por

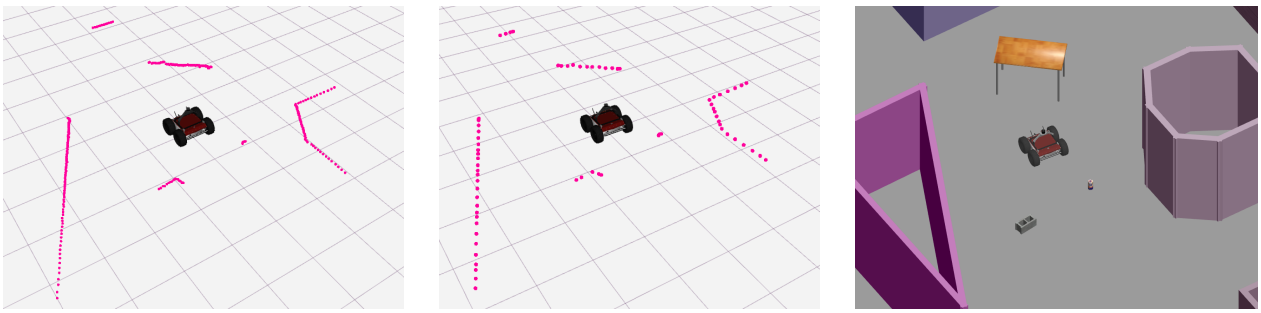
encima del robot como para ser considerados como obstáculos, como se observa en la Figura 5.4b. El tercer filtro aplicado permite recortar radialmente la nube de puntos, esto es, permite ajustar el rango de alcance máximo del sensor. Esto se puede observar en la Figura 5.4c.



(a) **(b)** **(c)**

Figura 5.4: Resultados al aplicar secuencialmente distintos filtros sobre la nube de puntos. De izquierda a derecha: (a) Resultado tras aplicar un filtro de caja. Se observa que los puntos sobre el robot han sido filtrados. (b) Resultado tras aplicar sobre el resultado de un filtro para cortar la nube de puntos. Se observa que los puntos pertenecientes al suelo y parte de las paredes han sido filtrados de la nube. (c) Resultado tras aplicar un filtro para limitar el alcance de la nube de puntos tras la utilización de los otros dos filtros descritos. En la imagen se utilizo un limite de 5 metros de distancia del robot como rango máximo.

Posteriormente, los datos obtenidos son transformados a formato de escáner láser. El objetivo de esto es compactar los datos, conservando la información perceptual relevante. Por este motivo, al realizar esta compactación, se toma para cada ángulo de visión, el punto que se encuentre mas cercano al robot. Adicionalmente, sobre este escáner generado se aplicó un filtro estadístico, para poder filtrar mediciones espurias, *outliers*, que se producen debido a, por ejemplo, reflexión de la luz en la superficie del robot, el suelo, etc. El resultado de la aplicación de ambas cosas puede verse en la Figura 5.11a.



(a) **(b)** **(c)**

Figura 5.5: Resultado obtenido al transformar la nube de puntos con filtros aplicados a formato láser. Como se observa en (a), la transformación permite convertir la nube filtrada a formato de láser, siendo aún posible detectar objetos que no podrían ser visualizados con lidars 2D convencionales si estos no se encuentran a una altura adecuada. En la imagen se observa como es posible, por ejemplo, detectar objetos como mesas que son de difícil detección para un láser convencional pues solo hay una única linea de vista en lugar de múltiples como se observa al contrastar con la Figura 5.4b. En (b), se observa el resultado al convertir el escáner resultante en (a) al formato requerido por el planificador local basado en aprendizaje reforzado. En (c) se muestra la escena sobre la cual se coloco el robot.

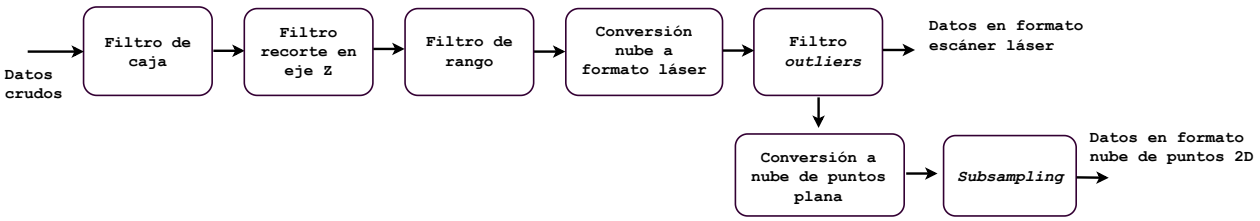


Figura 5.6: Diagrama de bloques filtros y transformaciones aplicadas sobre la nube de puntos.

Para poder alimentar el *stack* de navegación que utiliza un planificador basado en aprendizaje reforzado se construyo un nodo adicional que permite adaptar el formato de escáner láser a nube de puntos plana (2D) que requiere el sistema para funcionar y baja la resolución de la nube resultante al aplicar un filtro tipo voxel el cual aplica un submuestreo sobre los datos. El resultado de la aplicación de ambas cosas sobre el escáner se observa en la Figura 5.11b. El flujo completo desde los datos crudos que se reciben desde el sensor hasta las salidas como escáner láser y nube de puntos plana puede apreciarse en el diagrama de bloques que se presenta en la Figura 5.6.

5.2.2. Ambiente simulado

Simulación del robot móvil

Para la realización de las pruebas en simulación se trabajó con la base móvil Panther de Husarion. Este robot es de tipo *skid-steered*, teniendo dos pares de ruedas opuestas. A este robot se le equipó un LiDAR Ouster[3].

Las medidas del robot simulado con la montura del sensor se pueden observar en la Figura 5.7

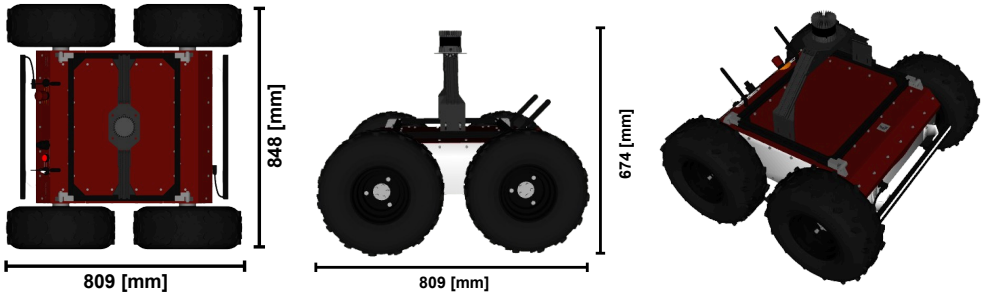


Figura 5.7: Robot Panther.

Simulación de ambientes de prueba

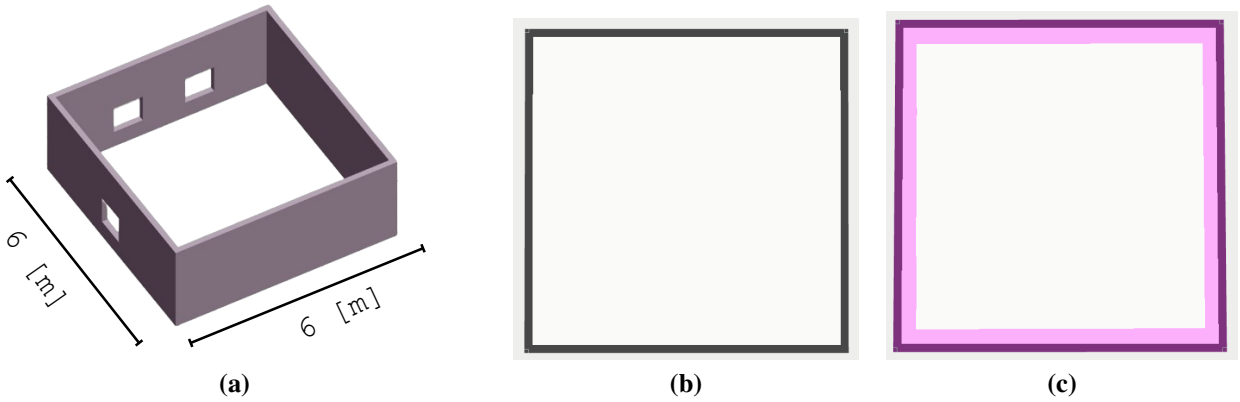
Para la evaluación en simulación se diseñaron y construyeron dos mundos en GAZEBO, un ambiente tipo *indoor* y un ambiente tipo *outdoor*. Estos ambientes pueden observarse en las Figuras 5.8 y 5.9.

El primer mundo corresponde a un ambiente simple de forma cuadrada. Se busca evaluar la

[3]La simulación de este sensor y los controladores del sensor real fueron obtenidos del repositorio oficial del proveedor (`https://github.com/ouster-lidar`). El plugin de Gazebo utilizado para este sensor fue desarrollado por AMTC.

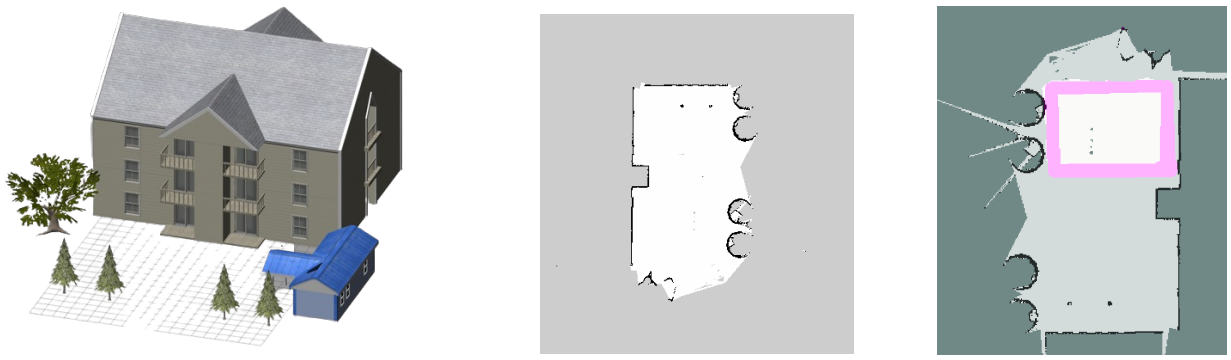
funcionalidad del sistema en formas básicas y ver como reacciona ante la presencia de paredes, las cuales desde el punto de vista del *stack* de navegación corresponden a obstáculos. Este aspecto es relevante a evaluar pues es posible que se presenten inconvenientes en la ejecución cuando el robot se acerca mucho a una esquina, pudiendo quedar estancado.

El segundo ambiente de prueba desarrollado es de tipo exterior, en este ambiente se busca evaluar el comportamiento del sistema en un ambiente libre de paredes, tipo exterior. Adicionalmente, en este ambiente se desarrollara una prueba de evasión de obstáculos no identificados lo cual es crucial a evaluar ya que para el despliegue en una aplicación real es de vital importancia que el robot pueda navegar sin chocar.



(a) **(b)** **(c)**

Figura 5.8: Ambiente de simulación tipo interior simple. (a) mundo en simulación. Dimensiones del mundo son 6 metros tanto de largo como ancho. (b) mapa creado a partir de *gmapping*. (c) Zona a cubrir en experimento delimitada. En este caso se cubre toda el área.



(a) **(b)** **(c)**

Figura 5.9: Ambiente de simulación tipo exterior simple (34 metros de ancho, 55 de largo). (a) mundo en simulación. (b) mapa creado a partir de *gmapping*. (c) Zona a cubrir en experimento delimitada. Área a cubrir es de 22 metros de largo y 14 de ancho.

5.2.3. Ambiente real

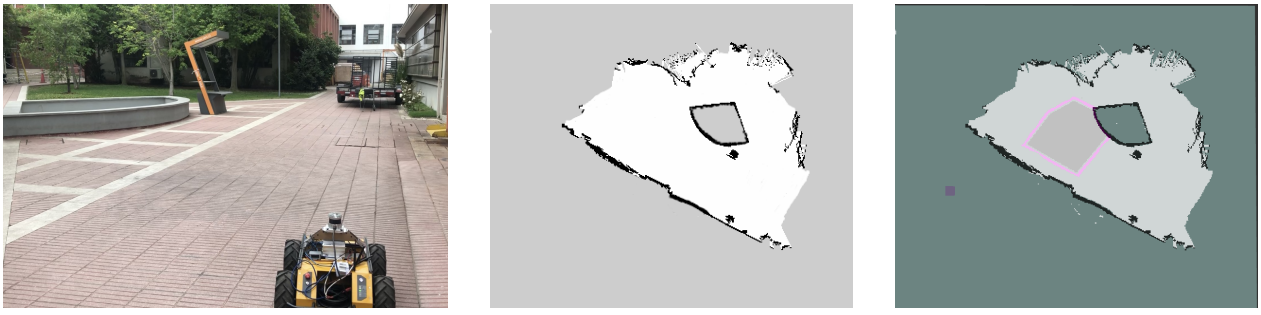
Configuración experimental (robot)

Para el despliegue del sistema en el mundo real, se trabajó con el mismo setup de las simulaciones, esto es, la base móvil utilizada fue Panther de Husarion, el sensor utilizado un Ouster OS0[4] y la montura para el sensor corresponde a la misma que es simulada. Las medidas del robot real son las que se muestran en la Figura 5.7, se adjuntan fotos del montaje en mundo real utilizado en la Figura 5.10, donde se aprecia que la configuración utilizada en simulación es idéntica al de despliegue en mundo real.

Figura 5.10: Robot Panther con montaje de sensor Ouster OS0 en mundo real.

Ambiente de pruebas

La evaluación en el mundo real se realizó en el patio al costado del AMTC, ubicado en el departamento de ingeniería eléctrica de la universidad de Chile. Particularmente, se trabajó en una sección del patio la cual se puede apreciar el la Figura 5.11, donde se adjunta un mapa del área total obtenido mediante *gmapping*. Adicionalmente, se demarca el área de trabajo donde se realizaron las pruebas. Esta área de trabajo se caracteriza por tener como obstáculo una pileta en el centro, la cual tiene un lado curvo que se utilizará para poner a prueba el sistema frente a obstáculos no poligonales.



(a) **(b)** **(c)**

Figura 5.11: Área al costado del AMTC. (a) Foto de la zona en mundo real. (b) mapa de la zona (obtenido con GMAPPING). Los pixeles blancos indican espacio libre, negro espacio con obstaculos y gris desconocido. (c) subsección del área a cubrir marcada (en rosado) en el mapa.

[4]Información en detalle de este sensor puede ser consultada en la pagina del proveedor: `https://ouster.com/products/hardware/os0-lidar-sensor`

5.3. Evaluación del sistema de *coverage navigation*

5.3.1. Pruebas realizadas

Con el fin de evaluar el desempeño del sistema, se realizaron las siguientes pruebas:

1. **Pruebas en simulación:** Se busca evaluar el comportamiento del sistema en distintos escenarios. En estas pruebas no se trabajará con obstáculos no previamente identificados, por lo tanto el área de trabajo es estática y conocida a priori por el robot. Se tiene como objetivo observar como distintas características del área a cubrir afectan el desempeño del sistema y sus componentes. Estas pruebas se realizaron con ambas versiones de los *stack* de navegación implementados (ver Sección 2.1.1) y con los seis algoritmos de CPP de `ipa_room_exploration`.
2. **Pruebas con obstáculo en simulación:** Se busca evaluar la reactividad del sistema frente a obstáculos dinámicos. Se tiene como objetivo observar si el sistema es capaz de evadir estos obstáculos y si es capaz de replanificar en las áreas que queden sin cubrir. Estas pruebas se realizaron con ambas versiones de los *stack* de navegación implementados y con dos de los algoritmos de CPP de `ipa_room_exploration`, que corresponden a aquellos de mayor y menor densidad de waypoints, según lo obtenido en el *benchmark* de la Sección 5.1.
3. **Pruebas en mundo real:** Se busca desplegar y evaluar el funcionamiento del sistema en el mundo real. El ambiente donde se realizaron estas pruebas fue el costado del AMTC, el cual se muestra en la Figura 5.11. Las pruebas realizadas se realizaron sin la presencia de obstáculos no definidos en el mapa. Para estas pruebas se trabajó con el *stack* de navegación basado en aprendizaje reforzado y con los seis algoritmos de `ipa_room_exploration`.

5.3.2. Métricas de desempeño

El desempeño del sistema se calcula mediante dos indicadores, los cuales son porcentaje de *coverage* obtenido tras la ejecución del plan y tiempo de ejecución. Estas son las variables más importantes para el sistema en ejecución, pues el porcentaje de *coverage* nos indica que tanto se logra cubrir el área incluso frente a imprevistos, mientras que el tiempo de ejecución nos permite conocer que tan eficientemente el sistema puede desempeñar la tarea.

Un porcentaje de *coverage* bajo indica que el sistema no logra cumplir su tarea eficazmente; por otra parte, un tiempo de ejecución muy alto indica que el sistema no logra ejecutar la tarea de forma eficiente. Luego, lo deseable es obtener porcentajes de *coverage* altos y tiempos de ejecución acordes al área que se busca a cubrir.

5.3.3. Análisis y Resultados

Resultados de pruebas en simulación

A continuación se muestran los resultados obtenidos en las pruebas de simulación ejecutadas. Cada gráfico contiene información respecto a la trayectoria ejecutada por el robot. Se muestra en el fondo de cada gráfica el mapa donde se ejecutó el sistema, el plan de cobertura obtenido utilizando estrellas que representan los *waypoints*, y el camino recorrido efectivamente por el robot. El mapa de colores indica la temporalidad de las poses del plan y del robot, esto es, indica cuales poses y *waypoints* fueron visitadas primero por el robot y cuales fueron visitadas al final. En los gráficos,

para referirse a un resultado obtenido con el *stack* de navegación de `move_base` se utiliza la sigla DWA (*Dynamic Window Approach*), que corresponde al planificador local utilizado en este *stack*; por otro lado, para referirse al *stack* de aprendizaje con planificador local basado en aprendizaje reforzado, se utiliza la sigla RL por aprendizaje reforzado en inglés (Reinforcement Learning). En esta sección también se utiliza la asignación algoritmo - número que se detalla en la Tabla 5.1.

Resultados y análisis prueba en simulación en entorno cerrado:

Métrica	***Stack***	**Alg1**	**Alg2**	**Alg3**	**Alg4**	**Alg5**	**Alg6**
Tiempo ejecución [min]	RL	3.51	**2.42**	**2.42**	3.34	3.34	6.23
	DWA	16.00	7.45	11.57	12.15	23.28	21.33
Cobertura total [%]	RL	92.11	79.28	90.61	90.41	87.43	97.12
	DWA	94.21	69.11	93.94	92.69	87.02	**97.41**

Tabla 5.3: Resultados obtenidos en prueba de simulación en entorno cerrado simple.

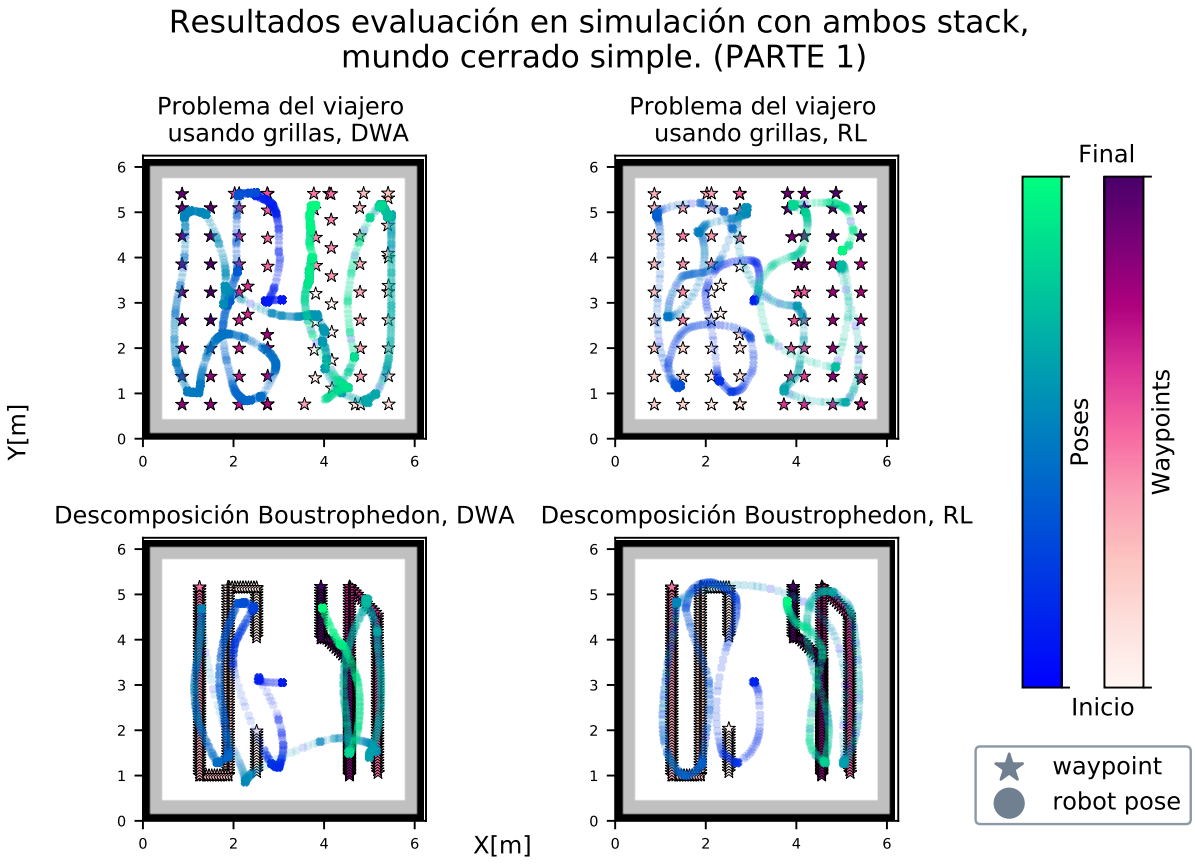


Figura 5.12: Resultados obtenidos en simulación utilizando los seis algoritmos de *coverage path planning* y ambos *stack* de navegación. DWA corresponde al *stack* de navegación de `move_base` y RL al *stack* con planificador local basado en aprendizaje reforzado. Al lado izquierdo se encuentran los resultados para el *stack* de `move_base` y al lado derecho los de el *stack* con aprendizaje reforzado. Los resultados se dividen en dos imágenes, en esta se muestran los resultados de los primeros dos algoritmos.

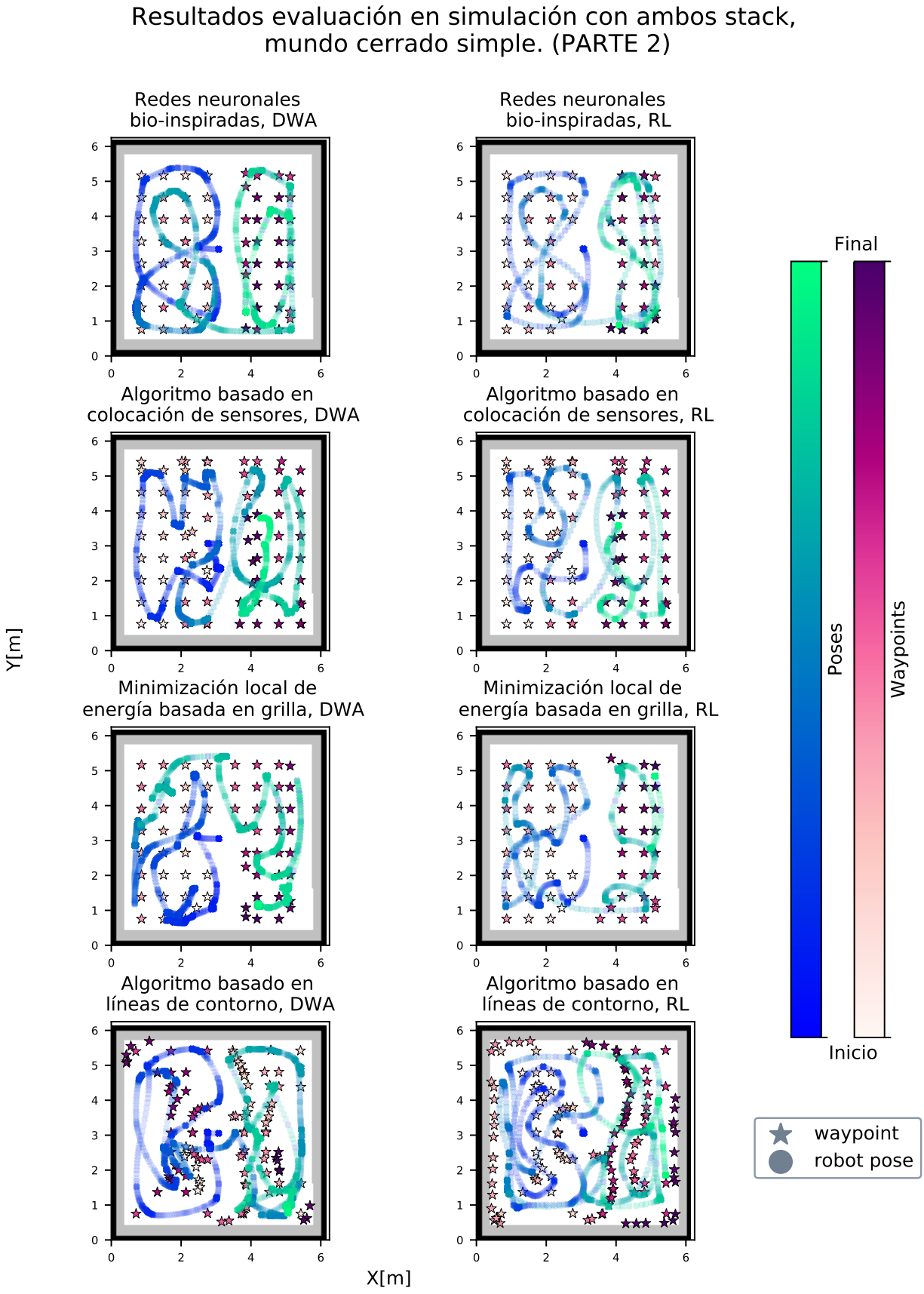


Figura 5.13: Resultados obtenidos en simulación en entorno cerrado simple, parte 2.

La ejecución de esta prueba permitió observar como los *stack* de navegación ejecutan los planes de cobertura dados por los distintos algoritmos de *coverage path planning*. Se observaron mayoritariamente resultados positivos en términos de cobertura obtenida, teniéndose que el algoritmo 6, el cual obtuvo uno de los peores resultados en esta métrica en el *benchmark* de la Sección 5.1, obtuvo los mejores resultados obteniendo una cobertura de aproximadamente un 97 % tanto en el caso del uso del *stack* de `move_base` como el de RL. Por el contrario, la cobertura mas baja fue alcanzada por el algoritmo 2, con el uso de ambos *stack*. Esto se puede explicar con el hecho de que los planes entregados por este planificador de rutas son cortos, no siendo suficientes para abarcar toda el área. Esto puede corroborarse visualmente en la Figura 5.12.

En cuanto a tiempos de ejecución, podemos notar diferencias significativas en torno al *stack* que se utilizó, teniendo resultados positivos para el *stack* de RL y resultados negativos para el *stack* de DWA. Es importante notar que para el *stack* de `move_base` los tiempos de ejecución son varias veces superiores a los obtenidos con el otro *stack*. Esto sucedió principalmente debido a que el *stack* de `move_base` se quedaba por periodos prolongados de tiempo atascado en las esquinas o lugares cercanos a las paredes, lo cual aumentó significativamente el tiempo de ejecución del plan. Aun con esto, este *stack* obtuvo bueno resultados en términos de cobertura, siendo en algunos casos ligeramente mejor que la obtenida con el *stack* de RL.

Siguiendo con los tiempos de ejecución, para el algoritmo de mejor cobertura en este experimento, algoritmo 6, se observaron tiempo de ejecución altos. Esto es particularmente notorio al utilizar el *stack* de RL donde el tiempo de ejecución es aproximadamente el doble que el de los otros casos. El segundo algoritmo que ofreció mejor cobertura fue el algoritmo 1. En este caso, para el *stack* de RL se observó un tiempo de ejecución mas bajo por lo cual se puede decir que este algoritmo ofrece un mejor desempeño en términos de relación cobertura vs tiempo de ejecución.

Respecto a lo sucedido durante la ejecución, es posible observar comportamientos distintos según el algoritmo de CPP elegido para el cálculo de la ruta de cobertura. Como se observa en las Figura 5.12, para el algoritmo 2, descomposición Boustrophedon, se observa una ejecución más cercana a lo que es el plan de cobertura original, esto puede deberse a que el plan es altamente denso en *waypoints* y por ende el robot tiene poco espacio para tomar una ruta distinta. Por otro lado, para los otros algoritmos se observa una ruta de ejcucción notoriamente distinta a a entregada por el plan, no alcanzandose todos los *waypoints*. Esto puede suceder por un *timeout* del *stack* debido a que una pose no pudo ser alcanzada en un cierto tiempo debido, por ejemplo, a que la pose esta en un espacio muy estrecho como una esquina y el robot no es capaz de planificar como llegar a ese punto sin colisionar.

Respecto a la ejecución según *stack*, si bien los caminos seguidos son distintos, no se aprecian grandes diferencias en las trayectorias que cada *stack* ejecutó.

Resultados y análisis prueba en simulación en entorno abierto:

Métrica	*Stack*	Alg1	Alg2	Alg3	Alg4	Alg5	Alg6
Tiempo ejecución [min]	RL	5.24	**4.18**	7.01	5.21	5.47	7.58
	DWA	24.58	10.15	17.43	24.17	21.04	18.50
Cobertura total [%]	RL	**95.90**	90.22	95.73	95.07	94.53	89.18
	DWA	95.07	84.86	94.59	94.90	93.73	92.42

Tabla 5.4: Resultados obtenidos en prueba de simulación en entorno abierto simple

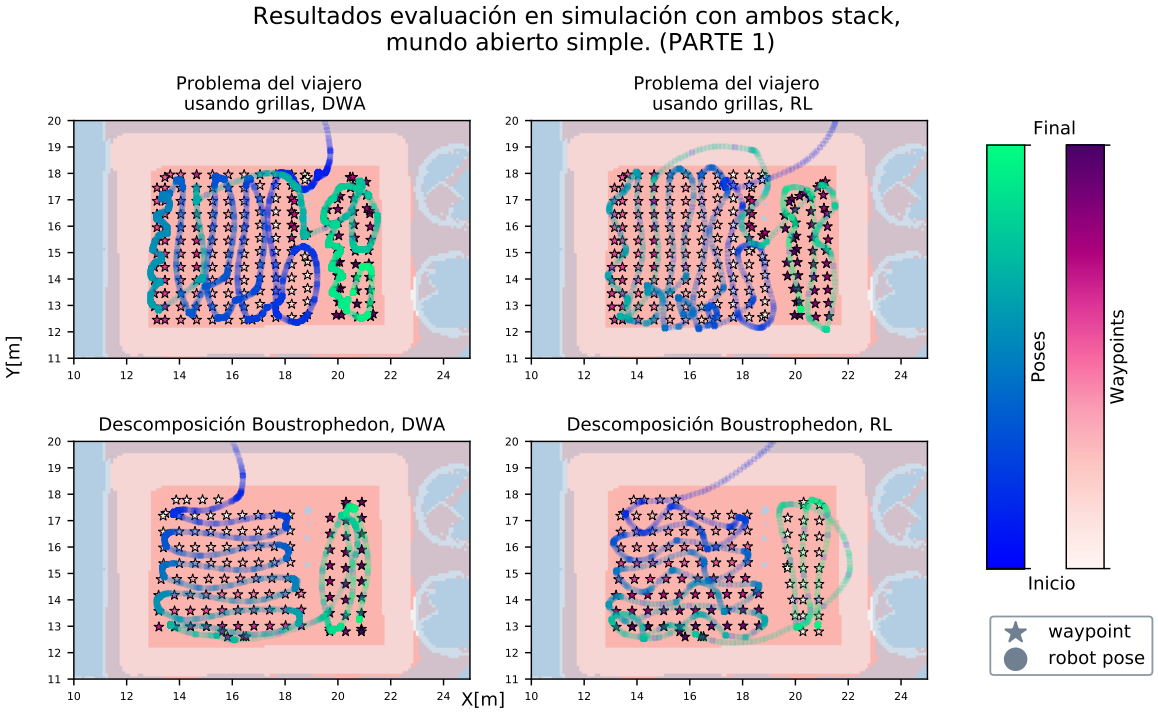


Figura 5.14: Resultados obtenidos en simulación en mundo abierto simple. Se utilizaron los seis algoritmos de *coverage path planning* y ambos *stack* de navegación (DWA corresponde al de `move_base` y RL al de aprendizaje reforzado) Al lado derecho se encuentran los resultados para el *stack* de `move_base` y al lado izquierdo los de el *stack* con planificador local basado aprendizaje reforzado. Los resultados se dividen en dos imágenes, en esta se muestran los resultados de los primeros dos algoritmos.

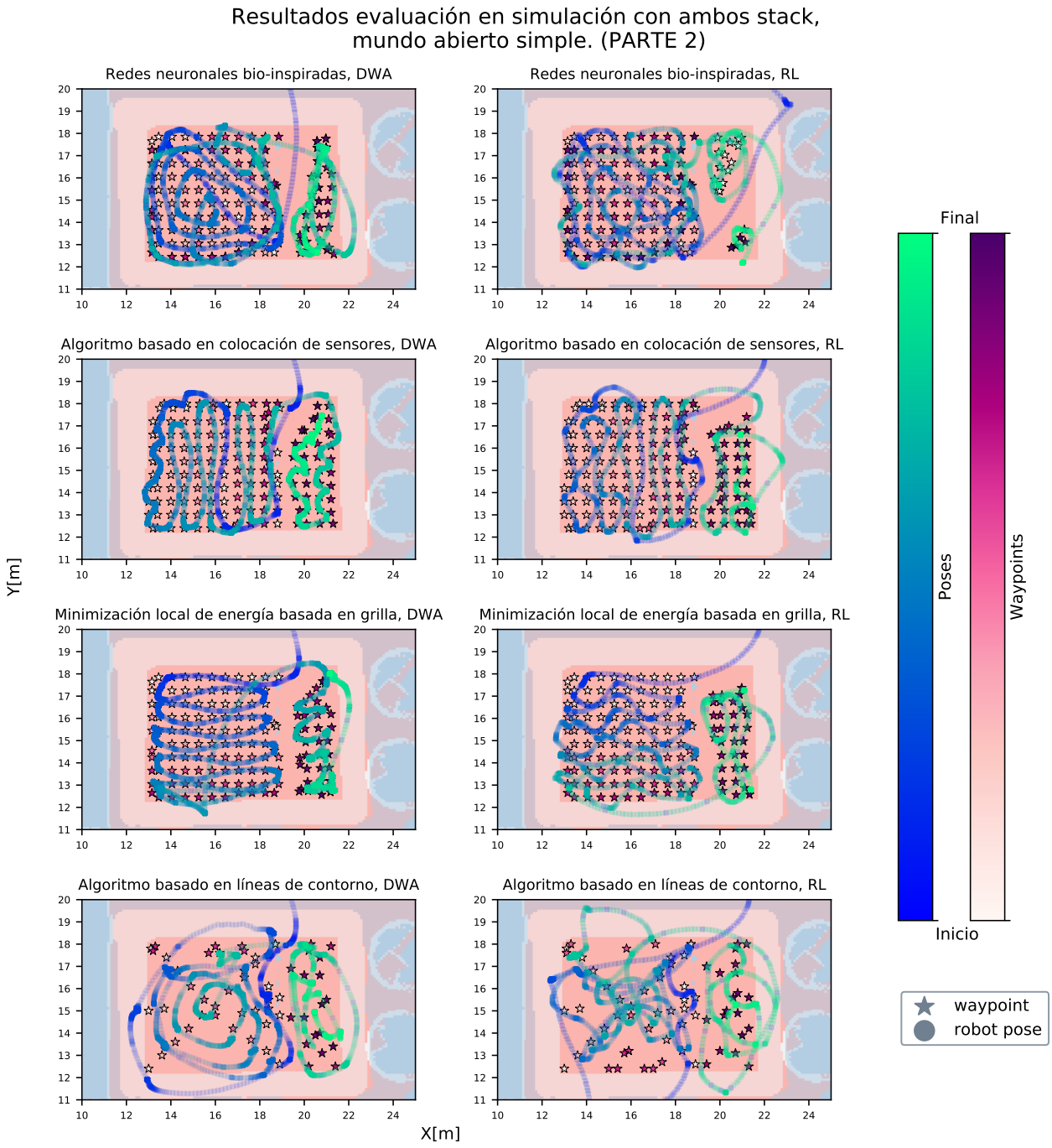


Figura 5.15: Resultados obtenidos en simulación en entorno abierto simple, parte 2.

Los resultados obtenidos en esta prueba permitieron observar como es el desempeño del sistema en ambientes de tipo exterior. Se observaron resultados de cobertura más altos respecto a los vistos en la prueba con un ambiente tipo *indoor* en la Sección 5.3.3. Esto puede deberse a que la zona a cubrir no posee límites como paredes donde el robot tenga problemas para alcanzar.

Al igual que lo visto en la prueba anterior, la cobertura alcanzada tras la ejecución del plan alcanza valores altos, teniendo más de un 90 % del área cubierta en la mayoría de los casos. Para esta prueba, el algoritmo con mejor *coverage* fue el algoritmo 1, seguido de cerca por los algoritmos

3 y 4. El algoritmo con peor desempeño fue el número 6, que en la prueba anterior obtuvo la mejor cobertura. Esto se puede deber a que en este caso el robot se alejó significativamente del plan de cobertura debido a la baja densidad de *waypoints* y la falta de paredes que eviten que el robot se salga del área de cobertura como se observa en la Figura 5.15 para este algoritmo.

En este caso, se obtuvieron resultados de cobertura ligeramente mejores para el *stack* de navegación con el planificador local basado el RL, exceptuando al algoritmo 6.

En cuanto a los tiempos de ejecución, estos fueron más altos que los obtenidos en la prueba con ambiente tipo interior debido a que el espacio a cubrir es más amplio. En este caso también se observó que el *stack* de RL obtuvo tiempos de ejecución notoriamente menores a que los obtenidos por el *stack* de `move_base`. A diferencia de la prueba anterior, en este ambiente no hay presencia de paredes que expliquen el por que de la notoria diferencia de ejecución por lo cual se puede decir que el *stack* de `move_base` ofrece de manera general una *performance* mucho más baja que el *stack* de RL. El tiempo de ejecución más bajo registrado fue con el uso del algoritmo 2, tanto en el caso de RL como DWA. Esto se debe a que el plan entregado por este algoritmo es más corto y por ende toma menos tiempo ejecutarlo. El segundo mejor tiempo obtenido fue con el uso del algoritmo 4, seguido muy de cerca por el algoritmo 1. Ambos siendo ejecutados por el *stack* RL. La cobertura obtenida por estos algoritmos y *stack* fueron las más altas, estando por encima del 95 %. A partir de las Figuras 5.14 y 5.14 se observa que los planes entregados y ejecutados por estos algoritmos son bastante similares.

Respecto a las trayectorias seguidas por el robot durante la ejecución, para los algoritmos 1, 2, 4 y 5 se observaron trayectorias similares las cuales corresponden mayoritariamente a movimientos en zig-zag. Se observa que se logra alcanzar la mayoría de los *waypoints* a diferencia de lo visto en la prueba anterior. Para el algoritmo 3, se observó una trayectoria más compleja en el sentido de que el plan tiene una mayor cantidad de giros. Por último para el algoritmo 7, se observó que la baja densidad de *waypoints* afecto severamente la ejecución del plan, obteniéndose rutas poco estructuradas que se alejan del plan de cobertura y más aun, se salen del área de cobertura. Dado esto, es posible observar lo crítico que es tener una densidad de poses de navegación alta para asegurar que el plan se cumplirá lo más cercanamente a lo propuesto por el algoritmo de CPP.

Resultados y análisis prueba en simulación evasión de obstáculo no identificado previamente:

Métrica	***stack***	**Alg1**	**Alg6**
Tiempo ejecución [min]	RL	5.59	**4.23**
	DWA	25.02	32.08
Cobertura total [%]	RL	88.97	80.60
	DWA	**89.10**	86.67

Tabla 5.5: Resultados obtenidos en prueba de simulación de evasión de obstáculo no identificado en mapa.

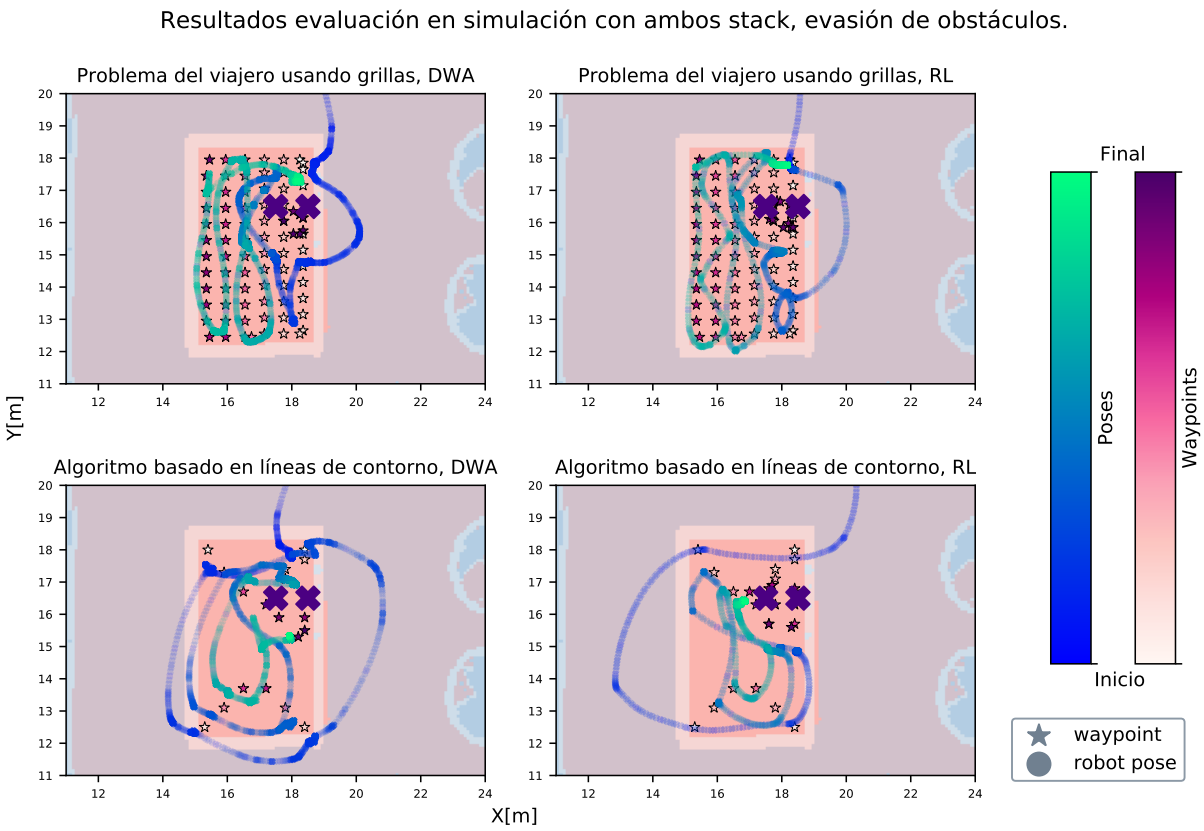


Figura 5.16: Resultados obtenidos en prueba de evasión de obstáculo no identificado en mapa. Las cruces violeta representan donde se colocó el obstáculo.

Mediante esta prueba se ha logrado analizar la capacidad del sistema para evitar colisionar con obstáculos no previamente identificados mientras cubre un área. Para esta prueba se colocó un obstáculo donde marcan las cruces y posteriormente a que el robot pasara por esta área, evadiendo el obstáculo, este fue retirado para observar si el sistema era capaz de realizar una replanificación que permitiera cubrir el área que quedo sin cobertura debido al obstáculo.

Se observó que para ambos *stack* fue posible evadir el obstáculo y luego generar un plan para cubrir esta área. Aun con esto, se observó una baja en la cobertura obtenida, obteniéndose valores ligeramente por debajo del 90 %.

En cuanto a los tiempo de ejecución, nuevamente se observó que el *stack* de aprendizaje reforzado obtuvo mejores resultados. En este caso, este resultado nos indica que este *stack* de navegación es mucho más eficaz en la evasión que el *stack* de `move_base`. Dado esto, para la siguiente prueba, despliegue en el mundo real, se decide solo utilizar el *stack* con planificador de aprendizaje reforzado. Esto se debe a que los tiempos de ejecución muy altos de DWA le quitan practicidad a la aplicación, pudiendo ser poco aceptable que la tarea tome tiempos significativos para completarse. Como para ambos *stack* la cobertura alcanzada es similar, en este sentido no hay mayor preferencia por el *stack* de `move_base`.

En cuanto a los algoritmos utilizados, se probó con dos algoritmos de distinta densidad, usando uno de densidad baja y uno de densidad promedio. Se observa que la baja densidad de poses afecta

negativamente el desempeño de la misma manera que lo visto en la Sección 5.3.3. No hay una mayor influencia en la capacidad de evadir obstáculos respecto a la densidad de poses pero si se observa que la trayectoria que sigue en el caso de mayor densidad se aleja menos del área de cobertura que para el algoritmo de menor densidad.

Resultados pruebas en mundo real

En la Figura 5.17 se muestran y en la Tabla 5.6 se resumen los resultados obtenidos tras desplegar el sistema en el mundo real con el robot Panther. Otras pruebas en el mundo real adicionales fueron ejecutadas con la base móvil Husky y pueden ser vistas en anexos junto a una prueba adicional que fue realizada con la base Panther en la cancha de Beauchef 850, en la facultad de ingeniería de la Universidad de Chile.

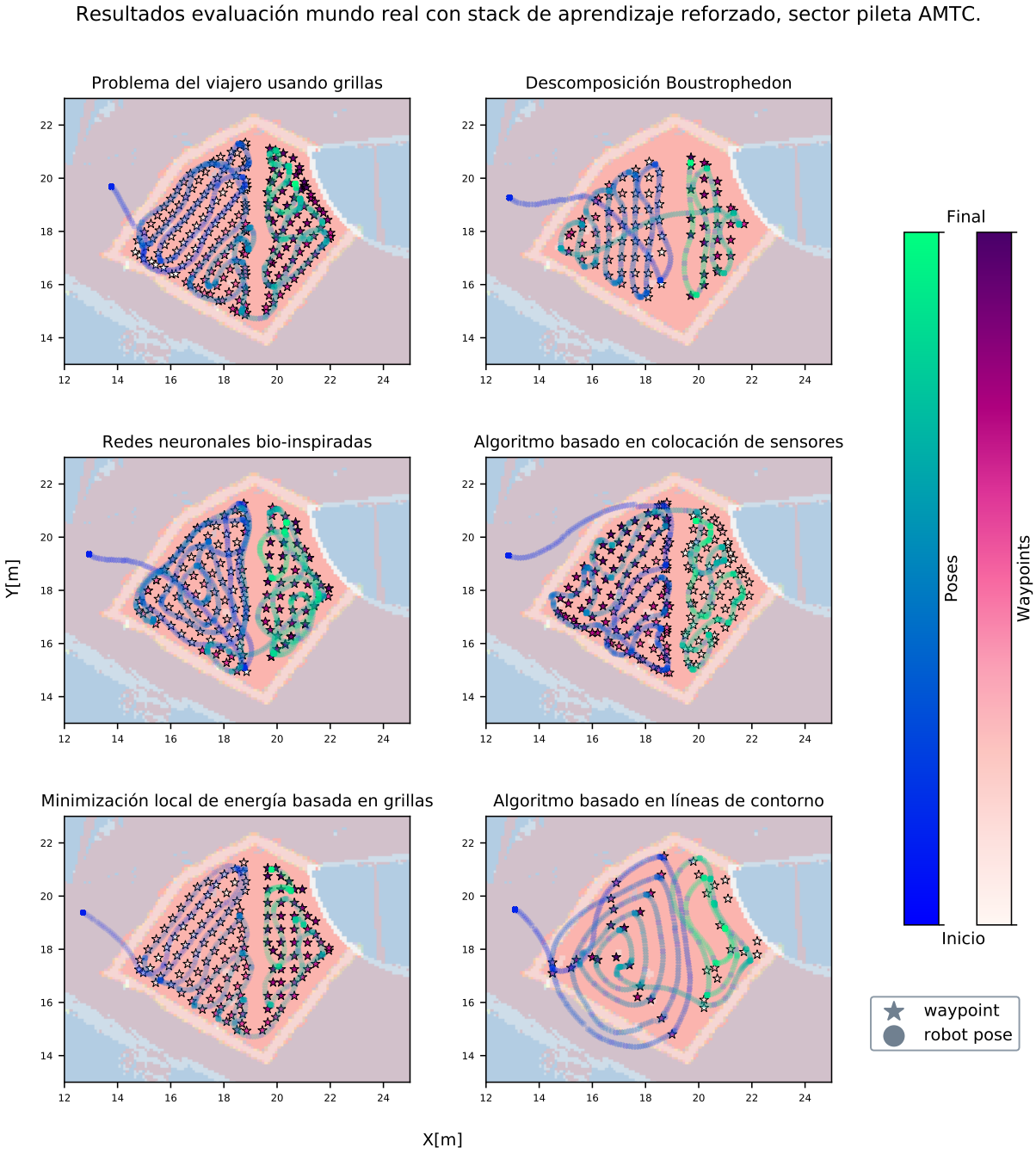


Figura 5.17: Resultados obtenidos en el mundo real utilizando el sistema con el *stack* de navegación en base a aprendizaje reforzado. El lugar de despliegue fue el sector al costado al AMTC, específicamente el área demarcada que se aprecia en la Figura 5.11.

Métrica	***stack***	**Alg1**	**Alg2**	**Alg3**	**Alg4**	**Alg5**	**Alg6**
Tiempo ejecución [min]	RL	4.57	**3.27**	5.27	4.19	3.45	4.12
Cobertura total [%]	RL	**98.25**	88.64	97.41	97.58	97.25	97.02

Tabla 5.6: Resultados obtenidos en prueba en mundo real en costado AMTC.

El despliegue del sistema en el mundo real permitió observar como el sistema se comporta en un ambiente real de tipo exterior. Los resultados obtenidos concuerdan con lo obtenido en las pruebas en simulación vistas anteriormente con lo cual se puede concluir que el sistema es desplegable en el mundo real.

En cuanto a la cobertura, se observaron resultados muy positivos obteniéndose porcentaje altos para todos los algoritmos, cercanos al 97 %. Similarmente, para los tiempos de ejecución, estos se mantuvieron bajos como los vistos en las pruebas de simulación de las Secciones 5.3.3 y 5.3.3. Dado esto, se puede decir que el despliegue del sistema en el mundo real fue exitoso, observándose resultados que evidencian el buen funcionamiento del sistema.

Respecto a la ejecución de los planes, se observa en general un comportamiento ordenado, en el sentido de que el plan seguido por el robot es similar al propuesto por el planificador de rutas de cobertura. Para el caso particular del algoritmo 6, líneas de contorno, se obtuvo una trayectoria de ejecución un tanto más alejada de la ruta de cobertura, nuevamente, este problema se debe a la baja densidad de poses de navegación que son los que guían la ejecución. Aun con esto, se obtuvieron buenos resultados de cobertura y tiempos de ejecución bajos.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

El trabajo realizado en esta memoria ha permitido abordar el problema de navegación de cobertura o *coverage navigation*, perteneciente al área de la robótica móvil. Mediante la utilización de un enfoque clásico para resolver el problema de navegación autónoma robótica y el uso de algoritmos de *coverage path planning* para resolver el problema del cálculo de rutas de cobertura, se logró el desarrollo de un sistema que permite a un robot adquirir la habilidad de cubrir un área eficazmente de manera autónoma. Particularmente, este sistema fue probado para robots *skid-steered*, realizando validaciones tanto en simulación como en el mundo real. Los resultados obtenidos en las distintas pruebas realizadas permiten concluir que el sistema desarrollado cumple con su propósito, esto es, cubrir un área eficientemente, satisfactoriamente.

Los *stack* de navegación desarrollados permitieron dotar a las plataformas móviles de la capacidad de cubrir un área de manera autónoma, más aún, con el *behavior tree* desarrollado se permitió dar una capa de robustez a esta habilidad, al permitir a los robots adaptarse a entornos cambiantes mediante la evasión de obstáculos tanto estáticos como dinámicos, y el recalculó de rutas de cobertura, lo cual permite cubrir áreas que pudieron quedar sin ser visitadas debido a imprevistos en el camino.

Los distintos algoritmos de *coverage path planning* utilizados demostraron ser capaces de entregar planes de cobertura que permiten al robot cubrir la región de interés. Como se observó a través del Capítulo 5, los algoritmos entregan planes de distinta naturaleza, siendo algunos más largos, densos en poses o más complejos en términos de maniobras de giro a ejecutar. Todos estos factores afectan el desempeño que se obtiene al ejecutar el plan, por ejemplo, para la densidad de *waypoints* se observó que una densidad muy baja contribuye negativamente a el porcentaje de cobertura que se obtiene y que un plan muy largo afecta negativamente al tiempo de ejecución.

Los *stacks* implementados se fundamentaron en el enfoque clásico con el cual se resuelve el problema de navegación autónoma, el cual consiste en dividir el problema en varios sub-problemas. Los componentes utilizados para la implementación tomaron rumbos distintos, utilizando para un caso como base la implementación de un *stack* dado por `move_base`, de ROS, y en el otro caso se utilizó como base la implementación de un planificador local basado en aprendizaje reforzado

desarrollado por AMTC. En ambos casos se observaron resultados positivos tanto en simulación como en el despliegue en el mundo real. Sin embargo, en el caso del *stack* basado en el uso de un planificador de aprendizaje reforzado se observaron mejorías importantes en el desempeño respecto a los tiempos de ejecución. Lo anterior sucedió debido a que este planificador local es capaz de evadir de manera más eficiente obstáculos tanto dinámicos como estáticos lo cual reduce significativamente el tiempo de ejecución del plan al no quedarse periodos de tiempo significativos tratando se salir de una esquina, por ejemplo. En cuanto a la cobertura obtenida, se obtuvieron resultados similares para ambos *stack*, observándose resultados satisfactorios.

Lo anterior permitió observar la importancia de la correcta elección y configuración de los distintos componentes del *stack*. Estos deben ser concordantes con el ambiente en el cual se desplegará la aplicación que se busca desarrollar y con el tipo de plataforma que se utilizará. Si bien la implementación basada en `move_base` no produjo resultados deficientes, especialmente en el mundo real, se observó que el planificador local utilizado, DWA, podía causar un deterioro en el rendimiento del sistema al aumentar los tiempos de ejecución debido a la pobre *performance* que se observo al evadir obstáculos y la dificultad para alcanzar ciertas poses, como aquellas cercanas a esquinas o paredes. Lo anterior hace que la solución que utiliza este *stack* sea poco práctica pues los tiempos elevados que demora en completar la tarea podrían ser en algunos casos no viables o aceptables.

6.2. Trabajo futuro

Por otro lado, las validaciones también permitieron identificar falencias en la aplicación desarrollada. Como se pudo ver primero en ambientes de simulación cerrados, el sistema presenta dificultades notorias para alcanzar lugares estrechos como esquinas, pudiendo cubrir de forma deficiente estas áreas, lo cual deriva en una inherente baja en el resultado obtenido. Asímismo, se pudo observar que la presencia de obstáculos no identificados y cambios en general respecto al mapa conocido del lugar, derivan en cambios en la forma en que se ejecuta el plan de cobertura. Esto puede resultar en áreas sin cubrir que no son suficientemente grandes para que el planificador de rutas de cobertura sea capaz de calcular una ruta para cubrirlas.

Los problemas mencionados anteriormente permiten vislumbrar distintas posibilidades de mejora para el sistema. Para tratar el problema de la dificultad para cubrir bordes y esquinas es posible agregar en el *behavior tree* desarrollando un paso extra antes de proceder a cubrir una sección del mapa. Este paso podría identificar bordes y esquinas, y calcular una estrategia para cubrir primero estas zonas problemáticas y posteriormente seguir con el cálculo de rutas y ejecución de estas para la zona. Al cubrir con antelación estas zonas problemáticas, el robot debería poder ejecutar mejor su tarea, ya que se podrían alcanzar porcentajes de *coverage* mas áltos. También podría ser posible realizar una implementación distinta del planificador local basado en aprendizaje reforzado, de modo que la formulación del problema tenga en cuenta estas dificultades y el robot aprenda a sortearlas de mejor manera. Por ejemplo, en esta formulación se podría incluir que el robot sea capaz de ejecutar movimientos en reversa lo cual le permitiría moverse con mayor libertad.

Tanto para los experimentos en simulación como en mundo real, se observó que durante la ejecución del plan en algunos casos se sale del área delimitada para cubrir. Dependiendo de los requerimientos específicos de la aplicación con la cual se quiera utilizar el sistema, este comportamiento podría ser no deseable. Una manera de corregir este comportamiento podría ser desarrollar

un láser virtual mediante *raycasting* 2D, de esta manera se genera un láser virtual que detecta los límites que se le dan a la zona de cobertura. Esto prevendrá que el robot se salga del espacio de cobertura ya que detectaría esta delimitación como un obstáculo y no como espacio libre.

Otro posible punto de mejora para el desempeño del sistema es realizar un suavizado de los planes obtenidos. Esto permitiría al robot cubrir con movimientos más simples el área, teniendo que realizar menos maniobras innecesarias. Esto puede ser de crítica importancia para bases móviles con baja maniobrabilidad, por ejemplo, tractores.

Por último, se propone para mejorar la cobertura obtenida, implementar un método que permita al robot calcular planes para las áreas pequeñas que puedan quedar sin cubrir y que con el sistema actual no es posible cubrir. Para permitir a los algoritmos poder calcular un plan para estas áreas es necesario que el tamaño del área sea al menos del tamaño del *footprint* del robot utilizado, por ende, sería necesario manipular el mapa del área cubierta de tal forma que el área a cubrir sea del tamaño mínimo necesario. Otra posibilidad para tratar este problema podría ser agregar memoria al robot respecto a los *waypoints* que no se pudieron ejecutar. Esto permitiría recordar al robot qué poses no fueron alcanzadas y luego intentar revisitarlas para cubrirlas. La dificultad que podría encontrarse en este caso tiene relación con la densidad de poses que entrega el plan. Si la densidad es baja, puede que al evadir un obstáculo no haya habido ningún *waypoint* por alcanzar, y por ende este método no conseguiría cubrir posteriormente esa área, al no haber ninguna pose asociada.

Bibliografía

[1] Christian Wögerer, Harald Bauer, Martijn N. Rooker, Gerhard Ebenhofer, Alberto Rovetta, Neil Martin Robertson, and Andreas Pichler. Locobot - low cost toolkit for building robot co-workers in assembly lines. In *International Conference on Intelligent Robotics and Applications*, 2012.

[2] Bryan Bridge, Tariq Pervez Sattar, and H. León-Rodriguez. Climbing robot cell for fast and flexible manufacture of large scale structures. 2006.

[3] Christian Vogel, Markus Fritzsche, and Norbert Elkmann. Safe human-robot cooperation with high-payload robots in industrial applications. *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 529–530, 2016.

[4] Yinchu Wang, Zhi He, Dandan Cao, Li Ma, Kai Li, Liangsheng Jia, and Yongjie Cui. Coverage path planning for kiwifruit picking robots based on deep reinforcement learning. *Comput. Electron. Agric.*, 205:107593, 2023.

[5] René Søndergaard Nilsson and Kun Zhou. Method and bench-marking framework for coverage path planning in arable farming. *Biosystems Engineering*, 198:248–265, 2020.

[6] Kalaivanan Sandamurthy and Kalpana Ramanujam. A hybrid weed optimized coverage path planning technique for autonomous harvesting in cashew orchards. *Information Processing in Agriculture*, 7:152–164, 2020.

[7] Xingguang Duan, Huanyu Tian, Changsheng Li, Zhe Han, Tengfei Cui, Qingxin Shi, Hao Wen, and Jin Wang. Virtual-fixture based drilling control for robot-assisted craniotomy: Learning from demonstration. *IEEE Robotics and Automation Letters*, 6:2327–2334, 2021.

[8] Gaoyuan Ma, Ming xing Lin, and Qingdong Wang. Mechanical design of a whole-arm exoskeleton rehabilitation robot based on pnf. *2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 777–780, 2016.

[9] Aravind Nehrujee, Hallel Andrew, . Reethajanetsurekha, Ann Patricia, Selvaraj Samuelkamaleshkumar, Henry Prakash, Srinivasan Sujatha, and Sivakumar Balasubramanian. Plug-and-train robot (pluto) for hand rehabilitation: Design and preliminary evaluation. *IEEE Access*, 9:134957–134971, 2020.

[10] C. Luo and Simon X. Yang. A bioinspired neural network for real-time concurrent map buil-

ding and complete coverage robot navigation in unknown environments. *IEEE Transactions on Neural Networks*, 19:1279–1298, 2008.

[11] Alejandro Marzinotto, Michele Colledanchise, Christian Smith, and Petter Ögren. Towards a unified behavior trees framework for robot control. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5420–5427, 2014.

[12] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: a factored solution to the simultaneous localization and mapping problem. In *AAAI/IAAI*, 2002.

[13] Raul Mur-Artal, José M. M. Montiel, and Juan D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31:1147–1163, 2015.

[14] Mathieu Labbé and François Michaud. Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36:416 – 446, 2018.

[15] Stergios I. Roumeliotis, Gaurav S. Sukhatme, and George A. Bekey. Circumventing dynamic modeling: evaluation of the error-state kalman filter applied to mobile robot localization. *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, 2:1656–1663 vol.2, 1999.

[16] Pi Yuzhen, Yuan Quande, and Zhang Benfa. The application of adaptive extended kalman filter in mobile robot localization. *2016 Chinese Control and Decision Conference (CCDC)*, pages 5337–5342, 2016.

[17] Leopoldo Jetto, Sauro Longhi, and Giuseppe Venturini. Development and experimental validation of an adaptive extended kalman filter for the localization of mobile robots. *IEEE Trans. Robotics Autom.*, 15:219–229, 1999.

[18] Shahram Rezaei and Raja Sengupta. Kalman filter-based integration of dgps and vehicle sensors for localization. *IEEE Transactions on Control Systems Technology*, 15:1080–1088, 2005.

[19] Motilal Agrawal and Kurt Konolige. Real-time localization in outdoor environments using stereo vision and inexpensive gps. *18th International Conference on Pattern Recognition (ICPR'06)*, 3:1063–1068, 2006.

[20] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte carlo localization for mobile robots. *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, 2:1322–1328 vol.2, 1999.

[21] Qibin Zhang, Peng-Cheng Wang, and Zonghai Chen. An improved particle filter for mobile robot localization based on particle swarm optimization. *Expert Syst. Appl.*, 135:181–193, 2019.

[22] Michael Montemerlo, Sebastian Thrun, and William Whittaker. Conditional particle filters for simultaneous mobile robot localization and people-tracking. *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, 1:695–701 vol.1,

2002.

[23] Jürgen Wolf, Wolfram Burgard, and Hans Burkhardt. Robust vision-based localization by combining an image-retrieval system with monte carlo localization. *IEEE Transactions on Robotics*, 21:208–216, 2005.

[24] Supriya Katwe, Nalini C. Iyer, Moin Khan, Mathew Peters, and Mahesh S. Mahale. Particle filter based localization of autonomous vehicle. *2021 2nd Global Conference for Advancement in Technology (GCAT)*, pages 1–6, 2021.

[25] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4:100–107, 1968.

[26] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[27] Brendan Englot and Franz S. Hover. Sampling-based coverage path planning for inspection of complex structures. *Proceedings of the International Conference on Automated Planning and Scheduling*, 2012.

[28] T. Bell, Michael O'Connor, V. K. Jones, Andrew K. Rekow, Gabriel Hugh Elkaim, and Bradford W. Parkinson. Realistic autofarming closed-loop tractor control over irregular paths using kinematic gps. *Journal of Navigation*, 51:327 – 335, 1998.

[29] Mark Ollis and Anthony Stentz. Vision-based perception for an automated harvester. *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robot and Systems. Innovative Robotics for Real-World Applications. IROS '97*, 3:1838–1844 vol.3, 1997.

[30] Zuo Llang Cao, Yuyu Huang, and Ernest L. Hall. Region filling operations with random obstacle avoidance for mobile robots. *J. Field Robotics*, 5:87–102, 1988.

[31] Ercan U. Acar, Howie Choset, Yangang Zhang, and Mark J. Schervish. Path planning for robotic demining: Robust sensor-based coverage of unstructured environments and probabilistic methods. *The International Journal of Robotics Research*, 22:441 – 466, 2003.

[32] Fumio Yasutomi, Makoto Yamada, and Kazuyoshi Tsukamoto. Cleaning robot control. *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pages 1839–1841 vol.3, 1988.

[33] Guoqing Hu, Zhengwei Hu, and Hongbo Wang. Complete coverage path planning for road cleaning robot. *2010 International Conference on Networking, Sensing and Control (ICNSC)*, pages 643–648, 2010.

[34] Peng Cheng, James F. Keller, and Vijay R. Kumar. Time-optimal uav trajectory planning for 3d urban structure coverage. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2750–2757, 2008.

[35] Enric Galceran and Marc Carreras. Planning coverage paths on bathymetric maps for in-detail inspection of the ocean floor. *2013 IEEE International Conference on Robotics and*

Automation, pages 4159–4164, 2013.

[36] Zongyuan Shen, Junnan Song, Khushboo Mittal, and Shalabh Gupta. Ct-cpp: Coverage path planning for 3d terrain reconstruction using dynamic coverage trees. *IEEE Robotics and Automation Letters*, 7:135–142, 2020.

[37] Hector I. A. Perez-Imaz, Paulo A. F. Rezeck, Douglas Guimarães Macharet, and Mario Fernando Montenegro Campos. Multi-robot 3d coverage path planning for first responders teams. *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 1374–1379, 2016.

[38] Howie Choset. Coverage for robotics – a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31:113–126, 2001.

[39] I. Z. Bouwer. *Algebraic Graph Theory (Norman Biggs)*. 1977.

[40] Merrill M. Flood. The traveling-salesman problem. *Operations Research*, 4:61–75, 1956.

[41] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, 4 edition, 2022.

[42] Jean-Claude Latombe. Robot motion planning. In *The Kluwer International Series in Engineering and Computer Science*, 1991.

[43] Howie Choset and Philippe Pignon. Coverage path planning: The boustrophedon cellular decomposition. 1998.

[44] Ercan U. Acar, Howie Choset, Alfred A. Rizzi, Prasad N. Atkar, and Douglas Hull. Morse decompositions for coverage tasks. *The International Journal of Robotics Research*, 21:331 – 344, 2002.

[45] Enrique González, Oscar Álvarez, Yul Díaz, Carlos Parra, and César Bustacara. Bsa: A complete coverage algorithm. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2040–2044, 2005.

[46] Yoav Gabriely and Elon D. Rimon. Spiral-stc: an on-line coverage algorithm of grid environments by a mobile robot. *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, 1:954–960 vol.1, 2002.

[47] C. Luo, Simon X. Yang, Deborah A. Stacey, and Jan C. Jofriet. Planning paths of complete coverage of an unstructured environment by a mobile robot. 2007.

[48] Vladimir J. Lumelsky, Snehasis Mukhopadhyay, and Kang Sun. Dynamic path planning in sensor-based terrain acquisition. *IEEE Trans. Robotics Autom.*, 6:462–472, 1990.

[49] Susan Hert, Sanjay Tiwari, and Vladimir J. Lumelsky. A terrain-covering algorithm for an auv. *Autonomous Robots*, 3:91–119, 1996.

[50] Yaru Kang and Dian xi Shi. A research on area coverage algorithm for robotics. *2018 IEEE*

International Conference of Intelligent Robotic and Control Engineering (IRCE), pages 6–13, 2018.

[51] C. Luo, Simon X. Yang, and Deborah A. Stacey. Real-time path planning with deadlock avoidance of multiple cleaning robots. *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, 3:4080–4085 vol.3, 2003.

[52] Noam Hazon and Gal A. Kaminka. Redundancy, efficiency and robustness in multi-robot coverage. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 735–741, 2005.

[53] C. Luo and S. X. Yang. A real-time cooperative sweeping strategy for multiple cleaning robots. *Proceedings of the IEEE Internatinal Symposium on Intelligent Control*, pages 660–665, 2002.

[54] Maxim A. Batalin and Gaurav S. Sukhatme. Spreading out: A local approach to multi-robot coverage. In *International Symposium on Distributed Autonomous Robotic Systems*, 2002.

[55] Kjerstin Easton and Joel W. Burdick. A coverage algorithm for multi-robot boundary inspection. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 727–734, 2005.

[56] Roland Y. Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots, Second Edition*. The MIT Press, 2011.

[57] Sedat Dogru and Lino Marques. A physics-based power model for skid-steered wheeled mobile robots. *IEEE Transactions on Robotics*, 34:421–433, 2018.

[58] Rameez Khan, Fahad Mumtaz Malik, Abid Raza, and Naveed Mazhar. Comprehensive study of skid-steer wheeled mobile robots: development and challenges. *Ind. Robot*, 48:142–156, 2020.

[59] Dominic Baril, Vincent Grondin, Simon-Pierre Deschênes, Johann Laconte, Maxime Vaidis, Vladimír Kubelka, André Gallant, Philippe Giguère, and F. Pomerleau. Evaluation of skid-steering kinematic models for subarctic environments. *2020 17th Conference on Computer and Robot Vision (CRV)*, pages 198–205, 2020.

[60] Michele Colledanchise and Petter Ögren. Behavior trees in robotics and ai: An introduction. *ArXiv*, abs/1709.00084, 2017.

[61] Marija Dakulovic, Sanja Horvatić, and Ivan Petrović. Complete coverage d* algorithm for path planning of a floor-cleaning mobile robot. *IFAC Proceedings Volumes*, 44:5950–5955, 2011.

[62] Anirudh Krishna Lakshmanan, Rajesh Elara Mohan, Balakrishnan Ramalingam, Anh Vu Le, Prabahar Veerajagadeshwar, Kamlesh Tiwari, and Muhammad Ilyas. Complete coverage path planning using reinforcement learning for tetromino based cleaning and maintenance robot. *Automation in Construction*, 112:103078, 2020.

[63] Mohamed Amine Yakoubi and Mohamed Tayeb Laskri. The path planning of cleaner robot for coverage region using genetic algorithms. *J. Innov. Digit. Ecosyst.*, 3:37–43, 2016.

[64] Jie Wu, Liang Cheng, Sensen Chu, and Yanjie Song. An autonomous coverage path planning algorithm for maritime search and rescue of persons-in-water based on deep reinforcement learning. *Ocean Engineering*, 2024.

[65] Nikolaos A. Kyriakakis, Magdalene Marinaki, Nikolaos F. Matsatsinis, and Yannis Marinakis. A cumulative unmanned aerial vehicle routing problem approach for humanitarian coverage path planning. *Eur. J. Oper. Res.*, 300:992–1004, 2021.

[66] Vishnu G. Nair and K. R. Guruprasad. Mr-simexcoverage: Multi-robot simultaneous exploration and coverage. *Comput. Electr. Eng.*, 85:106680, 2020.

[67] Sepideh Faghihi, Siavash Tavana, and Anton H. J. de Ruiter. Multiple spacecraft coordination and motion planning for full-coverage inspection of large complex space structures. *Acta Astronautica*, 2022.

[68] Zhexiong Shang, Justin M. Bradley, and Zhigang Shen. A co-optimal coverage path planning method for aerial scanning of complex structures. *Expert Syst. Appl.*, 158:113535, 2020.

[69] Chan Woo Jeon, Hak Jin Kim, Chang-Ho Yun, Xiongzhe Han, and Jung Hun Kim. Design and validation testing of a complete paddy field-coverage path planner for a fully autonomous tillage tractor. *Biosystems Engineering*, 208:79–97, 2021.

[70] Norawit Nangsue, Djitt Laowattana, Prakrankiat Youngkong, and Thavida Maneewarn. Complete coverage navigation for autonomous clay roller in salt-farming application. *2021 6th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*, pages 1–8, 2021.

[71] Mingxi Zhou, Jianguang Shi, and Lin Zhao. Towards the development of an online coverage path planner for uuv-based seafloor survey using an interferometric sonar. *2020 IEEE/OES Autonomous Underwater Vehicles Symposium (AUV)(50043)*, pages 1–5, 2020.

[72] Paolo Tripicchio, Matteo Unetti, Salvatore D'Avella, and Carlo Alberto Avizzano. Smooth coverage path planning for uavs with model predictive control trajectory tracking. *Electronics*, 2023.

[73] Gonzalo Mier, João Valente, and Sytze de Bruin. Fields2cover: An open-source coverage path planning library for unmanned agricultural vehicles. *IEEE Robotics and Automation Letters*, 8:2166–2172, 2022.

[74] Richard Bormann, Florian Jordan, Joshua Hampp, and Martin Hägele. Indoor coverage path planning: Survey, implementation, analysis. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1718–1725. IEEE, 2018.

[75] Daniel J. Rosenkrantz, Richard Edwin Stearns, and Philip M. Lewis. An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.*, 6:563–581, 1977.

[76] Noraini Mohd Razali and John Geraghty. Genetic algorithm performance with different se-

lection strategies in solving tsp. 2011.

[77] Fei Liu and Guangzhou Zeng. Study of genetic algorithm with reinforcement learning to solve the tsp. *Expert Syst. Appl.*, 36:6995–7001, 2009.

[78] Heinrich Braun. On solving travelling salesman problems by genetic algorithms. In *Parallel Problem Solving from Nature*, 1990.

[79] Petrică C. Pop, Ovidiu Cosma, C. Sabo, and Corina Pop Sitar. A comprehensive survey on the generalized traveling salesman problem. *European Journal of Operational Research*, 2023.

[80] S.X. Yang and C. Luo. A neural network approach to complete coverage path planning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34:718–724, 2004.

[81] Richard Bormann, Joshua Hampp, and Martin Hägele. New brooms sweep clean - an autonomous robotic cleaning assistant for professional office cleaning. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4470–4477, 2015.

[82] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artif. Intell.*, 99:21–71, 1998.

[83] Muhammad Asif Arain, Marcello Cirillo, Victor Manuel Hernández Bennetts, Erik Jan Schaffernicht, Marco Trincavelli, and Achim J. Lilienthal. Efficient measurement planning for remote gas sensing with mobile robots. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3428–3434, 2015.

[84] Francisco Leiva and Javier Ruiz del Solar. Robust rl-based map-less local planning: Using 2d point clouds as observations. *IEEE Robotics and Automation Letters*, 5:5787–5794, 2020.

[85] Giorgio Grisetti, C. Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23:34–46, 2007.

[86] Mark Pfeiffer, Samarth Shukla, Matteo Turchetta, César Cadena, Andreas Krause, Roland Y. Siegwart, and Juan I. Nieto. Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations. *IEEE Robotics and Automation Letters*, 3:4423–4430, 2018.

[87] Linhai Xie, Sen Wang, A. Markham, and Agathoniki Trigoni. Towards monocular vision based obstacle avoidance through deep reinforcement learning. *ArXiv*, abs/1706.09829, 2017.

[88] Esther M. Arkin, Sándor P. Fekete, and Joseph S.B. Mitchell. Approximation algorithms for lawn mowing and milling a preliminary version of this paper was entitled "the lawnmower problem" and appears in the proc. 5th canad. conf. comput. geom., waterloo, canada, 1993, pp.461–466. *Computational Geometry*, 17(1):25–50, 2000.

[89] Tingguang Li, Danny Ho, Chenming Li, Delong Zhu, Chaoqun Wang, and Max Q.-H. Meng. Houseexpo: A large-scale 2d indoor layout dataset for learning-based algorithms on mobile robots. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5839–5846, 2019.

Anexo A

Pruebas adicionales en el mundo real

A.1. Pruebas mundo real con robot Husky

La realización de estas pruebas se llevo a cabo en el área al costado del AMTC que se utilizo para las pruebas descritas en la Sección 5.3.3, se trabajo con una subsección especifica la cual se muestra en la figura 5.11. Las pruebas fueron ejecutadas con la plataforma móvil Husky[1] a la cual se equipó un sensor Ouster OS0. Esta plataforma al igual que el robot Panther es de tipo *skid-stereed*. Las dimensiones de su *footprint* son 990 [mm] de largo y 670[mm] de ancho, siendo entonces un poco mas angosto que la plataforma móvil Panther. En esta prueba, se utilizo el *stack* de navegación de `move_base` para la ejecución del plan de cobertura en el mundo real.

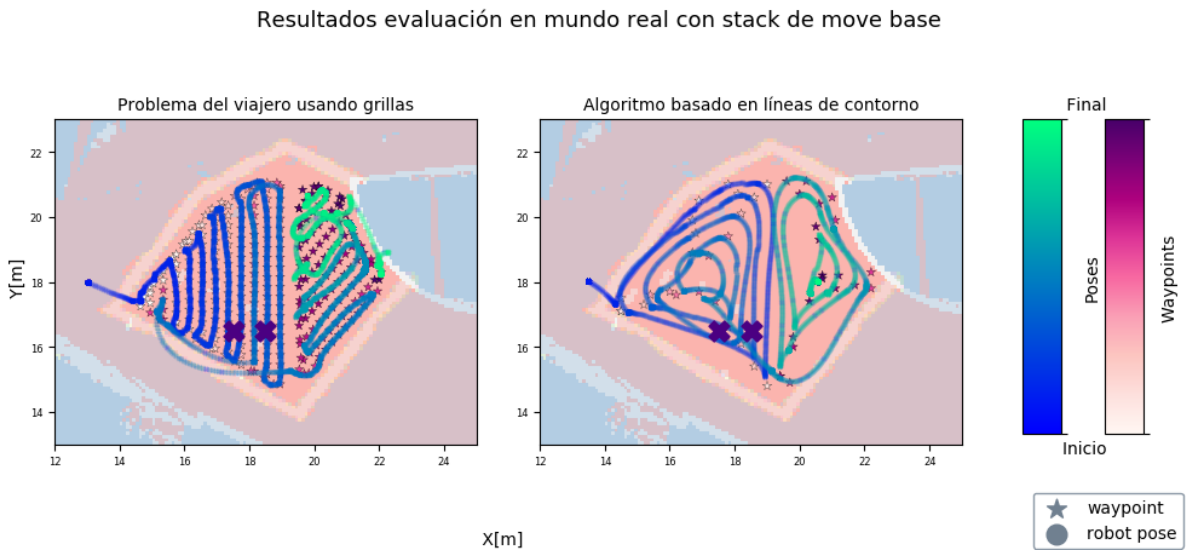


Figura A.1: Resultados obtenidos en el área de pileta AMTC utilizando como planificador global el algoritmo basado en grillas y ejecutado con la base móvil Husky. A la derecha se encuentra lo obtenido en el mundo real y a la izquierda a modo de comparación se muestra lo obtenido en simulación mediante el uso de localización *ground truth*.

[1]Información en detalle de esta base puede encontrase en `https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/`

La Figura A.1 muestra el resultado obtenido al ejecutar el sistema con el *stack* de `move_base`. Se observa que el robot sigue de cerca el plan de cobertura generado por el algoritmo de CPP, perdiendo pocos *waypoints*. Respecto a lo obtenido con estos algoritmos con el *stack* con planificador local basado en aprendizaje reforzado, se observa poca diferencia para el algoritmo 1, en términos de la trayectoria seguida. Para el algoritmo 6, se observa una diferencia mas marcada, en este caso la trayectoria se mantiene dentro del area a cubrir, en el caso del *stack* de RL, el robot en varias ocasiones salio de esta zona.

La realización de esta prueba permitió observar principalmente dos cosas. Lo primero es que el sistema es desplegable en otras plataformas móviles de tipo skid-stereed distintas al Panther mediante el ajuste de parámetros y en segundo lugar, permitió observar que el sistema utilizando el *stack* de `move_base` también es funcional en el mundo real.

A.2. Prueba en cancha 850, robot Panther

La realización de esta prueba se llevo a cabo en la cancha de futbol que se encuentra ubicada en la facultad de ciencias fisicas y matematicas de la universidad de chile, especificamente en el edifico Beauchef 850.

La base con la cual se despliega y su respectiva configuración es la misma utilizada para los experimentos vistos en la Sección 5.3.

En este caso la coberura obtenida alcanzo el valor de 99.38 %, tomando un tiempo de 52.08 minutos.

La realización de esta prueba permitió observar como se comporta el sistema en ambientes de exterior de gran tamaño. Se observa que el sistema es capaz de realizar la tarea de cobertura satisfactoriamente. Los resultados de esta prueba se pueden observar en la Figura A.2.

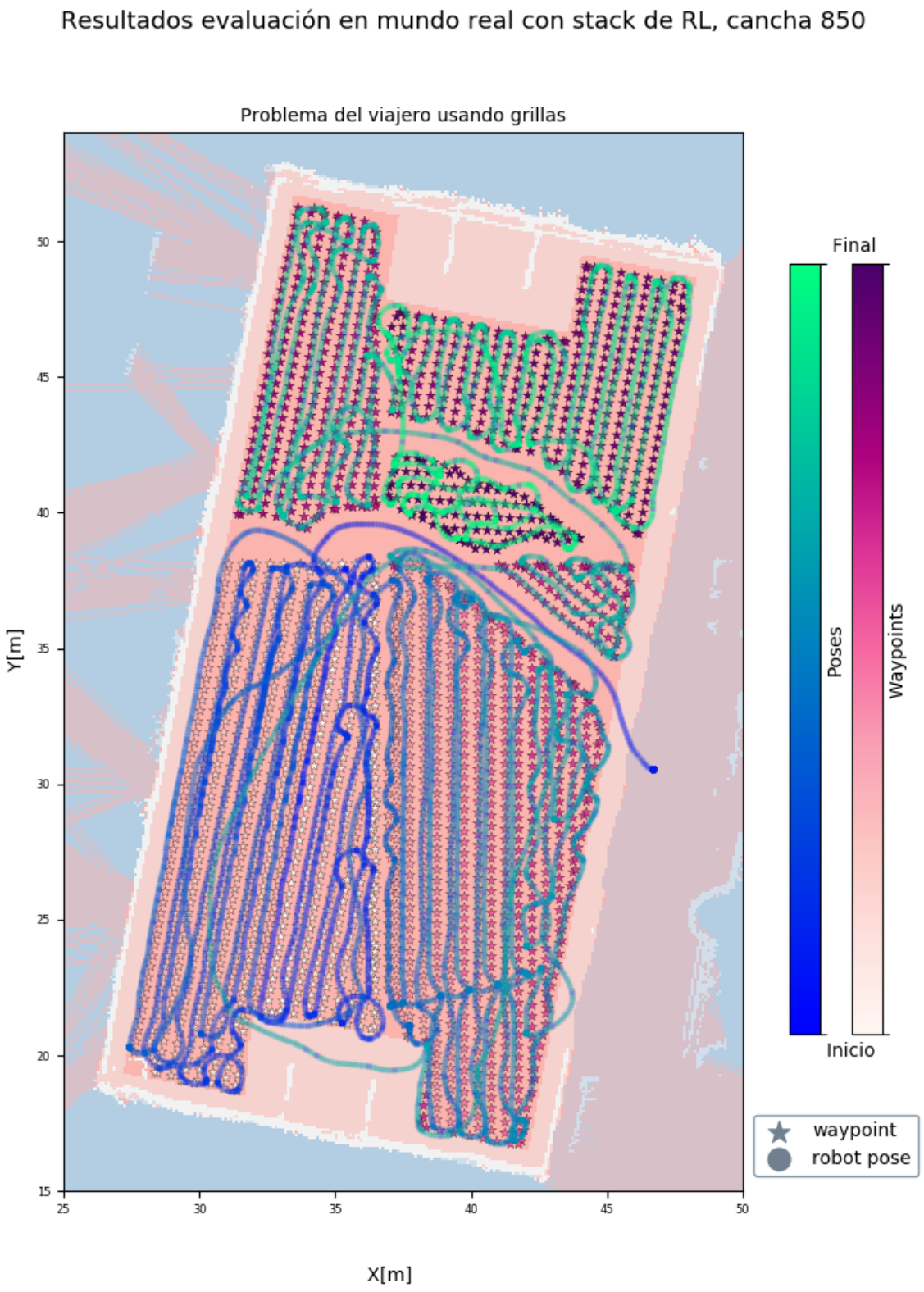


Figura A.2: Resultados obtenidos en el área de pileta AMTC utilizando como planificador global el algoritmo basado en grillas y ejecutado con la base móvil husky. A la derecha se encuentra lo obtenido en el mundo real y a la izquierda a modo de comparación se muestra lo obtenido en simulación mediante el uso de localización *ground truth*.